Transition-based Chinese Semantic Dependency Graph Parsing

Yuxuan Wang, Jiang Guo, Wanxiang Che, Ting Liu

School of Computer Science and Technology
Harbin Institute of Technology, Harbin, China, 150001
{yxwang, jguo, car, tliu}@ir.hit.edu.cn

Abstract. Chinese semantic dependency graph is extended from semantic dependency tree, which uses directed acyclic graphs to capture richer latent semantics of sentences. In this paper, we propose two approaches for Chinese semantic dependency graph parsing. In the first approach, we build a non-projective transition-based dependency parser with the SWAP-based algorithm. Then we use a classifier to add arc candidates generated by rules to the tree, forming a graph. In the second approach, we build a transition-based graph parser directly using a variant of the list-based transition system. For both approaches, neural networks are adopted to represent the parsing states. Both approaches yield significantly better results than the top systems in the SemEval-2016 Task 9: *Chinese Semantic Dependency Parsing.*

1 Introduction

Given a complete sentence, semantic dependency parsing aims at determining all the word pairs related to each other semantically and assigning specific predefined semantic relations. The results of semantic dependency parsing can be highly beneficial for question answering (*who did what to whom when and where*).

Chinese semantic dependency graphs are directed acyclic graphs (DAG) extended from traditional tree-structured representation of Chinese sentences. This graph structure can capture richer latent semantics. The goal of SemEval-2016 Task 9: *Chinese Semantic Dependency Parsing* is to identify such semantic structures from a corpus of Chinese sentences.[1] The task provides two distinguished corpora in the NEWS domain and the TEXTBOOKS domain. An example of semantic dependency graph is presented in Fig. 1. We can see that 他(he) has two head words, respectively 离开(leave) and 去(go) with the *Agent* (Agt) relation in the graph representation. More detailed information about the corpora will be reported in Section 4.

Transition-based dependency parsing has become increasingly popular in NLP because of its speed and accurate performance. This kind of parser construct dependency trees by using a sequence of transition actions over input sentences. They are mostly used to produce dependency trees, rather than graphs, or more accurately DAG in *Chinese Semantic Dependency Parsing.*

[1] http://alt.qcri.org/semeval2016/task9/

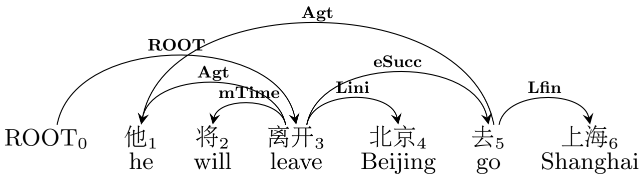

Fig. 1: An example of semantic dependency graph.

To address the challenge of DAG parsing, we investigate two approaches in this work. Our first approach has two stages. First, we adopt a set of linguistic-motivated rules to transform graphs to trees, based on which a traditional non-projective transition-based parser is trained. Then, we use a classifier to recover the extra arcs from a candidate arc set generated also by rules and add them to the parser's output to form dependency graphs. A similar approach was studied in [23], which applied pseudo-projective transformations [20] to transform non-projective dependencies to projective ones, and then use a dependency graph parser to parse projective graphs. This approach typically requires inconvenient pre-process and post-process.

We further propose a transition-based dependency graph parser that produce DAG directly for Chinese sentences. This parser is a variant of the list-based algorithm of [4], which was designed to parse dependency trees. In order to handle dependency graphs, we change the preconditions of the transitions to allow some words to have multiple head words. We have also tried to simplify the transition action set to investigate the relation between the parser performance and the number of transition actions. Extensive experiments show that both of our approaches obtain significantly better results compared with the top participated systems in the SemEval-2016 Task 9: *Chinese Semantic Dependency Parsing.*

2 Methods

Since most existing current dependency parsers only deal with dependency trees, it is a challenge to produce graphs with transition-based parser. To begin with, we propose a hybrid system which combines a traditional dependency tree parser and a binary classifier for complementing extra arcs. Then we propose a transition-based dependency parser that produce dependency graphs directly.

2.1 Tree-Based Method

In our first approach, we separate the task into two steps. In the first step, we use a traditional transition-based dependency parser to parse preprocessed semantic dependency trees. Then an SVM classifier is used to identify extra arcs that will be added to the parser results from candidate arc set created by rules.

Pre-process

Transforming DAGs to trees is typically a process of removing extra arcs of words which have multiple heads. We designed certain rules to remove such extra arcs in given semantic dependency graphs so that we can get semantic dependency trees for training the tree parser. These rules will be reused in the future to re-generate candidate arc set from trees produced by the tree parser.

These rules require human knowledge and are summarized by observing semantic dependency graphs in training data. We present them in the Appendix. Our rules can cover 95.5% graph situations in the NEWS corpus and 95.0% in the TEXTBOOKS corpus. The uncovered ones are basically all irregular situations, under which we hold the closest arc of the multi-head words.

For example, in causative sentences, a noun can be the object of the causative verb as well as the subject of the following verb, which introduces multiple heads to it. Fig. 2 shows an instance of this situation. For this kind of graph, we remove the dependency arcs between the noun and the following verb (介绍 (introduce)$\xrightarrow{Agt}$ 张先生 (Mr.Zhang)).

After dependency parsing, these rules are used again to re-generate candidate arcs from parsing results. For every verb pair with an arc between them, we find all noun children of the head verb "请 (invite)", and add arcs from the child verb "介绍 (introduce)" to these nouns to the candidate arc set (the two dashed arcs). According to the gold-standard graph, the label of the red arc is *Agt*, while the label of the blue one is NULL, meaning it should not be added to the final semantic dependency graph.

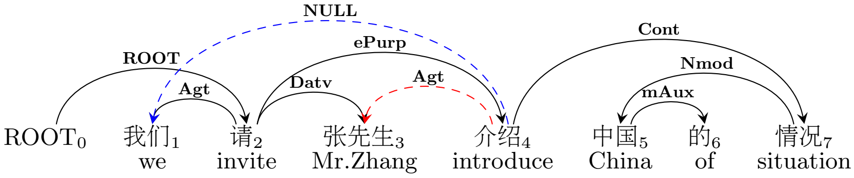

Fig. 2: A pre-process and post-process example of semantic dependency graph.

Dependency Parser

The semantic dependency trees transformed from DAGs are not necessarily projective. So we adopt the SWAP-based algorithm [19], which can parse non-projective trees with a time complexity that is quadratic in the worst case but still linear in the best case.

[3] proposed a transition-based dependency parser using neural networks as the classifier, which solved several problems caused by traditional sparse indicator features and yielded good results. Following their footsteps, we train a neural network classifier for use in a greedy, transition-based dependency parser. Their feature templates are used as the basic features in our experiment.

SVM Classifier for Post-process

The preprocessing rules mentioned above are used reversely to generate the training samples of SVM classifier from preprocessed semantic dependency trees. Labels of these samples include all possible relations of extra arcs and one *NULL* label indicating that the corresponding candidate arc should not be added to the tree. The features primarily include words, POS tags and distances. The specific feature templates are presented in [8].

2.2 Dependency Graph Parser

The above method is a compromise to the tree parser, which still requires careful pre-process and post-process. Moreover, for new dependency relation sets, the preprocessing rules need to be redesigned. We then present a transition-based dependency graph parser, which parses dependency graphs directly. In the Swap-based algorithm [19], we have to precompute the *projective order* which indicates when the Swap transition appears while generating oracle transition sequences. However, the coocurrence of non-projectivity and multiple heads makes it hard to compute the *projective order*. Here, we base our graph parser on the *list-based arc-eager algorithm* that was originally introduced to parse non-projective trees [4]. By simply modifying the preconditions of transitions, we make it capable of parsing dependency graphs. The specific preconditions are introduced in the next section.

We use a tuple $(\sigma, \delta, \beta, A)$ to represent each of the parsing state, where σ is a stack holding processed words, δ is a stack holding words popped out of σ that will be pushed back in the future, and β is a buffer holding unprocessed words. A is a set of labeled dependency arcs. We use index i to represent word w_i, and the index 0 represents the root of the graph w_0. It's important to notice that in this task all the roots w_0 of DAGs are required to have only one child. The initial state is $([0], [\,], [1, \cdots, n], \emptyset)$, while the terminal state is $(\sigma, \delta, [\,], A)$. During the parsing, arcs will only be generated between top element of σ, w_i, and the first element of β, w_j. The transition is generated by consulting the gold-standard trees during training and a neural network classifier during decoding.

Transition System 1

Table 1: Transitions in the *list-based arc-eager algorithm*.

Transitions	**Current State** $\Rightarrow$ **Next State**
Left$_l$-Reduce	$([\sigma\vert i], \delta, [j\vert\beta], A) \Rightarrow (\sigma, \delta, [j\vert\beta], A \cup \{(i \xleftarrow{l} j)\})$
Right$_l$-Shift	$([\sigma\vert i], \delta, [j\vert\beta], A) \Rightarrow ([\sigma\vert i\vert\delta\vert j], [\,], \beta, A \cup \{(i \xrightarrow{l} j)\})$
No-Shift	$([\sigma\vert i], \delta, [j\vert\beta], A) \Rightarrow ([\sigma\vert i\vert\delta\vert j], [\,], \beta, A)$
No-Reduce	$([\sigma\vert i], \delta, [j\vert\beta], A) \Rightarrow (\sigma, \delta, [j\vert\beta], A)$
Left$_l$-Pass	$([\sigma\vert i], \delta, [j\vert\beta], A) \Rightarrow (\sigma, [i\vert\delta], [j\vert\beta], A \cup \{(i \xleftarrow{l} j)\})$
Right$_l$-Pass	$([\sigma\vert i], \delta, [j\vert\beta], A) \Rightarrow (\sigma, [i\vert\delta], [j\vert\beta], A \cup \{(i \xrightarrow{l} j)\})$
No-Pass	$([\sigma\vert i], \delta, [j\vert\beta], A) \Rightarrow (\sigma, [i\vert\delta], [j\vert\beta], A)$

The transitions of the dependency graph parser is listed in Table 1, while corresponding preconditions are described in Table 2. Left$_l$-$*$ and Right$_l$-$*$ add an arc with label l from w_j to w_i, and vice versa. These transitions are performed only when one of w_i and w_j is the head of the other. Otherwise, No-$*$ will be performed. $*$-Shift is performed when no dependency exists between

Table 2: Preconditions of transitions in Table 1.

Transitions	**Preconditions**
Left$_l$-$*$	$[i \neq 0] \wedge \neg[(i \rightarrow^* j) \in A]$
Right$_l$-$*$	$\neg[(j \rightarrow^* i) \in A]$
$*$-Shift	$\neg[\exists k \in \sigma.(k \neq i) \wedge ((k \rightarrow j) \vee (k \leftarrow j))]$
$*$-Reduce	$[\exists h.(h \rightarrow i) \in A] \wedge \neg[\exists k \in \beta.(i \rightarrow k) \vee (i \leftarrow k)]$

w_j and any word in σ other than w_i, which pushes all words in δ and w_j into σ. $*$-Reduce is performed only when w_i has head and is not the head or child of any word in β, which pops w_i out of σ. $*$-Pass is performed when neither $*$-Shift nor $*$-Reduce can be performed, which moves w_i to the front of δ. $(i \xrightarrow{l} j)$ is used to denote an arc from w_i to w_j with label l. $(i \rightarrow j)$ and $(i \rightarrow^* j)$ indicate that w_i is a head and an ancestor of w_j respectively.

The preconditions of Left$_l$-$*$, Right$_l$-$*$ and $*$-Reduce are different from [4]'s algorithm. For Left$_l$-$*$ and Right$_l$-$*$, it's necessary to make sure no word has multiple heads in their tree parser, which is not required in our graph parser. For $*$-Reduce, we have to confirm that all of w_i's heads and children are found before removing it from σ. While they only need to check w_i's children, since w_i already has and can only have one head in tree structure.

Table 3: A transition sequence for dependency graph in Fig. 1 generated by *list-based arc-eager algorithm* with gold-standard graph.

State	**Transition**	σ	δ	β	A
0	Initialization	[0]	[ ]	$[1\|\beta]$	$\emptyset$
1	No-Shift	$[\sigma\|1]$	[ ]	$[2\|\beta]$	
2	No-Shift	$[\sigma\|2]$	[ ]	$[3\|\beta]$	
3	Left-Reduce	$[\sigma\|1]$	[ ]	$[3\|\beta]$	$A \cup \{2 \leftarrow \text{mTime} - 3\}$
4	Left-Pass	$[\sigma\|0]$	[1]	$[3\|\beta]$	$A \cup \{1 \leftarrow \text{Agt} - 3\}$
5	Right-Shift	$[\sigma\|3]$	[ ]	$[4\|\beta]$	$A \cup \{0 - \text{ROOT} \rightarrow 3\}$
6	Right-Shift	$[\sigma\|4]$	[ ]	$[5\|\beta]$	$A \cup \{3 - \text{Lini} \rightarrow 4\}$
7	No-Reduce	$[\sigma\|3]$	[ ]	$[5\|\beta]$	
8	Right-Pass	$[\sigma\|1]$	[3]	$[5\|\beta]$	$A \cup \{3 - \text{eSucc} \rightarrow 5\}$
9	Left-Reduce	$[\sigma\|0]$	[3]	$[5\|\beta]$	$A \cup \{1 \leftarrow \text{Agt} - 5\}$
10	No-Shift	$[\sigma\|5]$	[ ]	$[6\|\beta]$	
11	Right-Shift	$[\sigma\|6]$	[ ]	[ ]	$A \cup \{5 - \text{Lfin} \rightarrow 6\}$

A transition sequence for the sentence in Fig. 1 generated by the *list-based arc-eager algorithm* is presented in Table 3. In state 4, w_1 is moved from σ to δ because it has another head w_5 in β, and the arc between them will be generated in the future (state 9). In state 8, the transition is Right-Pass rather than Right-Shift because w_5 still has a child w_1 in σ, w_3 is moved to δ so that the arc between w_5 and w_1 can be generated (state 9). In states 3, 7 and 9, w_2, w_4 and w_1 are popped out of σ respectively because all of their children and heads have been generated.

Transition System 2

Table 4: Transitions and corresponding preconditions in the *simplified list-based algorithm.*

Transitions	Change of State & Preconditions
Left-Pop$_l$	$([\sigma\|i],\ \delta,\ [j\|\beta],\ A) \Rightarrow (\sigma,\ \delta,\ [j\|\beta],\ A \cup \{(i \xleftarrow{l} j)\})$ $[i \neq 0] \wedge \neg[(i \rightarrow^* j) \in A] \wedge \neg[\exists k \in \beta.(i \rightarrow k) \vee (i \leftarrow k)]$
Left-Arc$_l$	$([\sigma\|i],\ \delta,\ [j\|\beta],\ A) \Rightarrow (\sigma,\ [i\|\delta],\ [j\|\beta],\ A \cup \{(i \xleftarrow{l} j)\})$ $[i \neq 0] \wedge \neg[(i \rightarrow^* j) \in A]$
Right-Arc$_l$	$([\sigma\|i],\ \delta,\ [j\|\beta],\ A) \Rightarrow (\sigma,\ [i\|\delta],\ [j\|\beta],\ A \cup \{(i \xrightarrow{l} j)\})$ $\neg[(i \rightarrow^* j) \in A]$
Shift	$(\sigma,\ \delta,\ [j\|\beta],\ A) \quad \Rightarrow ([\sigma\|\delta\|j],\ [\ ],\ \beta,\ A)$ $\sigma = [\ ] \vee \neg[\exists k \in \sigma.(k \rightarrow j) \vee (k \leftarrow j)]$
Pass	$([\sigma\|i],\ \delta,\ [j\|\beta],\ A) \Rightarrow (\sigma,\ [i\|\delta],\ [j\|\beta],\ A)$ default transition

In order to improve the performance of our parser, we adopt another list-based arc-eager algorithm [5] which has less transitions which is referred to as *simplified list-based algorithm* henceforth.

Table 5: A transition sequence for dependency graph in Fig. 1 generated by *simplified list-based algorithm* with gold-standard graph.

State	Transition	σ	δ	β	A
0	Initialization	$[0]$	$[\ ]$	$[1\|\beta]$	$\emptyset$
1	Shift	$[\sigma\|1]$	$[\ ]$	$[2\|\beta]$	
2	Shift	$[\sigma\|2]$	$[\ ]$	$[3\|\beta]$	
3	Left-Pop	$[\sigma\|1]$	$[\ ]$	$[3\|\beta]$	$A \cup \{2 \leftarrow \text{mTime} - 3\}$
4	Left-Arc	$[\sigma\|0]$	$[1]$	$[3\|\beta]$	$A \cup \{1 \leftarrow \text{Agt} - 3\}$
5	Right-Arc	$[\ ]$	$[0, 1]$	$[3\|\beta]$	$A \cup \{0 - \text{ROOT} \rightarrow 3\}$
6	Shift	$[\sigma\|3]$	$[\ ]$	$[4\|\beta]$	
7	Right-Arc	$[\sigma\|1]$	$[3]$	$[4\|\beta]$	$A \cup \{3 - \text{Lini} \rightarrow 4\}$
8	Shift	$[\sigma\|4]$	$[\ ]$	$[5\|\beta]$	
9	Pass	$[\sigma\|3]$	$[4]$	$[5\|\beta]$	
10	Right-Arc	$[\sigma\|1]$	$[3, 4]$	$[5\|\beta]$	$A \cup \{3 - \text{eSucc} \rightarrow 5\}$
11	Left-Pop	$[\sigma\|0]$	$[3, 4]$	$[5\|\beta]$	$A \cup \{1 \leftarrow \text{Agt} - 5\}$
12	Shift	$[\sigma\|5]$	$[\ ]$	$[6\|\beta]$	
13	Right-Arc	$[\sigma\|4]$	$[5]$	$[6\|\beta]$	$A \cup \{5 - \text{Lfin} \rightarrow 6\}$

Left-Arc$_l$, Left-Pop$_l$, Right-Arc$_l$, Shift and Pass are transitions Left$_l$-Pass, Left$_l$-Reduce, Right$_l$-Pass, No-Shift and No-Pass in system 1 respectively. While Right$_l$-Shift and No-Reduce are discarded. In labeled parsing task like semantic dependency parsing, the number of transitions that generate arcs (Left$_l$-$*$, Right$_l$-$*$) equals to the number of label classes. Thus the number of transitions is reduced by a quarter for our task.

A transition sequence generated by *simplified list-based algorithm* for the same semantic dependency graph in Fig. 1 is presented in Table 5. Since Right-Shift is discarded, a Right-Arc and a Shift is used to replace it in the simplified algorithm, resulting in an extra transition for each Right-Shift (states 5 and 6, states 7 and 8). Since No-Reduce is discarded, the word that should be popped out of σ stays, which generate an extra Pass every time it is moved to δ (state 9). Generally speaking, we reduce the number of transitions at the price of longer transition sequences.

Feature Templates

Our feature templates are generally adapted from [3]. Considering the difference between dependency graphs and dependency trees, we add additional features regarding the heads of words on top of σ and in the front positions of β. The complete feature templates are presented in Table 6.

Table 6: Feature templates. (w−word; p−POS-tag; l−dependency label; $v_{lc}, v_{rc}, v_{lh}, v_{rh}$−valency; d−distance; c−cluster) S_i−word in σ; B_i−word in β; P_i−word in δ; lc, rc, lh, rh−left/rightmost child/head;llc, rrc, llh, rrh−leftmost of leftmost/rightmost of rightmost child/head.

Baseline Features
$S_0w, S_1w, B_0w, B_1w, P_0w, lc(S_0)w, rc(S_0)w, lh(S_0)w, rh(S_0)w, lc(B_0)w, lh(B_0)w, llc(S_0)w, rrc(S_0)w, llh(S_0)w, rrh(S_0)w, llc(B_0)w, llh(B_0)w; S_0p, S_1p, B_0p, B_1p, P_0p, lc(S_0)p, rc(S_0)p, lh(S_0)p, rh(S_0)p, lc(B_0)p, lh(B_0)p, llc(S_0)p, rrc(S_0)p, llh(S_0)p, rrh(S_0)p, llc(B_0)p, llh(B_0)p; lc(S_0)l, rc(S_0)l, lh(S_0)l, rh(S_0)l, lc(B_0)l, lh(B_0)l, llc(S_0)l, rrc(S_0)l, llh(S_0)l, rrh(S_0)l, llc(B_0)l, llh(B_0)l$
Valency & Distance
$S_0v_{lc}, S_0v_{rc}, S_0v_{lh}, S_0v_{rh}, B_0v_{lc}, B_0v_{lh}; d(S_0, B_0)$
Cluster
$S_0c, S_1c, B_0c, B_1c, P_0c, lc(S_0)c, rc(S_0)c, lh(S_0)c, rh(S_0)c, lc(B_0)c, lh(B_0)c, llc(S_0)c, rrc(S_0)c, llh(S_0)c, rrh(S_0)c, llc(B_0)c, llh(B_0)c$

Besides these baseline features, we add some non-local features including *valency*, *distance* and *cluster*. The numbers of left and right modifiers to a given head are identified as *left valency* and *right valency* respectively [30]. We extend the definition to graphs and use *left head valency* and *right head valency* to denote the numbers of left and right heads to a given modifier respectively. And the original valencies are then referred to as *left child valency* and *right child valency*. Brown Clustering [1] is a form of hierarchical clustering of words based on the contexts in which they occur and have proved beneficial for neural parsing [11]. We use pre-trained Brown clusters for each word involved in the baseline features. All of these new features are represented as embeddings and then pass through the neural network.

3 Experiments

3.1 Datasets

The SemEval-2016 Task 9 provides two distinguished corpora in the domain of NEWS and TEXTBOOKS. Detailed statics are presented in Table 7. The non-local dependencies [25] in the table refers to the dependency arcs which make dependency trees collapsed.

Table 7: Statics of the corpora.
(#g-sent is graph sentence number; #n-rate is non-local dependency rate.)

	NEWS				**TEXTBOOKS**			
	#sent	#word	#g-sent	#n-rate	#sent	#word	#g-sent	#n-rate
Train	8,301	250,249	3,615	4.8%	10,754	128,095	2,506	5.0%
Dev	534	15,325	223	4.2%	1,535	18,257	363	5.1%
Test	1,233	34,305	364	2.9%	3,073	36,097	707	5.0%

The evaluation measures of this task are on two granularity, dependency arc and the complete sentence. Labeled and unlabeled precision and recall with respect to predicted dependencies are used as evaluation measures. Since non-local dependencies are extremely difficult to discover, they are evaluated separately. For sentence level, labeled and unlabeled exact matches are used to measure sentence parsing accuracy. These metrics are abbreviated as:

- Labeled precision (LP), recall (LR), F1 (LF) and F1 score for non-local dependencies (NLF);
- Unlabeled precision (UP), recall (UR), F1 (UF) and F1 score for non-local dependencies (NUF);
- Labeled and unlabeled exact match (LM, UM).

When ranking systems, average F1 (LF) on testing sets are main references.

3.2 Results

The following hyper-parameters are used in all of our neural models: basic embedding size (for words, POS tags and dependency labels) $d = 50$, hidden layer size $h = 400$, dropout rate $p_d = 0.5$, regularization parameter $\lambda = 10^{-8}$, initial learning rate of Adagrad $\eta = 0.01$.

Table 8: Results of our systems and other participating systems in NEWS corpus

System	**LP**	**LR**	**LF**	**UP**	**UR**	**UF**	**NLF**	**NUF**	**LM**	**UM**
IHS-RD-Belarus	58.78	59.33	59.06	77.28	78.01	77.64	40.84	60.2	12.73	20.60
OCLSP (lbpg)	55.64	58.89	57.22	72.87	77.11	74.93	45.57	58.03	12.25	18.73
OCLSP (lbpgs)	58.38	57.25	57.81	76.28	74.81	75.54	41.56	54.34	12.57	20.11
OCLSP (lbpg75)	57.88	57.67	57.78	75.55	75.26	75.4	48.89	58.28	12.57	19.79
OSU CHGCG	55.52	55.85	55.69	73.51	73.94	73.72	**49.23**	60.71	5.03	11.35
Tree+SVM (base)	61.08	60.60	60.84	78.60	77.98	78.29	41.43	**63.62**	13.95	22.87
List-based (base)	60.44	59.85	60.14	77.69	76.94	77.31	46.80	62.50	13.46	22.38
Simplified (base)	60.65	60.09	60.37	77.98	77.25	77.61	43.41	59.31	13.06	21.65
Tree+SVM (all)	**61.47**	**61.01**	**61.24**	**79.19**	**78.60**	**78.89**	40.89	62.89	**14.36**	**23.03**
List-based (all)	61.02	60.40	60.71	78.26	77.47	77.86	47.01	61.90	13.06	22.47
Simplified (all)	61.13	60.40	60.76	78.42	77.48	77.95	46.34	61.93	13.22	22.47

The initial word embeddings and Brown clusters are trained using Xinhua portion of the Chinese Gigawords. We use 20-dimensional vector for cluster,

Table 9: Results of our systems and other participating systems in TEXT corpus

System	LP	LR	LF	UP	UR	UF	NLF	NUF	LM	UM
IHS-RD-Belarus	68.71	68.46	68.59	82.56	82.26	82.41	50.57	64.58	16.82	40.12
OCLSP (lbpg)	63.34	67.89	65.54	76.73	82.24	79.39	51.75	63.21	11.49	27.60
OCLSP (lbpgs)	67.35	65.11	66.21	81.22	78.52	79.85	47.79	55.51	12.82	33.29
OCLSP (lbpg75)	66.43	66.43	66.38	79.97	79.85	79.91	**57.51**	63.87	12.56	32.09
OSU CHGCG	65.36	64.98	65.17	79.06	78.60	78.83	54.70	65.71	11.36	32.02
Tree+SVM (base)	71.17	70.00	70.58	83.96	82.58	83.26	49.69	**68.56**	20.31	44.81
List-based (base)	70.05	68.58	69.31	82.42	80.68	81.54	53.57	63.96	19.23	41.46
Simplified (base)	70.63	69.19	69.90	82.59	80.91	81.74	54.24	64.55	20.08	42.01
Tree+SVM (all)	**71.35**	**70.19**	**70.76**	**84.23**	**82.87**	**83.54**	49.17	68.69	20.37	**45.10**
List-based (all)	70.76	69.34	70.04	82.85	81.19	82.01	55.59	65.55	20.31	42.56
Simplified (all)	71.01	69.59	70.29	82.87	81.22	82.04	54.48	65.59	**20.53**	42.43

and 10 for valency and distance. The number of Brown clusters are set to 256. The numbers of training samples for the SVM classifier in NEWS corpus and TEXTBOOKS corpus are 1,869,819 (6,748 positive) and 248,716 (3,542 positive) respectively. We use the liblinear toolkit [10] for training linear-kernel SVM classifiers. We conduct experiments with baseline features alone and with all features (baseline features, valency, distance and cluster) in both approaches. Table 8 and Table 9 show the results of our systems and other participating systems. The detailed description of other participating systems are presented in [2].

3.3 Discussions

The labeled F1 scores of our three systems with baseline features are higher than other participating systems. The first approach (Tree+SVM) has the best performance in LF and UF but worst in NLF. However, its NUF is higher than others. It is mainly due to the low precision of the SVM classifier, resulting from the small amount of the positive training samples. Some of the labels only occur very few times, making it difficult to predict the dependency labels of the extra arcs precisely. However, in the second approach (dependency graph parser), with lower NUF, we present much higher NLF scores, showing the high precision of dependency label prediction. This is because the labeling part is trained with all samples rather than non-local samples alone in dependency graph parser.

All of our systems perform better with richer non-local features. It is important to notice that the NUF and NLF of the first approach do not benefit from richer features which improve the non-local scores of the second approach a lot. This is probably because richer features only help improve the performance of tree parsing which do not provide non-local information. This is an advantage of our dependency graph parser, since we can improve the performance of entire structure prediction and non-local dependency prediction at the same time by using richer features.

With respect to non-local arcs, the *list-based arc-eager algorithm* works better than *simplified list-based algorithm* in most situations. This is because we reduce the number of transitions at the price of lengthening transition sequences in the *simplified list-based algorithm*. Longer transition sequences make it harder to discover non-local informations (NUF and NLF). However, smaller transition set provides higher precision for the entire structure prediction (UF and LF).

Generally speaking, although the first approach (Tree+SVM) performs better in LF and UF, it requires extra human knowledge to design preprocessing rules. So for new dependency relation sets, we have to redesign the rules manually in the first approach. Also its NLF is limited by the tree parser's performance. However, our dependency graph parser in the second approach can parse dependency graphs completely automatically and does not require extra human knowledge.

4 Related Works

Transition-based dependency parsing [28, 20, 29, 19, 13] makes structural predictions with a deterministic shift-reduce process. Most of these parsers are designed to parse dependency trees. [23] presented a transition-based approach to DAG parsing on the work of [20] on pseudo-projective transformations. Since their parser can only parse projective dependency graphs, their approach requires preprocess and post-process. [17] presented a DAG parser by extending the maximum spanning tree dependency framework of their early work in 2005, which is a graph-based dependency parser. Other parsing approaches that produce dependency graphs [6, 18, 22] are generally based on linguistically-motivated lexicalized grammar formalisms, such as HPSG, CCG and LFG.

Since [12]'s first attempt to use neural networks in a broad-coverage Penn Treebank parser, applying neural networks to dependency parsing and representing the states with dense embedding vectors has been more and more popular [26, 24, 3, 31, 27]. Recently, [9] proposed a technique for learning representations of parsing states in transition-based dependency parser with stack LSTM and yielded state-of-the-art performance.

Semantic dependency parsing integrates dependency structure and semantic information in the sentence based on dependency grammar [21]. [14] were the first to use dependency grammar in semantic analysis. Then [7] proposed Stanford typed dependencies representations. [15] [16] were the first to work on Chinese semantic dependency, and have manually annotated a corpus in the scale of one million words. HIT semantic dependency is established by Research Center for Social Computing and Information Retrieval in Harbin Institute of Technology in 2011. [8] refined the HIT dependency scheme with stronger linguistic theories, yielding a dependency scheme with more clear hierarchy.

5 Conclusion

We present two transition-based approaches for DAG parsing, and studied two kinds of transition set for the latter one. All of our systems yield significantly better LF than other participating systems in SemEval-2016 Task 9. We further provide extensive analysis and show the advantages and disadvantages of both approaches.

From the experiments, we can see that the performance of our graph parser can be further improved by using richer features. Therefore, our approach is expected to benefit from the recently proposed LSTM-based architectures [9]. We leave it to our future work.

References

1. Brown, P.F., Desouza, P.V., Mercer, R.L., Pietra, V.J.D., Lai, J.C.: Class-based n-gram models of natural language. Computational linguistics 18(4), 467–479 (1992)
2. Che, W., Shao, Y., Liu, T., Ding, Y.: Semeval-2016 task 9: Chinese semantic dependency parsing. In: Proceedings of the 10th international workshop on semantic evaluation (SemEval 2016), San Diego, US (forthcoming) (2016)
3. Chen, D., Manning, C.: A fast and accurate dependency parser using neural networks. In: Proc. of EMNLP. pp. 740–750 (2014)
4. Choi, J.D., McCallum, A.: Transition-based dependency parsing with selectional branching. In: Proc. of ACL. pp. 1052–1062 (2013)
5. Choi, J.D., Palmer, M.: Getting the most out of transition-based dependency parsing. In: Proc. of ACL. pp. 687–692. ACL (2011)
6. Clark, S., Hockenmaier, J., Steedman, M.: Building deep dependency structures with a wide-coverage ccg parser. In: Proc. of ACL. pp. 327–334. ACL (2002)
7. De Marneffe, M.C., Manning, C.D.: The stanford typed dependencies representation. In: COLING 2008: Proc. of the workshop on Cross-Framework and Cross-Domain Parser Evaluation. pp. 1–8 (2008)
8. Ding, Y., Shao, Y., Che, W., Liu, T.: Dependency graph based chinese semantic parsing. In: Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data, pp. 58–69. Springer (2014)
9. Dyer, C., Ballesteros, M., Ling, W., Matthews, A., Smith, N.A.: Transition-based dependency parsing with stack long short-term memory. In: Proc. of ACL and IJCNLP. pp. 334–343 (2015)
10. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: Liblinear: A library for large linear classification. The JMLR 9, 1871–1874 (2008)
11. Guo, J., Che, W., Yarowsky, D., Wang, H., Liu, T.: Cross-lingual dependency parsing based on distributed representations. 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). pp. 1234–1244 (July 2015)
12. Henderson, J.: Discriminative training of a neural network statistical parser. In: Proc. of ACL. p. 95. Association for Computational Linguistics (2004)
13. Huang, L., Sagae, K.: Dynamic programming for linear-time incremental parsing. In: Proc. of ACL. pp. 1077–1086. ACL (2010)
14. Johansson, R., Nugues, P.: Dependency-based syntactic-semantic analysis with propbank and nombank. In: Proc. of CoNLL. pp. 183–187. ACL (2008)
15. Li, M., Li, J., Dong, Z., Wang, Z., Lu, D.: Building a large chinese corpus annotated with semantic dependency. In: Proc. of SIGHAN. pp. 84–91. ACL (2003)

16. Li, M., Li, J., Wang, Z., Lu, D.: A statistical model for parsing semantic dependency relations in a chinese sentence. Chinese Journal of Computers 27(12), 1679–1687 (2004)
17. McDonald, R.T., Pereira, F.C.: Online learning of approximate dependency parsing algorithms. In: Proc. of EACL. pp. 81–88 (2006)
18. Miyao, Y., Tsujii, J.: Probabilistic disambiguation models for wide-coverage hpsg parsing. In: Proc. of ACL. pp. 83–90. ACL (2005)
19. Nivre, J.: Non-projective dependency parsing in expected linear time. In: Proc. of ACL and IJCNLP. pp. 351–359 (2009)
20. Nivre, J., Hall, J., Nilsson, J., Eryigit, G., Marinov, S.: Labeled pseudo-projective dependency parsing with support vector machines. In: Proc. of CoNLL. pp. 221–225. ACL (2006)
21. Robinson, J.J.: Dependency structures and transformational rules. Language pp. 259–285 (1970)
22. Sagae, K., Miyao, Y., Tsujii, J.: Hpsg parsing with shallow dependency constraints. In: ACL. vol. 45, p. 624 (2007)
23. Sagae, K., Tsujii, J.: Shift-reduce dependency dag parsing. In: Proc. of ICCL. pp. 753–760. ACL (2008)
24. Stenetorp, P.: Transition-based dependency parsing using recursive neural network-s. In: Deep Learning Workshop at NIPS (2013)
25. Sun, W., Du, Y., Kou, X., Ding, S., Wan, X.: Grammatical relations in chinese: Gb-ground extraction and data-driven parsing. In: ACL (1). pp. 436–456 (2014)
26. Titov, I., Henderson, J.: Constituent parsing with incremental sigmoid belief networks. In: ACL. vol. 45, p. 632 (2007)
27. Weiss, D., Alberti, C., Collins, M., Petrov, S.: Structured training for neural network transition-based parsing. In: Proc. of ACL and IJCNLP. pp. 323–333 (2015)
28. Yamada, H., Matsumoto, Y.: Statistical dependency analysis with support vector machines. In: Proc. of IWPT. pp. 195–206 (2003)
29. Zhang, Y., Clark, S.: A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In: Proceedings of the Conference on EMNLP. pp. 562–571. ACL (2008)
30. Zhang, Y., Nivre, J.: Transition-based dependency parsing with rich non-local features. In: Proc. of ACL. pp. 188–193 (2011)
31. Zhou, H., Zhang, Y., Chen, J.: A neural probabilistic structured-prediction model for transition-based dependency parsing. In: Proc. of ACL. pp. 1213–1222 (2015)

Appendix

Preprocessing rules used in our first approach:

- In a causative sentence, we remove the dependency arc between the following verb and the noun;
- For multi-head situation caused by dependency arc between a noun and a pronoun, we remove this arc to eliminate the situation;
- For multi-head situation caused by reverse relation between a noun and a verb, we remove this arc to eliminate the situation;
- For multi-head situation caused by possessor relation between two nouns, we remove this arc to eliminate the situation.