Efficient Disfluency Detection with Transition-based Parsing

Shuangzhi Wu[†] **, Dongdong Zhang**[‡] **, Ming Zhou**[‡] **, Tiejun Zhao**[†]
[†]Harbin Institute of Technology
[‡]Microsoft Research
{v-shuawu, dozhang, mingzhou}@microsoft.com
tjzhao@hit.edu.cn

Abstract

Automatic speech recognition (ASR) outputs often contain various disfluencies. It is necessary to remove these disfluencies before processing downstream tasks. In this paper, an efficient disfluency detection approach based on right-to-left transition-based parsing is proposed, which can efficiently identify disfluencies and keep ASR outputs grammatical. Our method exploits a global view to capture long-range dependencies for disfluency detection by integrating a rich set of syntactic and disfluency features with linear complexity. The experimental results show that our method outperforms state-of-the-art work and achieves a 85.1% f-score on the commonly used English Switchboard test set. We also apply our method to in-house annotated Chinese data and achieve a significantly higher f-score compared to the baseline of CRF-based approach.

1 Introduction

With the development of the mobile internet, speech inputs have become more and more popular in applications where automatic speech recognition (ASR) is the key component to convert speech into text. ASR outputs often contain various disfluencies which create barriers to subsequent text processing tasks like parsing, machine translation and summarization. Usually, disfluencies can be classified into uncompleted words, filled pauses (e.g. "uh", "um"), discourse markers (e.g. "I mean"), editing terms (e.g. "you know") and repairs. To identify and remove disfluencies, straightforward rules can be designed to tackle the former four classes of disfluencies since they often belong to a closed set. However, the repair type disfluency poses particularly more difficult problems as their form is more arbitrary. Typically, as shown in Figure 1, a repair disfluency type consists of a reparandum ("to Boston") and a filled pause ("um"), followed by its repair ("to Denver"). This special structure of disfluency constraint, which exists in many languages such as English and Chinese, reflects the scenarios of spontaneous speech and conversation, where people often correct preceding words with following words when they find that the preceding words are wrong or improper. This procedure might be interrupted and inserted with filled pauses when people are thinking or hesitating. The challenges of detecting repair disfluencies are that reparandums vary in length, may occur everywhere, and are sometimes nested.

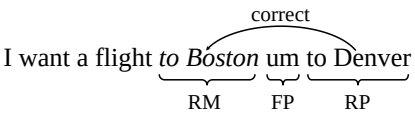

Figure 1: A typical example of repair type disfluency consists of FP (Filled Pause), RM (Reparandum), and RP (Repair). The preceding RM is corrected by the following RP.

There are many related works on disfluency detection, that mainly focus on detecting repair type of disfluencies. Straightforwardly, disfluency detection can be treated as a sequence labeling problem and solved by well-known machine learning algorithms such as conditional random fields (CRF) or max-margin markov network (M^3N) (Liu et al., 2006; Georgila, 2009; Qian and Liu, 2013), and prosodic features are also concerned in (Kahn et al., 2005; Zhang et al., 2006). These methods achieve good performance, but are not powerful enough to capture complicated disfluencies with longer spans or distances. Recently, syntax-based models such as transition-based parser have been used for detecting disflu-

Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, pages 495–503,
Beijing, China, July 26-31, 2015. ©2015 Association for Computational Linguistics

encies (Honnibal and Johnson, 2014; Rasooli and Tetreault, 2013). These methods can jointly perform dependency parsing and disfluency detection. But in these methods, great efforts are made to distinguish normal words from disfluent words as decisions cannot be made imminently from left to right, leading to inefficient implementation as well as performance loss.

In this paper, we propose detecting disfluencies using a right-to-left transition-based dependency parsing (R2L parsing), where the words are consumed from right to left to build the parsing tree based on which the current word is predicted to be either disfluent or normal. The proposed models cater to the disfluency constraint and integrate a rich set of features extracted from contexts of lexicons and partial syntactic tree structure, where the parsing model and disfluency predicting model are jointly calculated in a cascaded way. As shown in Figure 2(b), while the parsing tree is being built, disfluency tags are predicted and attached to the disfluency nodes. Our models are quite efficient with linear complexity of $2 * N$ (N is the length of input).

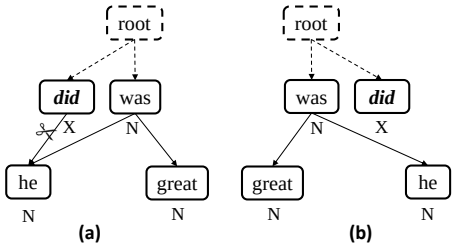

Figure 2: An instance of the detection procedure where 'N' stands for a normal word and 'X' a disfluency word. Words with italic font are Reparandums. (a) is the L2R detecting procedure and (b) is the R2L procedure.

Intuitively, compared with previous syntax-based work such as (Honnibal and Johnson, 2014) that uses left-to-right transition-based parsing (L2R parsing) model, our proposed approach simplifies disfluency detection by sequentially processing each word, without going back to modify the pre-built tree structure of disfluency words. As shown in Figure 2(a), the L2R parsing based joint approach needs to cut the pre-built dependency link between "did" and "he" when "was" is identified as the repair of "did", which is never needed in our method as Figure 2(b). Furthermore, our method overcomes the deficiency issue in decoding of L2R parsing based joint method, meaning the number of parsing transitions for each hypothesis path is not identical to $2 * N$, which leads to the failure of performing optimal search during decoding. For example, the involvement of the extra cut operation in Figure 2(a) destroys the competition scoring that accumulates over $2 * N$ transition actions among hypotheses in the standard transition-based parsing. Although the heuristic score, such as the normalization of transition count (Honnibal and Johnson, 2014), can be introduced, the total scores of all hypotheses are still not statistically comparable from a global view.

We conduct the experiments on English Switchboard corpus. The results show that our method can achieve a 85.1% f-score with a gain of 0.7 point over state-of-the-art M^{3}N labeling model in (Qian and Liu, 2013) and a gain of 1 point over state-of-the-art joint model proposed in (Honnibal and Johnson, 2014). We also apply our method on Chinese annotated data. As there is no available public data in Chinese, we annotate 25k Chinese sentences manually for training and testing. We achieve 71.2% f-score with 15 points gained compared to the CRF-based baseline, showing that our models are robust and language independent.

2 Transition-based dependency parsing

In a typical transition-based parsing, the Shift-Reduce decoding algorithm is applied and a queue and stack are maintained (Zhang and Clark, 2008). The queue stores the stream of the input and the front of the queue is indexed as the current word. The stack stores the unfinished words which may be linked to the current word or a future word in the queue. When words in the queue are consumed in sequential order, a set of transition actions is applied to build a parsing tree. There are four kinds of transition actions in the parsing process (Zhang and Clark, 2008), as described below.

- *Shift* : Removes the front of the queue and pushes it to the stack.
- *Reduce* : Pops the top of the stack.
- *LeftArc* : Pops the top of the stack, and links the popped word to the front of the queue.
- *RightArc* : Links the front of the queue to the top of the stack and, removes the front of the queue and pushes it to the stack.

The choice of each transition action during parsing is scored by a generalized perceptron (Collins,

2002) which can be trained over a rich set of non-local features. In decoding, beam search is performed to search the optimal sequence of transition actions. As each word must be pushed to the stack once and popped off once, the number of actions needed to parse a sentence is always $2 * N$, where N is the length of the sentence.

Transition-based dependency parsing (Zhang and Clark, 2008) can be performed in either a left-to-right or a right-to-left way, both of which have a performance that is comparable as illustrated in Section 4. However, when they are applied to disfluency detection, their behaviors are very different due to the disfluency structure constraint. We prove that right-to-left transition-based parsing is more efficient than left-to-right transition-based parsing for disfluency detection.

3 Our method

3.1 Model

Unlike previous joint methods (Honnibal and Johnson, 2014; Rasooli and Tetreault, 2013), we introduce dependency parsing into disfluency detection from theory. In the task of disfluency detection, we are given a stream of unstructured words from automatic speech recognition (ASR). We denote the word sequence with $W_1^n := w_1$, w_2,w_3,...,w_n, which is actually the inverse order of ASR words that should be w_n, w_{n-1},w_{n-2},...,w_1. The output of the task is a sequence of binary tags denoted as $D_1^n = d_1$, d_2,d_3,...,d_n, where each d_i corresponds to w_i, indicating whether w_i is a disfluency word (X) or not (N).[1]

Our task can be modeled as formula (1), which is to search the best sequence D^* given the stream of words W_1^n.

$$D^* = argmax_D P(D_1^n|W_1^n) \quad (1)$$

The dependency parsing tree is introduced into model (1) to guide detection. The rewritten formula is shown below:

$$D^* = argmax_D \sum_T P(D_1^n, T|W_1^n) \quad (2)$$

We jointly optimize disfluency detection and parsing with form (3), rather than considering all possible parsing trees:

$$(D^*, T^*) = argmax_{(D,T)} P(D_1^n, T|W_1^n) \quad (3)$$

[1]We just use tag 'N' to represent a normal word, in practice normal words will not be tagged anything by default.

As both the dependency tree and the disfluency tags are generated word by word, we decompose formula (3) into:

$$(D^*, T^*) = argmax_{(D,T)} \prod_{i=1}^{n} P(d_i, T_1^i|W_1^i, T_1^{i-1}) \quad (4)$$

where T_1^i is the partial tree after word w_i is consumed, d_i is the disfluency tag of w_i.

We simplify the joint optimization in a cascaded way with two different forms (5) and (6).

$$(D^*, T^*) = argmax_{(D,T)} \prod_{i=1}^{n} P(T_1^i|W_1^i, T_1^{i-1}) \times P(d_i|W_1^i, T_1^i) \quad (5)$$

$$(D^*, T^*) = argmax_{(D,T)} \prod_{i=1}^{n} P(d_i|W_1^i, T_1^{i-1}) \times P(T_1^i|W_1^i, T_1^{i-1}, d_i) \quad (6)$$

Here, $P(T_1^i|.)$ is the parsing model, and $P(d_i|.)$ is the disfluency model used to predict the disluency tags on condition of the contexts of partial trees that have been built.

In (5), the parsing model is calculated first, followed by the calculation of the disfluency model. Inspired by (Zhang et al., 2013), we associate the disfluency tags to the transition actions so that the calculation of $P(d_i|W_1^i, T_1^i)$ can be omitted as d_i can be inferred from the partial tree T_1^i. We then get

$$(D^*, T^*) = argmax_{(D,T)} \prod_{i=1}^{n} P(d_i, T_1^i|W_1^i, T_1^{i-1}) \quad (7)$$

Where the parsing and disfluency detection are unified into one model. We refer to this model as the Unified Transition(UT) model.

While in (6), the disfluency model is calculated first, followed by the calculation of the parsing model. We model $P(d_i|.)$ as a binary classifier to classify whether a word is disfluent or not. We refer to this model as the binary classifier transition (BCT) model.

3.2 Unified transition-based model (UT)

In model (7), in addition to the standard 4 transition actions mentioned in Section 2, the UT model

adds 2 new transition actions which extend the original *Shift* and *RightArc* transitions as shown below:

- *Dis_Shift*: Performs what *Shift* does then marks the pushed word as disfluent.
- *Dis_RightArc*: Adds a virtual link from the front of the queue to the top of the stack which is similar to *Right_Arc*, marking the front of the queue as disfluenct and pushing it to the stack.

Figure 3 shows an example of how the UT model works. Given an input "*he did great was great*", the optimal parsing tree is predicted by the UT model. According to the parsing tree, we can get the disfluency tags "N X X N N" which have been attached to each word. To ensure the normal words are built grammatical in the parse tree, we apply a constraint to the UT model.

UT model constraint: When a word is marked disfluent, all the words in its left and right subtrees will be marked disfluent and all the links of its descendent offsprings will be converted to virtual links, no matter what actions are applied to these words.

For example, the italic word "*great*" will be marked disfluent, no matter what actions are performed on it.

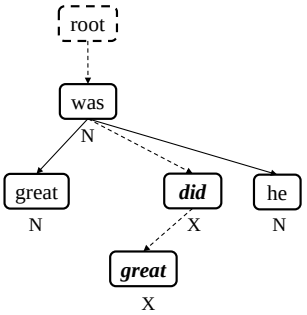

Figure 3: An example of UT model, where 'N' means the word is a fluent word and 'X' means it is disfluent. Words with italic font are Reparandums.

3.3 A binary classifier transition-based model (BCT)

In model (6), we perform the binary classifier and the parsing model together by augmenting the Shift-Reduce algorithm with a binary classifier transition(BCT) action:

- *BCT* : Classifies whether the current word is disfluent or not. If it is, remove it from the queue, push it into the stack which is similar to *Shift* and then mark it as disfluent, otherwise the original transition actions will be used.

It is noted that when BCT is performed, the next action must be *Reduce*. This constraint guarantees that any disfluent word will not have any descendent offspring. Figure 2(b) shows an example of the BCT model. When the partial tree "*great was*" is built, the next word "*did*" is obviously disfluent. Unlike UT model, the BCT will not link the word "*did*" to any word. Instead only a virtual link will add it to the virtual root.

3.4 Training and decoding

In practice, we use the same linear model for both models (6) and (7) to score a parsing tree as:

$$Score(T) = \sum_{action} \phi(action) \cdot \vec{\lambda}$$

Where $\phi(action)$ is the feature vector extracted from partial hypothesis T for a certain $action$ and $\vec{\lambda}$ is the weight vector. $\phi(action) \cdot \vec{\lambda}$ calculates the score of a certain transition action. The score of a parsing tree T is the sum of $action$ scores.

In addition to the basic features introduced in (Zhang and Nivre, 2011) that are defined over bag of words and POS-tags as well as tree-based context, our models also integrate three classes of new features combined with Brown cluster features (Brown et al., 1992) that relate to the right-to-left transition-based parsing procedure as detailed below.

Simple repetition function

- $\delta_I(a, b)$: A logic function which indicates whether a and b are identical.

Syntax-based repetition function

- $\delta_L(a, b)$: A logic function which indicates whether a is a left child of b.
- $\delta_R(a, b)$: A logic function which indicates whether a is a right child of b.

Longest subtree similarity function

- $N_I(a, b)$: The count of identical children on the left side of the root node between subtrees rooted at a and b.
- $N_\#(a_{0..n}, b)$: The count of words among a_0 .. a_n that are on the right of the subtree rooted at b.

Table 1 summarizes the features we use in the model computation, where w_s denotes the top word of the stack, w_0 denotes the front word of the queue and $w_{0..2}$ denotes the top three words of the queue. Every p_i corresponds to the POS-tag of w_i and $p_{0..2}$ represents the POS-tags of $w_{0..2}$. In addition, $w_i c$ means the Brown cluster of w_i. With these symbols, several new feature templates are defined in Table 1. Both our models have the same feature templates.

Basic features	All templates in (Zhang and Nivre, 2011)
New disfluency features	
Function unigrams	$\delta_I(w_s, w_0)$;$\delta_I(p_s, p_0)$; $\delta_L(w_0, w_s)$;$\delta_L(p_0, p_s)$; $\delta_R(w_0, w_s)$;$\delta_R(p_0, p_s)$; $N_I(w_0, w_s)$;$N_I(p_0, p_s)$; $N_\#(w_{0..2}, w_s)$;$N_\#(p_{0..2}, p_s)$;
Function bigrams	$\delta_I(w_s, w_0)\delta_I(p_s, p_0)$; $\delta_L(w_0, w_s)\delta_L(p_0, p_s)$; $\delta_R(w_0, w_s)\delta_R(p_0, p_s)$; $N_I(w_0, w_s)N_I(p_0, p_s)$; $N_\#(w_{0..2}, w_s)N_\#(p_{0..2}, p_s)$; $\delta_I(w_s, w_0)w_s c$; $\delta_I(w_s, w_0)w_0 c$;
Function trigrams	$w_s w_0 \delta_I(w_s, w_0)$; $w_s w_0 \delta_I(p_s, p_0)$;

Table 1: Feature templates designed for disfluency detection and dependency parsing.

Similar to the work in (Zhang and Clark, 2008; Zhang and Nivre, 2011), we train our models by averaged perceptron (Collins, 2002). In decoding, beam search is performed to get the optimal parsing tree as well as the tag sequence.

4 Experiments

4.1 Experimental setup

Our training data is the Switchboard portion of the English Penn Treebank (Marcus et al., 1993) corpus, which consists of telephone conversations about assigned topics. As not all the Switchboard data has syntactic bracketing, we only use the subcorpus of PAESED/MRG/SWBD. Following the experiment settings in (Charniak and Johnson, 2001), the training subcorpus contains directories 2 and 3 in PAESED/MRG/SWBD and directory 4 is split into test and development sets. We use the Stanford dependency converter (De Marneffe et al., 2006) to get the dependency structure from the Switchboard corpus, as Honnibal and Johnson (2014) prove that Stanford converter is robust to the Switchboard data.

For our Chinese experiments, no public Chinese corpus is available. We annotate about 25k spoken sentences with only disfluency annotations according to the guideline proposed by Meteer et al. (1995). In order to generate similar data format as English Switchboard corpus, we use Chinese dependency parsing trained on the Chinese Treebank corpus to parse the annotated data and use these parsed data for training and testing . For our Chinese experiment setting, we respectively select about 2k sentences for development and testing. The rest are used for training.

To train the UT model, we create data format adaptation by replacing the original *Shift* and *RightArc* of disfluent words with *Dis_Shift* and *Dis_RightArc*, since they are just extensions of *Shift* and *RightArc*. For the BCT model, disfluent words are directly depended to the root node and all their links and labels are removed. We then link all the fluent children of disfluent words to parents of disfluent words. We also remove partial words and punctuation from data to simulate speech recognizer results where such information is not available (Johnson and Charniak, 2004). Additionally, following Honnibal and Johnson (2014), we remove all one token sentences as these sentences are trivial for disfluency detection, then lowercase the text and discard filled pauses like "um" and "uh".

The evaluation metrics of disfluency detection are precision (Prec.), recall (Rec.) and f-score (F1). For parsing accuracy metrics, we use unlabeled attachment score (UAS) and labeled attachment score (LAS). For our primary comparison, we evaluate the widely used CRF labeling model, the state-of-the-art M^{3}N model presented by Qian and Liu (2013) which has been commonly used as baseline in previous works and the state-of-the-art L2R parsing based joint model proposed by Honnibal and Johnson (2014).

4.2 Experimental results

4.2.1 Performance of disfluency detection on English Swtichboard corpus

The evaluation results of both disfluency detection and parsing accuracy are presented in Table 2. The accuracy of M^{3}N directly refers to the re-

	Disfluency detection accuracy			Parsing accuracy	
Method	Prec.	Rec.	F1	UAS	LAS
CRF(BOW)	81.2%	44.9%	57.8%	88.7%	84.7%
CRF(BOW+POS)	88.3%	62.2%	73.1%	89.2%	85.6%
M^3N	N/A	N/A	84.1%	N/A	N/A
$M^3N^\dagger$	90.5%	79.1%	84.4%	91%	88.2%
H&J	N/A	N/A	84.1%	90.5%	N/A
UT(basic features)	86%	72.5%	78.7%	91.9%	89.0%
UT(+new features)	88.8%	75.1%	81.3%	92.1%	89.4%
BCT(basic features)	88.2%	77.9%	82.7%	92.1%	89.3%
BCT(+new features)	90.3%	80.5%	85.1%	92.2%	89.6%

Table 2: Disfluency detection and parsing accuracies on English Switchboard data. The accuracy of M^3N refers to the result reported in (Qian and Liu, 2013). H&J is the L2R parsing based joint model in (Honnibal and Johnson, 2014). The results of $M^3N^\dagger$ come from the experiments with toolkit released by Qian and Liu (2013) on our pre-processed corpus.

sults reported in (Qian and Liu, 2013). The results of $M^3N^\dagger$ come from our experiments with the toolkit[2] released by Qian and Liu (2013) which uses our data set with the same pre-processing. It is comparable between our models and the L2R parsing based joint model presented by Honnibal and Johnson (2014), as we all conduct experiments on the same pre-processed data set. In order to compare parsing accuracy, we use the CRF and $M^3N^\dagger$ model to pre-process the test set by removing all the detected disfluencies, then evaluate the parsing performance on the processed set. From the table, our BCT model with new disfluency features achieves the best performance on disfluency detection as well as dependency parsing.

The performance of the CRF model is low, because the local features are not powerful enough to capture long span disfluencies. Our main comparison is with the $M^3N^\dagger$ labeling model and the L2R parsing based model by Honnibal and Johnson (2014). As illustrated in Table 2, the BCT model outperforms the $M^3N^\dagger$ model (we got an accuracy of 84.4%, though 84.1% was reported in their paper) and the L2R parsing based model respectively by 0.7 point and 1 point on disfluency detection, which shows our method can efficiently tackle disfluencies. This is because our method can cater extremely well to the disfluency constraint and perform optimal search with identical transition counts over all hypotheses in beam search. Furthermore, our global syntactic and disfluency features can help capture long-range dependencies for disfluency detection. However, the UT model does not perform as well as BCT. This is because the UT model suffers from the risk that normal words may be linked to disfluencies which may bring error propagation in decoding. In addition our models with only basic features respectively score about 3 points below the models adding new features, which shows that these features are important for disfluency detection. In comparing parsing accuracy, our BCT model outperforms all the other models, showing that this model is more robust on disfluent parsing.

[2]The toolkit is available at https://code.google.com/p/disfluency-detection/downloads.

4.2.2 Performance of disfluency detection on different part-of-speeches

In this section, we further analyze the frequency of different part-of-speeches in disfluencies and test the performance on different part-of-speeches. Five classes of words take up more than 73% of all disfluencies as shown in Table 3, which are pronouns (contain PRP and PRP$), verbs (contain VB,VBD,VBP,VBZ,VBN), determiners (contain DT), prepositions (contain IN) and conjunctions (contain CC). Obviously, these classes of words appear frequently in our communication.

	Pron.	Verb	Dete.	Prep.	Conj.	Others
Dist.	30.2%	14.7%	13%	8.7%	6.7%	26.7%

Table 3: Distribution of different part-of-speeches in disfluencies. Conj.=conjunction; Dete.=determiner; Pron.=pronoun; Prep.= preposition.

Table 4 illustrates the performance (f-score) on these classes of words. The results of L2R parsing based joint model in (Honnibal and Johnson, 2014) are not listed because we cannot get such detailed data.

	CRF (BOW)	CRF (BOW +POS)	$M^3N^\dagger$	UT (+feat.)	BCT (+feat.)
Pron.	73.9%	85%	92%	91.5%	93.8%
Verb	38.2%	64.8%	84.2%	82.3%	84.7%
Dete.	66.8%	80%	88%	83.7%	87%
Prep.	60%	71.5%	79.1%	76.1%	79.3%
Conj.	75.2%	82.2%	81.6%	79.5%	83.2%
Others	43.2%	61%	78.4%	72.3%	79.1%

Table 4: Performance on different classes of words. Dete.=determiner; Pron.=pronoun; Conj.=conjunction; Prep.= preposition. feat.=new disfluency features

As shown in Table 4, our BCT model outperforms all other models except that the performance on determiner is lower than $M^3N^\dagger$, which shows that our algorithm can significantly tackle common disfluencies.

4.2.3 Performance of disfluency detection on Chinese annotated corpus

In addition to English experiments, we also apply our method on Chinese annotated data. As there is no standard Chinese corpus, no Chinese experimental results are reported in (Honnibal and Johnson, 2014; Qian and Liu, 2013). We only use the CRF-based labeling model with lexical and POS-tag features as baselines. Table 5 shows the results of Chinese disfluency detection.

Model	Prec.	Rec.	F1
CRF(BOW)	89.5%	35.6%	50.9%
CRF(BOW+POS)	83.4%	41.6%	55.5%
UT(+new features)	86.7%	59.5%	70.6%
BCT(+new features)	85.5%	61%	71.2%

Table 5: Disfluency detection performance on Chinese annotated data.

Our models outperform the CRF model with bag of words and POS-tag features by more than 15 points on f-score which shows that our method is more effective. As shown latter in 4.2.4, the standard transition-based parsing is not robust in parsing disfluent text. There are a lot of parsing errors in Chinese training data. Even though we are still able to get promising results with less data and un-golden parsing annotations. We believe that if we were to have the golden Chinese syntactic annotations and more data, we would get much better results.

4.2.4 Performance of transition-based parsing

In order to show whether the advantage of the BCT model is caused by the disfluency constraint or the difference between R2L and L2R parsing models, in this section, we make a comparison between the original left-to-right transition-based parsing and right-to-left parsing. These experiments are performed with the Penn Treebank (PTB) Wall Street Journal (WSJ) corpus. We follow the standard approach to split the corpus as 2-21 for training, 22 for development and section 23 for testing (McDonald et al., 2005). The features for the two parsers are basic features in Table 1. The POS-tagger model that we implement for a pre-process before parsing also uses structured perceptron for training and can achieve a competitive accuracy of 96.7%. The beam size for both POS-tagger and parsing is set to 5. Table 6 presents the results on WSJ test set and Switchboard (SWBD) test set.

Data sets	Model	UAS	LAS
WSJ	L2R Parsing	92.1%	89.8%
	R2L Parsing	92.0%	89.6%
SWBD	L2R Parsing	88.4%	84.4%
	R2L Parsing	88.7%	84.8%

Table 6: Performance of our parsers on different test sets.

The parsing accuracy on SWBD is lower than WSJ which means that the parsers are more robust on written text data. The performances of R2L and L2R parsing are comparable on both SWBD and WSJ test sets. This demonstrates that the effectiveness of our disfluency detection model mainly relies on catering to the disfluency constraint by using R2L parsing based approach, instead of the difference in parsing models between L2R and R2L parsings.

5 Related work

In practice, disfluency detection has been extensively studied in both speech processing field and natural language processing field. Noisy channel models have been widely used in the past to detect

disfluencies. Johnson and Charniak (2004) proposed a TAG-based noisy channel model where the TAG model was used to find rough copies. Thereafter, a language model and MaxEnt reranker were added to the noisy channel model by Johnson et al. (2004). Following their framework, Zwarts and Johnson (2011) extended this model using minimal expected f-loss oriented n-best reranking with additional corpus for language model training.

Recently, the max-margin markov networks (M^3N) based model has achieved great improvement in this task. Qian and Liu (2013) presented a multi-step learning method using weighted M^3N model for disfluency detection. They showed that M^3N model outperformed many other labeling models such as CRF model. Following this work, Wang et al. (2014) used a beam-search decoder to combine multiple models such as M^3N and language model, they achieved the highest f-score. However, direct comparison with their work is difficult as they utilized the whole SWBD data while we only use the subcorpus with syntactic annotation which is only half the SWBD corpus and they also used extra corpus for language model training.

Additionally, syntax-based approaches have been proposed which concern parsing and disfluency detection together. Lease and Johnson (2006) involved disfluency detection in a PCFG parser to parse the input along with detecting disfluencies. Miller and Schuler (2008) used a right corner transform of syntax trees to produce a syntactic tree with speech repairs. But their performance was not as good as labeling models. There exist two methods published recently which are similar to ours. Rasooli and Tetreault (2013) designed a joint model for both disfluency detection and dependency parsing. They regarded the two tasks as a two step classifications. Honnibal and Johnson (2014) presented a new joint model by extending the original transition actions with a new "Edit" transition. They achieved the state-of-the-art performance on both disfluency detection and parsing. But this model suffers from the problem that the number of transition actions is not identical for different hypotheses in decoding, leading to the failure of performing optimal search. In contrast, our novel right-to-left transition-based joint method caters to the disfluency constraint which can not only overcome the decoding deficiency in previous work but also achieve significantly higher performance on disfluency detection as well as dependency parsing.

6 Conclusion and Future Work

In this paper, we propose a novel approach for disfluency detection. Our models jointly perform parsing and disfluency detection from right to left by integrating a rich set of disfluency features which can yield parsing structure and difluency tags at the same time with linear complexity. The algorithm is easy to implement without complicated backtrack operations. Experiential results show that our approach outperforms the baselines on the English Switchboard corpus and experiments on the Chinese annotated corpus also show the language independent nature of our method. The state-of-the-art performance on disfluency detection and dependency parsing can benefit the downstream tasks of text processing.

In future work, we will try to add new classes of features to further improve performance by capturing the property of disfluencies. We would also like to make an end-to-end MT test over transcribed speech texts with disfluencies removed based on the method proposed in this paper.

Acknowledgments

We are grateful to the anonymous reviewers for their insightful comments. We also thank Mu Li, Shujie Liu, Lei Cui and Nan Yang for the helpful discussions.

References

Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. 1992. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479.

Eugene Charniak and Mark Johnson. 2001. Edit detection and parsing for transcribed speech. In *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*, pages 1–9. Association for Computational Linguistics.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8. Association for Computational Linguistics.

Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454.

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. Association for Computational Linguistics.

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

Mark Johnson and Eugene Charniak. 2004. A tag-based noisy channel model of speech repairs. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 33. Association for Computational Linguistics.

Mark Johnson, Eugene Charniak, and Matthew Lease. 2004. An improved model for recognizing disfluencies in conversational speech. In *Proceedings of Rich Transcription Workshop*.

Jeremy G Kahn, Matthew Lease, Eugene Charniak, Mark Johnson, and Mari Ostendorf. 2005. Effective use of prosody in parsing conversational speech. In *Proceedings of the conference on human language technology and empirical methods in natural language processing*, pages 233–240. Association for Computational Linguistics.

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. Association for Computational Linguistics.

Yang Liu, Elizabeth Shriberg, Andreas Stolcke, Dustin Hillard, Mari Ostendorf, and Mary Harper. 2006. Enriching speech recognition with automatic detection of sentence boundaries and disfluencies. *Audio, Speech, and Language Processing, IEEE Transactions on*, 14(5):1526–1540.

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.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 91–98. Association for Computational Linguistics.

Marie W Meteer, Ann A Taylor, Robert MacIntyre, and Rukmini Iyer. 1995. *Dysfluency annotation stylebook for the switchboard corpus*. University of Pennsylvania.

Tim Miller and William Schuler. 2008. A unified syntactic model for parsing fluent and disfluent speech. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, pages 105–108. Association for Computational Linguistics.

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

Mohammad Sadegh Rasooli and Joel R Tetreault. 2013. Joint parsing and disfluency detection in linear time. In *EMNLP*, pages 124–129.

Xuancong Wang, Hwee Tou Ng, and Khe Chai Sim. 2014. A beam-search decoder for disfluency detection. In *Proc. of COLING*.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 562–571. 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: short papers-Volume 2*, pages 188–193. 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 the 44th annual meeting of the Association for Computational Linguistics*, pages 561–568. Association for Computational Linguistics.

Dongdong Zhang, Shuangzhi Wu, Nan Yang, and Mu Li. 2013. Punctuation prediction with transition-based parsing. In *ACL (1)*, pages 752–760.

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-Volume 1*, pages 703–711. Association for Computational Linguistics.