Error-repair Dependency Parsing for Ungrammatical Texts

Keisuke Sakaguchi[†] and **Matt Post**[‡] and **Benjamin Van Durme**[†‡]
[†]Center for Language and Speech Processing, Johns Hopkins University
[‡]Human Language Technology Center of Excellence, Johns Hopkins University
{keisuke,post,vandurme}@cs.jhu.edu

Abstract

We propose a new dependency parsing scheme which jointly parses a sentence and repairs grammatical errors by extending the non-directional transition-based formalism of Goldberg and Elhadad (2010) with three additional actions: SUBSTITUTE, DELETE, INSERT. Because these actions may cause an infinite loop in derivation, we also introduce simple constraints that ensure the parser termination. We evaluate our model with respect to dependency accuracy and grammaticality improvements for ungrammatical sentences, demonstrating the robustness and applicability of our scheme.

1 Introduction

Robustness has always been a desirable property for natural language parsers: humans generate a variety of noisy outputs, such as ungrammatical webpages, speech disfluencies, and the text in language learner's essays. Such *non-canonical* text contains grammatical errors such as substitutions, insertions, and deletions. For example, a non-native speaker of English might write *"*I look in forward hear from you"*, where *in* is inserted, *to* is deleted, and *hearing* is substituted incorrectly.

We propose a novel dependency parsing scheme that jointly parses and repairs ungrammatical sentences with these sorts of errors. The parser is based on the *non-directional easy-first* (EF) parser introduced by Goldberg and Elhadad (2010) (GE herein), which iteratively adds the most probable arc until the parse tree is completed. These actions are called ATTACHLEFT and ATTACHRIGHT depending on the direction of the arc. We extend the EF parsing scheme to be robust for ungrammatical inputs by correcting grammatical errors with three new actions: SUBSTITUTE, INSERT, and DELETE. These new actions do not add an arc between tokens but instead they edit a single token. As a result, the parser is able to jointly parse and correct grammatical errors in the input sentence. We call this new scheme *Error-Repair Non-Directional Easy-First parsing* (EREF). Since the new actions may greatly increase the search space (e.g., infinite substitutions), we also introduce simple constraints to avoid such issues.

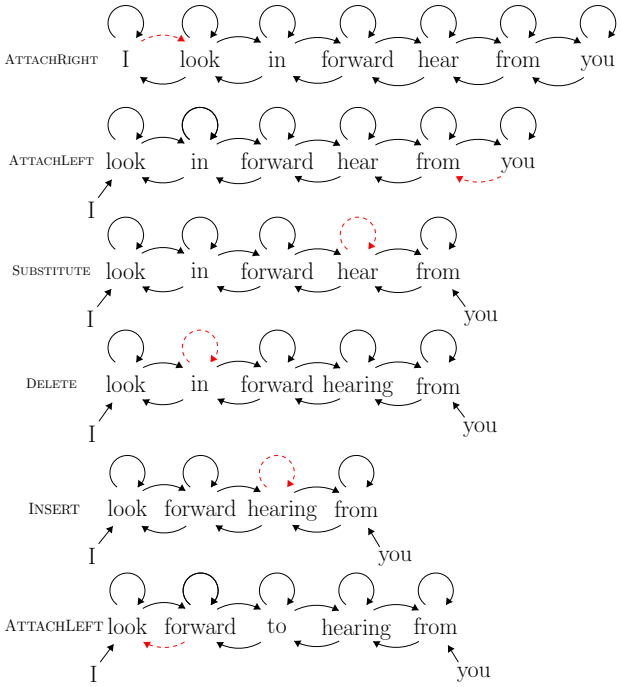

Figure 1: Illustrative example of partial derivation under error-repair easy-first non-directional dependency parsing. Solid arrows represent ATTACHRIGHT and ATTACHLEFT in Goldberg and Elhadad (2010). Dotted arcs correspond to actions for each step. Following the notation by GE: arcs are directed from a child to its parent.

We first describe the technical details of EREF (§2) and then evaluate our EREF parser with respect to dependency accuracy (robustness) and grammaticality improvements (§3). Finally, we

Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Short Papers), pages 189–195
Vancouver, Canada, July 30 - August 4, 2017.
https://doi.org/10.18653/v1/P17-2030

position this effort at the intersection of noisy text parsing and grammatical error correction (§4).

2 Model

Non-directional Easy-first Parsing Let us begin with a brief review of a non-directional easy-first (EF) parsing scheme proposed by GE, which is the foundation of our proposed scheme described in the following sections.

The EF parser has a list of partial structures p_1, ..., p_k (called *pending*) initialized with sentence tokens w_1, ..., w_n, and it keeps updating *pending* through derivations. Unlike left-to-right (e.g., shift-reduce) parsing algorithms (Yamada and Matsumoto, 2003; Nivre, 2004), EF iteratively selects the best pair of adjoining tokens and chooses the direction of attachment: ATTACHLEFT or ATTACHRIGHT. Once the action is committed, the corresponding dependency arc is added and the child token is removed from *pending*. The first two derivations in Figure 1 depict ATTACHRIGHT and ATTACHLEFT. Pseudocode is shown in Algorithm 1 (lines 1, 3-12).

The parser is trained using the structured perceptron (Collins, 2002) to choose actions to take given a set of features expanded from templates. The cost of actions is computed at every step by checking the *validity*: whether a new arc is included in the gold parse and whether the child already has all its children. See GE for further description of feature templates and structured perceptron training. Since it is possible that there are multiple *valid* sequence of actions and it is important to examine a large search space, the parser is allowed to explore (possibly incorrect) actions with a certain probability, termed *learning with exploration* by Goldberg and Nivre (2013).

Error-repair variant of EF Error-repair non-directional easy-first parsing scheme (EREF) is a variant of EF. We add three new actions: SUBSTITUTE, DELETE, INSERT as $Acts_{\text{ER}}$. We do not deal with a swapping action (Nivre, 2009) to deal with word reordering errors, since the errors are even less frequent than other error types (Leacock et al., 2014). SUBSTITUTE replaces a token to a grammatically more probable token, DELETE removes an unnecessary token, and INSERT inserts a new token at a designated index. These actions are shown in Figure 1 and Algorithm 1 (lines 13-25). Because the length of *pending* decreases as an attachment occurs, the parser also keeps the token indices in *repaired* (line 5), which holds all tokens in a sentence throughout the parsing process. Furthermore, the parser updates token indices in *pending* and *repaired* when INSERT or DELETE occurs. Technically, when a token at i is deleted/inserted, the parser decrements/increments the indices that are $k >= i$ (before executing the action) in *pending*, *repaired*, and parents and children in a (partial) dependency tree (*Arcs*).

To find the best candidate for SUBSTITUTE and INSERT efficiently, we restrict candidates to the same part-of-speech or pre-defined candidate list. We select the best candidate by comparing each n-gram language model score with the same surrounding context.

Similar to EF, while training the parser, the cost

Algorithm 1: Error-repair non-directional parsing

Input: ungrammatical sentence= w_1 ... w_n
Output: a set of dependency arcs (*Arcs*), repaired sentence ($\hat{w}_1$ ... $\hat{w}_m$)

1 *Acts* = { ATTACHLEFT, ATTACHRIGHT }
2 $Acts_{\text{ER}}$ = { DELETE, INSERT, SUBSTITUTE }
3 *Arcs* = { }
4 *pending* = $p_1 ... p_n \leftarrow w_1 ... w_n$
5 *repaired* = $\hat{w}_1 ... \hat{w}_n \leftarrow w_1 ... w_n$
6 **while** *len (pending)* > 1 **do**
7 $best \leftarrow \underset{act \in Acts \cup Acts_{\text{ER}}}{\text{argmax}} score\ (act\ (i))$
8 s.t. $1 \leq i \leq len(pending) \cap isLegal(act, pending)$
9 **if** *best* $\in$ *Acts* **then**
10 (*parent*, *child*) $\leftarrow$ *edgeFor*(*best*)
11 *Arcs*.add((*parent*, *child*))
12 *pending*.remove(*child*)
13 **else if** *best* = SUBSTITUTE **then**
14 c = *bestCandidate*(*best*, *repaired*)
15 *pending*.replace(p_i, c)
16 *repaired*.replace($\hat{w}_{p_i.idx}$, c)
17 **else if** *best* = DELETE **then**
18 *pending*.remove(p_i)
19 *repaired*.remove($\hat{w}_{p_i.idx}$)
20 *Arcs*.updateIndex()
21 **else if** *best* = INSERT **then**
22 c = *bestCandidate*(*best*, *repaired*)
23 *pending*.insert(i, c)
24 *repaired*.insert($p_i.idx$, c)
25 *Arcs*.updateIndex()
26 **end**
27 **return** *Arcs, repaired*

Algorithm 2: Check validity during training

```
1 Function isValid(act, repaired, Gold)
2     d_before = editDistance(repaired, Gold)
3     repaired+ = repaired.apply(act)
4     d_after = editDistance(repaired+, Gold)
5     if d_before > d_after then return true;
6     else return false;
7 end
```

for $Acts_{\text{ER}}$ is based on *validity*. The *validity* of the new actions is computed by taking the edit distance (d) between the *Gold* tokens (w_1^* ... w_r^*) and the sentence state that the parser stores in *repaired* ($\hat{w}_1$ ... $\hat{w}_m$). When the edit distance after taking an action (d_{after}) is smaller than before (d_{before}), we regard the action as *valid* (Algorithm 2).

One serious concern of EREF is that the new actions may cause an infinite loop during parsing (e.g., infinite SUBSTITUTE, or an alternative DELETE and INSERT sequence.). To avoid this, we introduce two constraints: (1) *edit flag* and (2) *edit limit*. *Edit flag* is assigned for each token as a property, and a parser is not allowed to execute $Acts_{\text{ER}}$ on a token if its flag is on. The flag is turned on when a parser executes $Acts_{\text{ER}}$ on a token whose flag is off. In INSERT action, the flag of the inserted token is activated, while the subsequent token (which gave rise to the INSERT) is not. *Edit limit* is set to be the number of tokens in a sentence, and the parser is not allowed to perform $Acts_{\text{ER}}$ when the total number of execution of $Acts_{\text{ER}}$ exceeds the limit. These two constraints prevent the parser from falling into an infinite loop as well as parsing in the same order of time complexity as GE. We also add the following constraints to avoid unreasonable derivations: (i) a word with a dependent cannot be deleted and (ii) any child words cannot be substituted. All the constraints are implemented in the *isLegal*() function in Algorithm 1 (line 8). We note that these constraints not only prevent undesirable derivations but also leads to an efficiency in exploring the search space during training.

3 Experiment

Data and Evaluation We evaluate EREF with respect to dependency parsing accuracy (Exp1) and grammaticality improvement (Exp2).[1]

[1]Code for the experiments is available at http://github.com/keisks/error-repair-parsing

In the first experiment, as in GE, we train and evaluate our parser on the English dataset from the Penn Treebank (Marcus et al., 1993) with the Penn2Malt conversion program (Sections 2-21 for training, 22 for tuning, and 23 for test). We use the PTB for the dependency experiment, since there are no ungrammatical text corpora that has dependency annotation on the *corrected* texts by human.

We choose the following most frequent error types that are used in CoNLL 2013 shared task (Ng et al., 2013):

1. Determiner (substitution, deletion, insertion)
2. Preposition (substitution, deletion, insertion)
3. Noun number (singular vs. plural)
4. Verb form (tense and aspect)
5. Subject verb agreement

Regarding the candidate sets for INSERT and SUBSTITUTE actions, following Rozovskaya and Roth (2014), we focus on the most common candidates for each error type, setting the determiner candidates to be {*a, an, the,* ϕ (as deletion)}, preposition candidates to be {*on, about, from, for, of, to, at, in, with, by,* ϕ}, and verb forms to be {VB(P|Z|G|D|N)}. We build a 5-gram language model on English Gigaword with the KenLM Toolkit (Heafield, 2011) for EREF to select the best candidate.

We manually inject grammatical errors into PTB with certain error-rates similarly to the GenERRate toolkit by Foster and Andersen (2009), which is designed to create synthetic errors into sentences to improve grammatical error detection.

We train and tune EREF models with different token-level error injection rates from 5% (E05) to 20% (E20), because language learner corpora have generally around 5% to 15% of token level errors depending on learners' proficiency (Leacock et al., 2014). Since the error injection is stochastic, we train each model with 10 runs and take an average of parser performance on the test set.

As a baseline, we use the original parser as described by GE, which is equivalent to EREF with training on an error-free corpus (E00). Since the EF baseline does not allow error correction during parsing, we pre-process the test data with a grammatical error correction system similar to Rozovskaya and Roth (2014), where a combination of classifiers for each error type corrects grammatical errors.

For evaluation, we jointly parse and correct grammatical errors in the test set with different

(%)	Baseline	E05	E10	E15	E20
0	**91.43**	91.12	90.87	90.61	90.29
5	89.99	**90.00**	89.87	89.72	89.48
10	87.84	87.99	88.07	**88.14**	88.04
15	85.64	86.18	86.54	86.75	**86.82**
20	84.12	84.78	85.28	85.50	**85.76**
∇	-0.37	-0.32	-0.28	-0.26	**-0.23**

Table 1: Unlabeled dependency accuracy results with the 5x5 models and test sets. ∇ shows the slope of deterioration in parser performance.

	E05	E10	E15	E20
# edited sents (out of 5,124)	175	391	583	861
grammaticality (source)	2.92	2.95	2.95	2.89
grammaticality (this work)	2.96	2.99	**3.27**	**2.98**

Table 2: Grammaticality scores by 1-4 scale regression model (Heilman et al., 2014). The first row shows the number of sentences that are made (at least one) change. Bold numbers show statistically significant improvements.

error injection rates (from 0% to 20%). It is important to note that the number of tokens between the parser output and the oracle may be different because of error injection into the test set and $Acts_{\text{ER}}$ during parsing. To handle this mismatch, we evaluate the dependency accuracy with alignment (Favre et al., 2010) in the spirit of SParseval (Roark et al., 2006), where tokens between a hypothesis and oracle are aligned prior to calculating the dependency accuracy.

In the second experiment, we use the Treebank of Learner English (TLE) (Berzak et al., 2016) to see the grammaticality improvement in a real scenario. TLE contains 5,124 sentences and 2.69 (std 1.9) token errors per sentence. The average sentence length is 19.06 (std 9.47). TLE also provides dependency labels and POS tags on the *raw (ungrammatical)* sentences. It is important to note that TLE has dependency annotation only for the original *ungrammatical* sentences, and therefore we do not compute the accuracy of dependency parse in this experiment. Since the corpus size is small, we train EREF (E05 to E20) on 100k sentences from Annotated Gigaword (Napoles et al., 2012) and used TLE as a test set. Spelling errors are ignored because EREF can use the POS information. Grammaticality is evaluated by a regression model (Heilman et al., 2014), which scores grammaticality on the ordinal scale (from 1 to 4).

Results Table 1 shows the result of unlabeled dependency accuracy (UAS).[2] As previously presented (Foster, 2007; Cahill, 2015), our experiment also shows that parser performance deteriorated as the error rate in the test corpus increased. On the error-free test set (0%), the baseline (EF pipeline) outperforms other EREF models; the accuracy is lower when the parser is trained on noisier data. The difference among the models becomes small when the test set has 10% error injection rate. As the rate increases further, the trend of parser accuracy reverses. When the test set has 15% or higher noise, the E20 is the most accurate parser. This trend is presented by the slope of deterioration $\nabla = \frac{\text{accuracy}_{20\%} - \text{accuracy}_{0\%}}{20}$ in Table 1; a parser trained on noisier training data shows smaller decline and more robustness.[3] This indicates that the EREF is more robust than the vanilla EF on ungrammatical texts by jointly parsing and correcting errors.

Successful cases
I 'm looking forward to [-see-] {+seeing+} you next summer
I 've never [-approve-] {+approved+} his deal
There is not {+a+} possibility to travel
Failure cases
I 've [-assisted-] {+assisting+} your new musical show
I am writing to complain [-about-] {+with+} somethings
I hope you liked {+the+} everything I said

Table 3: Successful and failure examples by EREF. The edits are represented by [-deletion-] and {+insertion+}. Adjacent pairs of deletion and insertion are considered as substitution.

Table 2 demonstrates the result of grammaticality improvement (1-4 scale) on the TLE corpus, and Table 3 shows successful and failure corrections by EREF. Minimally trained models (E05 and E10) show little improvement in grammaticality because the models are too conservative to make edits. The models with higher error-injection rates (E15 and E20) achieve 0.1 to 0.3 improvements that are statistically significant. There is still room to improve regarding the amount of corrections. This is probably because TLE contains a variety of errors (e.g., collocation, punctuation) in addition to the five error types we focus. To deal with other error types, we can extend EREF by adding more actions, although it increases the search space.

From a practical perspective, the level of ungrammaticality should be realized ahead of time. This is an issue to be addressed in future research.

[2] Technically, it is possible to train the model with learning labels simultaneously (LAS), but there is a trade-off between search space and training time. The primary goal of this experiment is to see if the EREF is able to detect and correct grammatical errors.

[3] Baseline model *without* preprocessing always underperformed the preprocessed baseline.

4 Related Work

Our work lies at the intersection of parsing non-canonical texts and grammatical error correction.

Joint dependency parsing and disfluency detection has been pursued (Rasooli and Tetreault, 2013, 2014; Honnibal and Johnson, 2014; Wu et al., 2015; Yoshikawa et al., 2016), where a parser jointly parses and detects disfluency (e.g., reparandum and interregnum) for a given speech utterance. Our work could be considered an extension via adding SUBSTITUTE and INSERT actions, although we depend on easy-first non-directional parsing framework instead of a left-to-right strategy. Importantly, the DELETE action is easier to handle than the SUBSTITUTE and INSERT actions, because they bring us challenging issues about a process of candidate word generation and avoiding an infinite loop in derivation. We have addressed these issues as explained in §2.

In terms of the literature from grammatical error correction, this work is closely related to Dahlmeier and Ng (2012), where they show an error correction decoder with the easy-first strategy. The decoder iteratively corrects the most probable ungrammatical token by applying different classifiers for each error type. The EREF parser also depends on the easy-first strategy to find ungrammatical index to be deleted, inserted, or substituted, but it parses and corrects errors jointly whereas the decoder is designed as a grammatical error correction framework rather than a parser.

There is a line of work for parsing ungrammatical sentences (e.g., web forum) by adapting an *existing* parsing scheme on domain specific annotations (Petrov and McDonald, 2012; Cahill, 2015; Berzak et al., 2016; Nagata and Sakaguchi, 2016). Although we share an interest with respect to dealing with ungrammatical sentences, EREF focuses on the parsing scheme for repairing grammatical errors instead of adapting a parser with a domain specific annotation scheme.

More broadly, our work can also be regarded as one of the joint parsing and text normalization tasks such as joint spelling correction and POS tagging (Sakaguchi et al., 2012), word segmentation and POS tagging (Kaji and Kitsuregawa, 2014; Qian et al., 2015).

5 Conclusions

We have presented an *error-repair* variant of the *non-directional easy-first* dependency parser. We have introduced three new actions, SUBSTITUTE, INSERT, and DELETE into the parser so that it jointly parses and corrects grammatical errors in a sentence. To address the issue of parsing incompletion due to the new actions, we have proposed simple constraints that keep track of editing history for each token and the total number of edits during derivation. The experimental result has demonstrated robustness of EREF parsers against EF and grammaticality improvement. Our work is positioned at the intersection of noisy text parsing and grammatical error correction. The EREF is a flexible formalism not only for grammatical error correction but other tasks with jointly editing and parsing a given sentence.

Acknowledgments

This work was supported in part by the JHU Human Language Technology Center of Excellence (HLTCOE), and DARPA LORELEI. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes. The views and conclusions contained in this publication are those of the authors and should not be interpreted as representing official policies or endorsements of DARPA or the U.S. Government.

References

Yevgeni Berzak, Jessica Kenney, Carolyn Spadine, Jing Xian Wang, Lucia Lam, Keiko Sophie Mori, Sebastian Garza, and Boris Katz. 2016. Universal dependencies for learner english. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 737–746.

Aoife Cahill. 2015. Parsing learner text: to shoehorn or not to shoehorn. In *Proceedings of The 9th Linguistic Annotation Workshop*. Association for Computational Linguistics, Denver, Colorado, USA, pages 144–147.

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*. Association for Computational Linguistics, pages 1–8.

Daniel Dahlmeier and Hwee Tou Ng. 2012. A beam-search decoder for grammatical error correction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, Jeju Island, Korea, pages 568–578.

Benoit Favre, Bernd Bohnet, and Dilek Hakkani-Tür. 2010. Evaluation of semantic role labeling and dependency parsing of automatic speech recognition output. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*. pages 5342–5345.

Jennifer Foster. 2007. Treebanks gone bad. *International Journal of Document Analysis and Recognition (IJDAR)* 10(3):129–145.

Jennifer Foster and Oistein Andersen. 2009. Generrate: Generating errors for use in grammatical error detection. In *Proceedings of the Fourth Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics, Boulder, Colorado, pages 82–90.

Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, Los Angeles, California, pages 742–750.

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *Transactions of the Association for Computational Linguistics* 1:403–414.

Kenneth Heafield. 2011. KenLM: faster and smaller language model queries. In *Proceedings of the EMNLP 2011 Sixth Workshop on Statistical Machine Translation*. Edinburgh, Scotland, United Kingdom, pages 187–197.

Michael Heilman, Aoife Cahill, Nitin Madnani, Melissa Lopez, Matthew Mulholland, and Joel Tetreault. 2014. Predicting grammaticality on an ordinal scale. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, Baltimore, Maryland, pages 174–180.

Matthew Honnibal and Mark Johnson. 2014. Joint incremental disfluency detection and dependency parsing. *Transactions of the Association for Computational Linguistics* 2:131–142.

Nobuhiro Kaji and Masaru Kitsuregawa. 2014. Accurate word segmentation and pos tagging for japanese microblogs: Corpus annotation and joint modeling with lexical normalization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, pages 99–109.

Claudia Leacock, Martin Chodorow, Michael Gamon, and Joel Tetreault. 2014. Automated grammatical error detection for language learners. *Synthesis lectures on human language technologies* 7(1):1–170.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics* 19(2):313–330.

Ryo Nagata and Keisuke Sakaguchi. 2016. Phrase structure annotation and parsing for learner english. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Berlin, Germany, pages 1837–1847.

Courtney Napoles, Matthew Gormley, and Benjamin Van Durme. 2012. Annotated gigaword. In *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*. Association for Computational Linguistics, pages 95–100.

Hwee Tou Ng, Siew Mei Wu, Yuanbin Wu, Christian Hadiwinoto, and Joel Tetreault. 2013. The conll-2013 shared task on grammatical error correction. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, Sofia, Bulgaria, pages 1–12.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*. Association for Computational Linguistics, Stroudsburg, PA, USA, IncrementParsing '04, pages 50–57.

Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. Association for Computational Linguistics, Suntec, Singapore, pages 351–359.

Slav Petrov and Ryan McDonald. 2012. Overview of the 2012 shared task on parsing the web. *Notes of the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL)* .

Tao Qian, Yue Zhang, Meishan Zhang, Yafeng Ren, and Donghong Ji. 2015. A transition-based model for joint segmentation, pos-tagging and normalization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Lisbon, Portugal, pages 1837–1846.

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*. Association for Computational Linguistics, Seattle, Washington, USA, pages 124–129.

Mohammad Sadegh Rasooli and Joel Tetreault. 2014. Non-monotonic parsing of fluent umm i mean disfluent sentences. In *Proceedings of the 14th Conference of the European Chapter of the Association for*

Computational Linguistics, volume 2: Short Papers. Association for Computational Linguistics, Gothenburg, Sweden, pages 48–53.

Brian Roark, Mary Harper, Eugene Charniak, Bonnie Dorr, Mark Johnson, Jeremy Kahn, Yang Liu, Mari Ostendorf, John Hale, Anna Krasnyanskaya, Matthew Lease, Izhak Shafran, Matthew Snover, Robin Stewart, and Lisa Yung. 2006. Sparseval: Evaluation metrics for parsing speech. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*. European Language Resources Association (ELRA).

Alla Rozovskaya and Dan Roth. 2014. Building a state-of-the-art grammatical error correction system. *Transactions of the Association for Computational Linguistics* 2:414–434.

Keisuke Sakaguchi, Tomoya Mizumoto, Mamoru Komachi, and Yuji Matsumoto. 2012. Joint English spelling error correction and POS tagging for language learners writing. In *Proceedings of COLING 2012*. The COLING 2012 Organizing Committee, Mumbai, India, pages 2357–2374.

Shuangzhi Wu, Dongdong Zhang, Ming Zhou, and Tiejun Zhao. 2015. Efficient disfluency detection with transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, Beijing, China, pages 495–503.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *In Proceedings of IWPT*. pages 195–206.

Masashi Yoshikawa, Hiroyuki Shindo, and Yuji Matsumoto. 2016. Joint transition-based dependency parsing and disfluency detection for automatic speech recognition texts. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Austin, Texas, pages 1036–1041.