Jointly or Separately: Which is Better for Parsing Heterogeneous Dependencies?

Meishan Zhang[†], **Wanxiang Che**[†], **Yanqiu Shao**[‡] , **Ting Liu**[†*]
[†]Research Center for Social Computing and Information Retrieval
Harbin Institute of Technology, China
`{mszhang, car, tliu}@ir.hit.edu.cn`
[‡]Beijing Language and Culture University
`yqshao163@163.com`

Abstract

For languages such as English, several constituent-to-dependency conversion schemes are proposed to construct corpora for dependency parsing. It is hard to determine which scheme is better because they reflect different views of dependency analysis. We usually obtain dependency parsers of different schemes by training with the specific corpus separately. It neglects the correlations between these schemes, which can potentially benefit the parsers. In this paper, we study how these correlations influence final dependency parsing performances, by proposing a joint model which can make full use of the correlations between heterogeneous dependencies, and finally we can answer the following question: parsing heterogeneous dependencies jointly or separately, which is better? We conduct experiments with two different schemes on the Penn Treebank and the Chinese Penn Treebank respectively, arriving at the same conclusion that jointly parsing heterogeneous dependencies can give improved performances for both schemes over the individual models.

1 Introduction

Dependency parsing has been intensively studied in recent years (McDonald et al., 2005; Nivre, 2008; Zhang and Clark, 2008; Huang et al., 2009; Koo and Collins, 2010; Zhang and Nivre, 2011; Sartorio et al., 2013; Choi and McCallum, 2013; Martins et al., 2013). Widely-used corpus for training a dependency parser is usually constructed according to a specific constituent-to-dependency conversion scheme. Several conversion schemes for certain languages have been available. For example, the English language has at least four schemes based on the Penn Treebank (PTB), including the `Yamada` scheme (Yamada and Matsumoto, 2003), the `CoNLL 2007` scheme (Nilsson et al., 2007), the `Stanford` scheme (de Marneffe and Manning, 2008) and the `LTH` scheme (Johansson and Nugues, 2007). There are different conversion schemes for the Chinese Penn Treebank (CTB) as well, including the `Zhang` scheme (Zhang and Clark, 2008) and the `Stanford` scheme (de Marneffe and Manning, 2008). It is hard to judge which scheme is more superior, because each scheme reflects a specific view of dependency analysis, and also there is another fact that different natural language processing (NLP) applications can prefer different conversion schemes (Elming et al., 2013).

Traditionally, we get dependency parsers of different schemes by training with the specific corpus separately. The method neglects the correlations between these schemes, which can potentially help different dependency parsers. On the one hand, there are many consistent dependencies across heterogeneous dependency trees. Some dependency structures remain constant in different conversion schemes. Taking the `Yamada` and the `Stanford` schemes as an example, overall 70.27% of the dependencies are identical (ignoring the dependency labels), according to our experimental analysis. We show a concrete example for the two heterogeneous dependency trees in Figure 1, where six of the twelve dependencies are consistent in the two dependency trees (shown by the solid arcs).

On the other hand, differences between heterogeneous dependencies can possibly boost the evidences of the consistent dependencies. For example in Figure 1, the dependencies “do$\overset{\text{VC}}{\curvearrowleft}$think”

[*]Corresponding author.

This work is licenced under a Creative Commons Attribution 4.0 International License. Page numbers and proceedings footer are added by the organizers. License details: `http://creativecommons.org/licenses/by/4.0/`

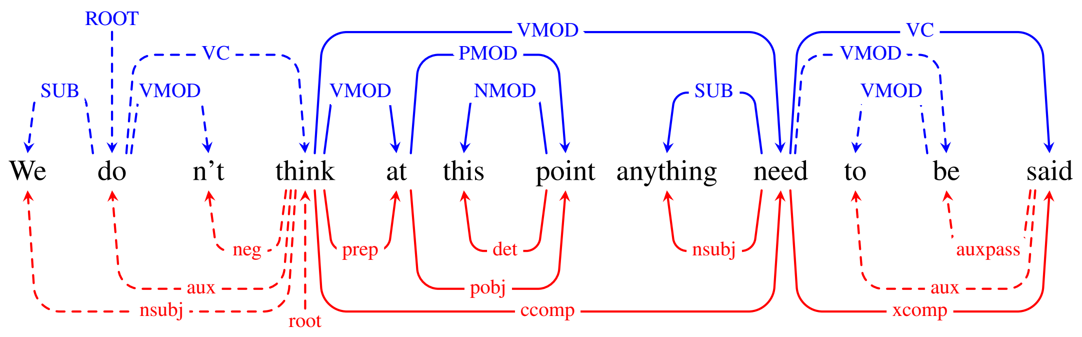


Figure 1: An example to show the differences and similarities of two dependency schemes. The above dependency tree is based on the `Yamada` scheme, while the below dependency tree is based on the `Stanford` scheme. The solid arcs show the consistent dependencies between the two dependency trees, while the dashed arcs show the differences between the two trees.

and "We$\overset{\text{nsubj}}{\curvearrowleft}$think" from the two trees can both be potential evidences to support the dependency "think$\curvearrowright$at". Another example, the label "PMOD" from the `Yamada` scheme and the label "pobj" from the `Stanford` scheme on a same dependency "at$\curvearrowright$point" can make it more reliable than one alone.

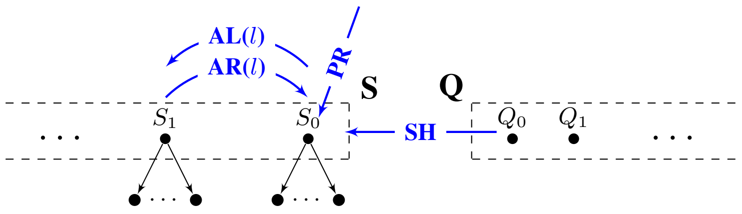
In this paper, we investigate the influences of the correlations between different dependency schemes on parsing performances. We propose a joint model to parse heterogeneous dependencies from two schemes simultaneously, so that the correlations can be fully used by their interactions in a single model. Joint models have been widely studied to enhance multiple tasks in NLP community, including joint word segmentation and POS-tagging (Jiang et al., 2008; Kruengkrai et al., 2009; Zhang and Clark, 2010), joint POS-tagging and dependency parsing (Li et al., 2011; Hatori et al., 2011), and the joint word segmentation, POS-tagging and dependency parsing (Hatori et al., 2012). These models are proposed over pipelined tasks. We apply the joint model into parallel tasks, and parse heterogeneous dependencies together. To our knowledge, we are the first work to investigate joint models on parallel tasks.

We exploit a transition-based framework with global learning and beam-search decoding to implement the joint model (Zhang and Clark, 2011). The joint model is extended from a state-of-the-art transition-based dependency parsing model. We conduct experiments on PTB with the `Yamada` and the `Stanford` schemes, and also on CTB 5.1 with the `Zhang` and the `Stanford` schemes. The results show that our joint model gives improved performances over the individual baseline models for both schemes on both English and Chinese languages, demonstrating positive effects of the correlations between the two schemes. We make the source code freely available at `http://sourceforge.net/projects/zpar/,version0.7`.

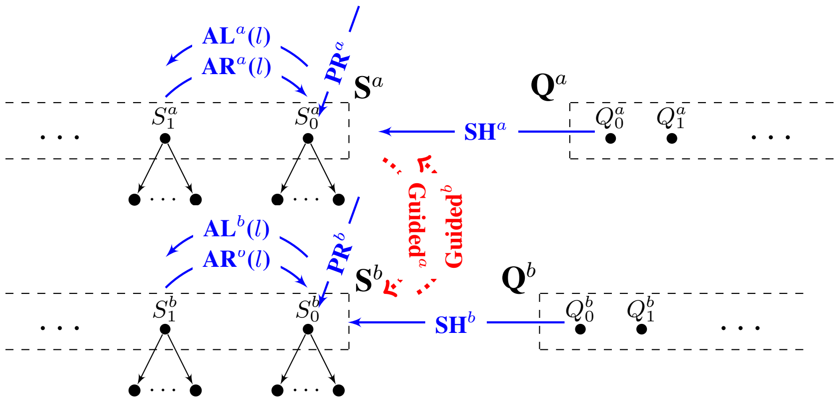
2 Baseline

Traditionally, the dependency parsers of different schemes are trained with their corpus separately, using a state-of-the-art dependency parsing algorithm (Zhang and Clark, 2008; Huang et al., 2009; Koo and Collins, 2010; Zhang and McDonald, 2012; Choi and McCallum, 2013). In this work, we exploit a transition-based arc-standard dependency parsing model combined with global learning and beam-search decoding as the baseline. which is initially proposed by Huang et al. (2009). In the following, we give a detailed description of the model.

In a typical transition-based system for dependency parsing, we define a transition state, which consists of a stack to save partial-parsed trees and a queue to save unprocessed words. The parsing is performed incrementally via a set of transition actions. The transition actions are used to change contents of the stack and the queue in a transition state. Initially, a start state has an empty stack and all words of a sentence in its queue. Then transition actions are applied to the start state, and change states step by step. Finally, we arrive at an end state with only one parsed tree on the stack and no words in the queue. We score each state by its features generated from the historical actions.



(a) Arc-standard dependency parsing model for a single dependency tree



(b) The joint model based on arc-standard dependency parsing for two dependency trees

Figure 2: Illustrations for the baseline dependency parsing model and our proposed joint model.

In the baseline arc-standard transition system, we define four kinds of actions, as shown in Figure 2(a). They are *shift* (SH), *arc-left with dependency label* l (AL(l)), *arc-right with dependency label* l (AR(l)) and *pop-root* (PR), respectively. The *shift* action shifts the first element Q_0 of the queue onto the stack; the action *arc-left with dependency label* l builds a left arc between the top element S_0 and the second top element S_1 on the stack, with the dependency label being specified by l; the action *arc-right with dependency label* l builds a right arc between the top element S_0 and the second top element S_1 on the stack, with the dependency label being specified by l; and the *pop-root* action defines the root node of a dependency tree when there is only one element on the stack and no element in the queue.

During decoding, each state may have several actions. We employ a fixed beam to reduce the search space. The low-score states are pruned from the beam when it is full. The feature templates in our baseline are shown by Table 1, referring to *baseline feature templates*. We learn the feature weights by the averaged percepron algorithm with early-update (Collins and Roark, 2004; Zhang and Clark, 2011).

3 The Proposed Joint Model

The aforementioned baseline model can only handle a single dependency tree. In order to parse multiple dependency trees for a sentence, we usually use individual dependency parsers. This method is not able to exploit the correlations across different dependency schemes. The joint model to parse multiple dependency trees with a single model is an elegant way to exploit these correlations fully. Inspired by this, we make a novel extension to the baseline arc-standard transition system, arriving at a joint model to parse two heterogeneous dependency trees for a sentence simultaneously.

In the new transition system, we double the original transition state of one stack and one queue into two stacks and two queues, as shown by Figure 2(b). We use stacks S^a and S^b and queues Q^a and Q^b to save partial-parsed dependency trees and unprocessed words for two schemes a and b, respectively. Similarly, the transition actions are doubled as well. We have eight transition actions, where four of them are aimed for scheme a, and the other four are aimed for scheme b. The concrete action definitions are similar to the original actions, except an additional constraint that actions should be operated over the corresponding stack and queue of scheme a or b.

We assume that the actions to build a specific tree of scheme a are $A_1^a A_2^a \cdots A_n^a$, and the actions to

Baseline feature templates
Unigram features
S_0w S_0t S_0wt S_1w S_1t S_1wt N_0w N_0t N_0wt N_1w N_1t N_1wt
Bigram features
$S_0w{\cdot}S_1w$ $S_0w{\cdot}S_1t$ $S_0t{\cdot}S_1w$ $S_0t{\cdot}S_1t$ $S_0w{\cdot}N_0w$ $S_0w{\cdot}N_0t$ $S_0t{\cdot}N_0w$ $S_0t{\cdot}N_0t$
Second-order features
$S_{0l}w$ $S_{0r}w$ $S_{0l}t$ $S_{0r}t$ $S_{0l}l$ $S_{0r}l$ $S_{1l}w$ $S_{1r}w$ $S_{1l}t$ $S_{1r}t$ $S_{1l}l$ $S_{1r}l$
$S_{0l2}w$ $S_{0r2}w$ $S_{0l2}t$ $S_{0r2}t$ $S_{0l2}l2$ $S_{0r2}l2$ $S_{1l2}w$ $S_{1r2}w$ $S_{1l2}t$ $S_{1r2}t$ $S_{1l2}l2$ $S_{1r2}l2$
Third-order features
$S_0t{\cdot}S_{0l}t{\cdot}S_{0l2}t$ $S_0t{\cdot}S_{0r}t{\cdot}S_{0r2}t$ $S_1t{\cdot}S_{1l}t{\cdot}S_{1l2}t$ $S_1t{\cdot}S_{1r}t{\cdot}S_{1r2}t$
$S_0t{\cdot}S_1t{\cdot}S_{0l}t$ $S_0t{\cdot}S_1t{\cdot}S_{0l2}t$ $S_0t{\cdot}S_1t{\cdot}S_{0r}t$ $S_0t{\cdot}S_1t{\cdot}S_{0r2}t$
$S_0t{\cdot}S_1t{\cdot}S_{1l}t$ $S_0t{\cdot}S_1t{\cdot}S_{1l2}t$ $S_0t{\cdot}S_1t{\cdot}S_{1r}t$ $S_0t{\cdot}S_1t{\cdot}S_{1r2}t$
Valancy features
S_0wv_l S_0tv_l S_0wv_r S_0tv_r S_1wv_l S_1tv_l S_1wv_r S_1tv_r
Label set features
S_0ws_r S_0ts_r S_0ws_l S_0ts_l S_1ws_l S_1ts_l
Proposed new feature templates for the joint model
Guided head features
$S_0w{\cdot}h_{guide}$ $S_0t{\cdot}h_{guide}$ $S_0wt{\cdot}h_{guide}$ $S_1w{\cdot}h_{guide}$ $S_1t{\cdot}h_{guide}$ h_{guide}
Guided label features
$S_0w{\cdot}S_0l_{guide}$ $S_0t{\cdot}S_0l_{guide}$ $S_0wt{\cdot}S_0l_{guide}$ $S_1w{\cdot}S_0l_{guide}$ $S_1t{\cdot}S_0l_{guide}$ S_0l_{guide}
$S_0w{\cdot}S_1l_{guide}$ $S_0t{\cdot}S_1l_{guide}$ $S_0wt{\cdot}S_1l_{guide}$ $S_1w{\cdot}S_1l_{guide}$ $S_1t{\cdot}S_1l_{guide}$ S_1l_{guide}

Table 1: Feature templates for the baseline and joint models, where w denotes the word; t denotes the POS tag; v_l and v_r denote the left and right valencies; l denotes the dependency label; s_l and s_r denotes the label sets of the left and right children; the subscripts l and r denote the left-most and the right-most children, respectively; the subscripts $l2$ and $r2$ denote the second left-most and the second right-most children, respectively; h_{guide} denotes the head direction of the top two elements on the processing stack in the other tree; l_{guide} denotes the label of the same word in the other tree.

build a specific tree of scheme b for the same sentence are $A_1^bA_2^b\cdots A_n^b$. We use $\mathrm{ST}_0^a\mathrm{ST}_1^a\cdots\mathrm{ST}_n^a$ and $\mathrm{ST}_0^b\mathrm{ST}_1^b\cdots\mathrm{ST}_n^b$ to denote the historical states for the two action sequences, respectively. A sequence of actions should consist of $A_1^aA_2^a\cdots A_n^a$ and $A_1^bA_2^b\cdots A_n^b$ in a joint model. However, one question that needs to be answered is that, for a joint state $(\mathrm{ST}_i^a, \mathrm{ST}_j^b)$, which action should be chosen as the next step to merge the two action sequences into one sequence, A_{i+1}^a or A_{j+1}^b? To resolve the problem, we employ a parameter t to limit the next action in the joint model. When t is above zero, an action for scheme b can be applied only if the last action of scheme a is t steps in advance. For example, the action sequence is $A_1^aA_1^bA_2^aA_2^b\cdots A_n^aA_n^b$ when $t = 1$. t can be negative as well, denoting the reverse constraints.

In the joint model, we extract features separately for the two dependency schemes. When the next action is aimed for scheme a, we will extract features from S^a and Q^a, according to *baseline feature templates* in Table 1. In order to make use of the correlations between the two dependency parsing trees, we introduce several new feature templates, shown in Table 1 referring to *proposed new feature templates for the joint model.* The new features are based on two kinds of atomic features: the guided head h_{guide} and the guided dependency label l_{guide}. Assuming that the currently processing scheme is a, when the top two elements (S_0^a and S_1^a) have both found their heads in Guided^b (the partial-parsed trees of scheme b), we can fire the atomic feature h_{guide}, which denotes the arc direction between S_0 and S_1 in Guide^b ($S_0 \curvearrowleft S_1$, $S_0 \curvearrowright S_1$ or other). When S_0^a or S_1^a has its dependency label in Guided^b, we can fire the atomic feature l_{guide}, which denotes the dependency label of S_0^a or S_1^a in Guided^b. Similarly we can extract the h_{guide} and l_{guide} from Guide^a when we are processing scheme b. When t is infinite, we always have the two atomic features, because the other tree is already parsed. Thus the proposed new features can be the most effective when $t = \infty$ and $t = -\infty$. In other conditions, the other tree may not be ready for the new feature extracting. Similar to the baseline model, we use the beam-search decoding strategy to reduce the search space, and use the averaged perceptron with early-update to learn the feature weights.

We are especially interested in two cases of the joint models when t is infinite ($t = \infty$ and $t = -\infty$), where the tree of one specified scheme is always processed after the other tree is finished, because the new features can be most effectively exploited according to the above analysis. We assume that the first and second processing schemes are s_1 and s_2 respectively, to facilitate the below descriptions. We can see that the joint model behaves similarly to a pipeline reranking model, in optimizing scheme s_1's parsing performances. First we get K-best (K equals the beam size of the joint model) candidates for scheme s_1, and then employ additional evidences from scheme s_2's result, to rerank the K-best candidates, obtaining a better result. The joint model also behaves similarly to a pipeline feature-based stacking model (Li et al., 2012), in optimizing scheme s_2's parsing performances. After acquiring the best result of scheme s_1, we can use it to generate guided features to parse dependencies of scheme s_2. Thus additional information from scheme s_1 can be imported into the parsing model of scheme s_2. Different with the pipeline reranking and the feature-based stacking models, we employ a single model to achieve the two goals, making the interactions between the two schemes be better performed.

4 Experiments

4.1 Experimental Settings

In order to evaluate the baseline and joint models, we conduct experiments on English and Chinese data. For English, we obtain heterogeneous dependencies by the `Yamada` and the `Stanford` schemes, respectively. We transform the bracket constituent trees of English sentences into the `Yamada` dependencies with the Penn2Malt tool,[1] and into the `Stanford` dependencies with the Stanford parser version 3.3.1.[2] Following the standard splitting of PTB, we use sections 2-21 as the training data set, section 22 as the development data set, and section 23 as the final test data set. For Chinese, we obtain heterogeneous dependencies by the `Zhang` and the `Stanford` schemes, respectively. The `Zhang` dependencies are obtained by the Penn2Malt tool using the head rules from Zhang and Clark (2008), while the `Stanford` dependencies are obtained by the Stanford parser version 3.3.1 similar to English.

We use predicted POS tags in all the experiments. We utilize a linear-CRF POS tagger to obtain automatic POS tags for English and Chinese datasets.[3] We use a beam size of 64 to train dependency parsing models. We train the joint models with the `Yamada` or `Zhang` dependencies being handled on stack S^a and queue Q^a, and the `Stanford` dependencies being handled on stack S^b and queue Q^b, referring to Section 3. We follow the standard measures of dependency parsing to evaluate the baseline and joint models, including unlabeled attachment score (UAS), labeled attachment score (LAS) and complete match (CM). We ignore the punctuation words for all these measures.

4.2 Development Results

4.2.1 Baseline

Table 2 at the subtable "Baseline" shows the baseline results on the development data set. The performances of the `Yamada` scheme are better than those of the `Stanford` scheme. The UAS and LAS of the `Yamada` scheme are 92.83 and 91.73 respectively, while they are 92.85 and 90.49 for the `Stanford` scheme respectively. The results demonstrate that parsing the `Stanford` dependencies is more difficult than parsing the `Yamada` dependencies because of the lower performances of the `Stanford` scheme.

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

[2] The tool is available on `http://nlp.stanford.edu/software/lex-parser.shtml`. We use three options to perform the conversion: "-basic" and "-keepPunct", respectively.

[3] The tagging accuracies are 97.30% on the English test dataset and 93.68% on the Chinese test dataset. We thank Hao Zhang for sharing the data used in Martins et al. (2013) and Zhang et al. (2013a).

Model	`Yamada`			`Stanford`		
	UAS	LAS	CM	UAS	LAS	CM
Baseline	92.83	91.73	47.35	92.85	90.49	50.06
The joint models, where the `Yamada` dependencies are processed with priority						
$t = 1$	92.65	91.55	46.35	93.11	90.75	50.24
$t = 2$	92.65	91.57	46.71	93.15	90.77	50.59
$t = 3$	92.82	91.74	47.12	93.19	90.82	50.76
$t = 4$	92.89	91.78	47.35	93.27	90.93	51.29
$t = \infty$	**93.04**	**92.01**	**48.65**	**93.52**	**91.15**	**52.59**
The joint models, where the `Stanford` dependencies are processed with priority						
$t = -1$	92.62	91.54	46.71	93.10	90.70	50.76
$t = -2$	92.50	91.41	46.18	93.06	90.74	51.12
$t = -3$	92.57	91.42	47.00	93.10	90.68	51.35
$t = -4$	92.74	91.60	47.41	93.15	90.72	51.29
$t = -\infty$	**93.04**	91.95	47.88	93.19	90.91	50.71

Table 2: The main results on the development data set of the baseline and proposed joint models.

4.2.2 Parameter Tuning

The proposed joint model has one parameter t to adjust. The parameter t is used to control the decoding in a joint model, determining which kind of dependencies should be processed at the next step. In our joint model, if t is larger than zero, scheme a (the `Yamada` scheme) should be handled t steps in advance, while when t is smaller than zero, scheme b (the `Stanford` scheme) should be handled in advance. When the value of t is infinite, the dependency tree of one scheme is handled until the dependency tree of the other scheme is finished for a sentence.

As shown by Table 2, we have two major findings. First, the joint models are slightly better when t is above zero, by decoding with the `Yamada` scheme in advance. The phenomenon demonstrates that the decoding sequence is important in the joint parsing models. Second, no matter when t is above or below zero, the performances arrive at the peak when t is infinite. One benefit of the joint models is that we can use the correlations between different dependency trees, through the new features proposed by us. The new features can be the most effective when t is infinite according to the analysis Section 3. Thus this finding indicates that the new features are crucial in the joint models, since the ineffective utilization would decrease the model performances a lot. Actually, when the absolute value of t is small, the features can sometimes be fired and in some other times are not able to be fired, making the training insufficient and also inconsistent for certain word-pair dependencies when their distances can differ (when $t = 1$ for example, the joint model can fire the new features only if the dependency distance equals 1). This would make the final model deficient, and can even hurt performances of the `Yamada` scheme.

According to the results on the development data set, we use the $t = \infty$ for the final joint model, which first finishes the `Yamada` tree and then the `Stanford` tree for each sentence. Our final model achieves increases of 0.21 on UAS and 0.28 on LAS for the `Yamada` scheme, and increases 0.67 on UAS and 0.66 on LAS for the `Stanford` scheme.

4.2.3 Feature Ablation

In order to test the effectiveness of the proposed new features, we conduct a feature ablation experiment. Table 3 shows the results, where the mark "/wo" denotes the model without the new features proposed by us. For the `Yamada` scheme, losses of 0.15 on UAS and 0.21 on LAS are shown without the new features. While for `Stanford` scheme, larger decreases are shown by 0.57 on UAS and 0.58 on LAS, respectively. The results demonstrate the new features are effective in the joint model.

Model	Yamada			Stanford		
	UAS	LAS	CM	UAS	LAS	CM
Our joint model	93.04	92.01	48.65	93.52	91.15	52.59
Our joint model/wo	92.89	91.80	48.25	92.95	90.57	50.62
Δ	-0.15	-0.21	-0.40	-0.57	-0.58	-1.97

Table 3: Feature ablation results.

Model	Yamada			Stanford		
	UAS	LAS	CM	UAS	LAS	CM
Baseline	92.71	91.67	47.48	92.72	90.61	47.76
Our joint model	**92.89**	**91.86**	**48.39**	**93.30**‡	**91.19**‡	**50.37**
Zhang and Nivre (2011)	92.9	91.8	48.0	–	–	–
Rush and Petrov (2012)	–	–	–	92.7*	–	–
Martins et al. (2013)	93.07	–	–	92.82*	–	–
Zhang et al. (2013a)	93.50	92.41	–	93.64*	91.28*	–
Zhang and McDonald (2014)	93.57	92.48	–	93.71*/93.01**	91.37*/90.64**	–
Kong and Smith (2014)	–	–	–	92.20**	89.67**	–

Table 4: The final results on the test data set, where the results with mark ‡ demonstrates that the p-value is below 10^{-3} using t-test. Our Stanford dependencies are slightly different with previous works, where the results with mark * show the numbers for the Stanford dependencies from Stanford parser version 2.0.5 and the results with mark ** show the numbers for the Stanford dependencies from Stanford parser version 3.3.0.

4.3 Final Results

Table 4 shows our final results on the English test dataset. The final joint model achieves better performances than the baseline models for both the Yamada and the Stanford schemes, by increases of 0.18 on UAS and 0.19 on LAS for the Yamada scheme, and increases of 0.58 on UAS and 0.58 on LAS for the Stanford scheme. The results demonstrate that the interactions between the two dependency schemes are useful, and the joint model is superior to separately trained models in handling heterogeneous dependencies.

We compare our results with some representative previous work of dependency parsing as well. Zhang and Nivre (2011) is a feature-rich transition-based dependency parser using the arc-eager transition system. Rush and Petrov (2012), Zhang et al. (2013a) and Zhang and McDonald (2014) are state-of-the-art graph-based dependency parsers. Martins et al. (2013) and Kong and Smith (2014) report their results with the full TurboParser. TurboParser is also a graph-based dependency parser but its decoding algorithm has major differences with the general MST-style decoding.

4.4 Analysis

To better understand the joint model, we conduct analysis work on the Chinese development dataset. First, we make a comparison to see whether the consistent dependencies give larger increases by the joint model. As mentioned before, the consistent dependencies can be supported by different evidences from heterogeneous dependencies. We compute the proportion of the consistent dependencies (ignoring the dependency labels) between the Yamada and the Stanford dependencies, finding that 70.27% of the overall dependencies are consistent. Table 5 shows the comparison results. The joint model shows improvements for the consistent dependencies. However, it does not always show positive effectiveness for the inconsistent dependencies. The results support our initial motivation that consistent dependencies can benefit much in joint models .

We also make a comparison between the baseline and joint models with respect to dependency distance. We use the F-measure value to evaluate the performances. The dependency distances are normal-

	Yamada				Stanford			
	Consistent		Inconsistent		Consistent		Inconsistent	
	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
Baseline	93.43	92.39	91.44	90.17	93.74	91.35	90.75	88.47
Our joint model	93.81	92.85	91.21	90.02	94.58	92.15	91.01	88.78
Δ	**+0.38**	**+0.46**	-0.23	-0.15	**+0.84**	**+0.80**	+0.36	+0.31

Table 5: Performances of the baseline and joint models by whether the dependencies are consistent across the Yamada and the Stanford schemes, where the bold numbers denote the larger increases by comparisons of consistent and inconsistent dependencies for each scheme.

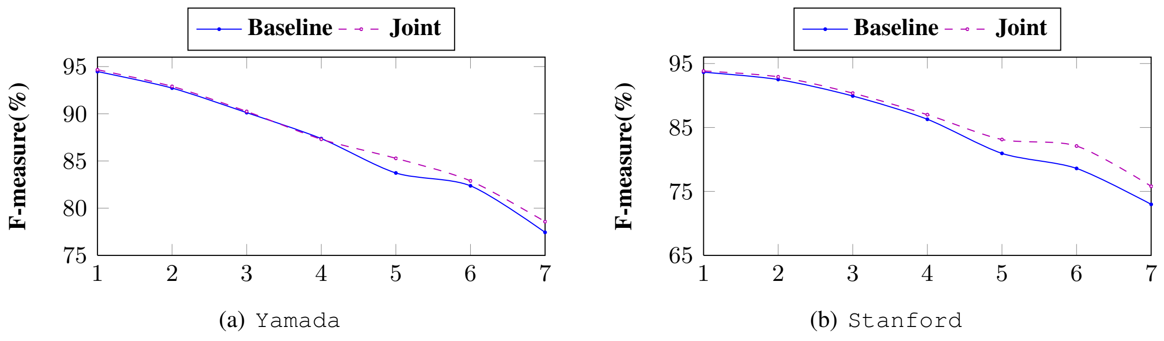


Figure 3: F-measures of the two heterogeneous dependencies with respect to dependency distance.

ized to a max value of 7. Figure 3 shows the comparison results. We find that the joint model can achieve consistent better performances for the dependencies of different dependency distance, demonstrating the robustness of the joint model in improving parsing performances. The joint model performs slightly better for long-distance dependencies, which is more obvious for the Stanford scheme.

4.5 Parsing Heterogeneous Chinese Dependencies

Table 6 shows our final results on the Chinese test data set. For Chinese, the joint model achieves better performances with Stanford dependencies being parsed first. The final joint model achieves better performances than the baseline models for both the Zhang and the Stanford schemes, by increases of 1.13 on UAS and 0.99 on LAS for the Zhang scheme, and increases of 0.30 on UAS and 0.36 on LAS for the Stanford scheme. The results also demonstrate similar conclusions with the experiments on English dataset.

5 Related Work

Our work is mainly inspired by the work of joint models. There are a number of successful studies on joint modeling pipelined tasks where one task is a prerequisite step of another task, for example, the joint model of word segmentation and POS-tagging (Jiang et al., 2008; Kruengkrai et al., 2009; Zhang and Clark, 2010), the joint model of POS-tagging and parsing (Li et al., 2011; Hatori et al., 2011; Bohnet and Nivre, 2012), the joint model of word segmentation, POS-tagging and parsing (Hatori et

Model	Zhang			Stanford		
	UAS	LAS	CM	UAS	LAS	CM
Baseline	79.07	76.08	27.96	80.33	75.29	31.14
Our joint model	**80.20**‡	**77.07**‡	**30.10**	**80.63**	**75.65**	**31.20**

Table 6: The final results on the test data set, where the results with mark ‡ demonstrates that the p-value is below 10^{-3} using t-test.

al., 2012; Zhang et al., 2013b; Zhang et al., 2014), and the joint model of morphological and syntactic analysis tasks (Bohnet et al., 2013). In our work, we propose a joint model on parallel tasks, to parse two heterogeneous dependency trees simultaneously.

There has been a line of work on exploiting multiple treebanks with heterogeneous dependencies to enhance dependency parsing. Li et al. (2012) proposed a feature-based stacking model to enhance a specific target dependency parser with the help of another treebank. Zhou and Zhao (2013) presented a joint inference framework to combine the parsing results based on two different treebanks. All these work are case studies of annotation adaptation from different sources, which have been done for Chinese word segmentation and POS-tagging as well (Jiang et al., 2009; Sun and Wan, 2012). In contrast to their work, we study the heterogeneous annotations derived from the same source. We use a unified model to parsing heterogeneous dependencies together.

Our joint parsing model exploits a transition-based framework with global learning and beam-search decoding (Zhang and Clark, 2011), extended from a arc-standard transition-based parsing model (Huang et al., 2009). The transition-based framework is easily adapted to a number of joint models, including joint word segmentation and POS-tagging (Zhang and Clark, 2010), the joint POS-tagging and parsing (Hatori et al., 2012; Bohnet and Nivre, 2012), and also joint word segmentation, POS-tagging and parsing (Hatori et al., 2012; Zhang et al., 2013b; Zhang et al., 2014).

6 Conclusions

We studied the effectiveness of the correlations between different constituent-to-dependency schemes for dependency parsing, by exploiting these information with a joint model to parse two heterogeneous dependency trees simultaneously. We make a novel extension to a transition-based arc-standard dependency parsing algorithm for the joint model. We evaluate our baseline and joint models on both English and Chinese datasets, based on the `Yamada/Zhang` and the `Stanford` dependency schemes. Final results demonstrate that the joint model which handles two heterogeneous dependencies can give improved performances for dependencies of both schemes. The source code for the joint model is publicly available at `http://sourceforge.net/projects/zpar/,version0.7`.

Acknowledgments

We thank Yue Zhang and the anonymous reviewers for their constructive comments, and gratefully acknowledge the support of the National Basic Research Program (973 Program) of China via Grant 2014CB340503, the National Natural Science Foundation of China (NSFC) via Grant 61133012, 61170144 and 61370164.

References

Bernd Bohnet and Joakim Nivre. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the EMNLP-CONLL*, pages 1455–1465, Jeju Island, Korea, July.

Bernd Bohnet, Joakim Nivre, Igor Boguslavsky, Richárd Farkas Filip Ginter, and Jan Hajic. 2013. Joint morphological and syntactic analysis for richly inflected languages. *TACL*, 1.

Jinho D. Choi and Andrew McCallum. 2013. Transition-based dependency parsing with selectional branching. In *Proceedings of ACL*, pages 1052–1062, August.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the ACL*, pages 111–118, Barcelona, Spain, July.

Marie-Catherine de Marneffe and Christopher D. Manning. 2008. The Stanford typed dependencies representation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pages 1–8, Manchester, UK, August.

Jakob Elming, Anders Johannsen, Sigrid Klerke, Emanuele Lapponi, Hector Martinez Alonso, and Anders Søgaard. 2013. Down-stream effects of tree-to-dependency conversions. In *Proceedings of the NAACL*, pages 617–626, Atlanta, Georgia, June.

Jun Hatori, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2011. Incremental joint POS tagging and dependency parsing in Chinese. In *Proceedings of 5th IJCNLP*, pages 1216–1224, Chiang Mai, Thailand, November.

Jun Hatori, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2012. Incremental joint approach to word segmentation, POS tagging, and dependency parsing in Chinese. In *Proceedings of the 50th ACL*, pages 1045–1053, Jeju Island, Korea, July.

Liang Huang, Wenbin Jiang, and Qun Liu. 2009. Bilingually-constrained (monolingual) shift-reduce parsing. In *Proceedings of the EMNLP*, pages 1222–1231.

Wenbin Jiang, Liang Huang, Qun Liu, and Yajuan Lü. 2008. A cascaded linear model for joint Chinese word segmentation and part-of-speech tagging. In *Proceedings of ACL-08*, pages 897–904, Columbus, Ohio, June.

Wenbin Jiang, Liang Huang, and Qun Liu. 2009. Automatic adaptation of annotation standards: Chinese word segmentation and POS tagging: a case study. In *Proceedings of the ACL-IJCNLP*, pages 522–530.

Richard Johansson and Pierre Nugues. 2007. Extended constituent-to-dependency conversion for english. In *Proceedings of NODALIDA 2007*, Tartu, Estonia.

Lingpeng Kong and Noah A Smith. 2014. An empirical comparison of parsing methods for stanford dependencies. *arXiv preprint arXiv:1404.4314*.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the ACL*, pages 1–11.

Canasai Kruengkrai, Kiyotaka Uchimoto, Jun'ichi Kazama, Yiou Wang, Kentaro Torisawa, and Hitoshi Isahara. 2009. An error-driven word-character hybrid model for joint Chinese word segmentation and POS tagging. In *Proceedings of the ACL-IJCNLP*, pages 513–521, Suntec, Singapore, August.

Zhenghua Li, Min Zhang, Wanxiang Che, Ting Liu, Wenliang Chen, and Haizhou Li. 2011. Joint models for Chinese POS tagging and dependency parsing. In *Proceedings of the EMNLP*, pages 1180–1191, Edinburgh, Scotland, UK., July.

Zhenghua Li, Ting Liu, and Wanxiang Che. 2012. Exploiting multiple treebanks for parsing with quasi-synchronous grammars. In *Proceedings of the 50th ACL*, pages 675–684, Jeju Island, Korea, July.

Andre Martins, Miguel Almeida, and Noah A. Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the 51st ACL*, pages 617–622, Sofia, Bulgaria, August. Association for Computational Linguistics.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of ACL*, number June, pages 91–98, Morristown, NJ, USA.

Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL*, pages 915–932.

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

Alexander M Rush and Slav Petrov. 2012. Vine pruning for efficient multi-pass dependency parsing. In *Proceedings of the NAACL*, pages 498–507.

Francesco Sartorio, Giorgio Satta, and Joakim Nivre. 2013. A transition-based dependency parser using a dynamic parsing strategy. In *Proceedings of the 51st ACL*, pages 135–144, Sofia, Bulgaria, August.

Weiwei Sun and Xiaojun Wan. 2012. Reducing approximation and estimation errors for Chinese lexical processing with heterogeneous annotations. In *Proceedings of the 50th ACL*, pages 232–241, Jeju Island, Korea, July.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*, volume 3.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of EMNLP*, pages 562–571, Honolulu, Hawaii, October.

Yue Zhang and Stephen Clark. 2010. A fast decoder for joint word segmentation and POS-tagging using a single discriminative model. In *Proceedings of the EMNLP*, pages 843–852, Cambridge, MA, October.

Yue Zhang and Stephen Clark. 2011. Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics*, 37(1):105–151.

Hao Zhang and Ryan McDonald. 2012. Generalized higher-order dependency parsing with cube pruning. In *Proceedings of the EMNLP*, pages 320–331.

Hao Zhang and Ryan McDonald. 2014. Enforcing structural diversity in cube-pruned dependency parsing. In *Proceedings of ACL*. Association for Computational Linguistics.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th ACL*, pages 188–193, Portland, Oregon, USA, June.

Hao Zhang, Liang Huang, Kai Zhao, and Ryan McDonald. 2013a. Online learning for inexact hypergraph search. In *Proceedings of the EMNLP*, pages 908–913, Seattle, Washington, USA, October. Association for Computational Linguistics.

Meishan Zhang, Yue Zhang, Wanxiang Che, and Ting Liu. 2013b. Chinese parsing exploiting characters. In *Proceedings of the 51st ACL*, pages 125–134, Sofia, Bulgaria, August.

Meishan Zhang, Yue Zhang, Wanxiang Che, and Ting Liu. 2014. Character-level Chinese Dependency Parsing. In *Proceedings of the 52st ACL*.

Guangyou Zhou and Jun Zhao. 2013. Joint inference for heterogeneous dependency parsing. In *Proceedings of the 51st ACL*, pages 104–109, Sofia, Bulgaria, August.