Improving Multi-Pass Transition-Based Dependency Parsing using Enhanced Shift Actions

Chenxi Zhu, Xipeng Qiu, and Xuanjing Huang

[1] Shanghai Key Laboratory of Intelligent Information Processing
[2] School of Computer Science, Fudan University, Shanghai, China
{13210240078, xpqiu,xjhuang}@fudan.edu.cn

Abstract. In multi-pass transition-based dependency parsing algorithm, the *shift* actions are usually inconsistent for the same node pair in different passes. Some node pairs have a indeed dependency relation, but the modifier node has not been a complete subtree yet. The bottom-up parsing strategy requires to perform *shift* action for these node pairs. In this paper, we propose a method to improve performance of parsing by using enhanced *shift* actions. These actions can be further used as features for the next parsing decision. Experimental results show that our method is effective to improve the performance of parsing.

1 Introduction

In recent years, data-driven deterministic dependency parsing has received an increasing amount of interests. Dependency parsing uses dependency representation of syntactic structure, which directly reflects relationships among the words in a sentence. Dependency parsing is usually more efficient and accurate than constituency parsing, since dependency trees are inherently lexicalized and do not need full structure grammar and extra non-terminal nodes. Therefore, dependency parsing is widely used in a variety of practical tasks, especially for web-scale data.

Currently, there are two main kinds of approaches for data-driven dependency parsing: graph-based [10] and transition-based methods [18, 12].

The graph-based methods generate dependency trees by considering all possible spanning trees [10]. Graph-based methods usually consist of two stages. In the scoring stage, a classifier is used to score all possible edges (or other small substructures), in the decoding stage, the highest scoring parse tree is found from all possible outputs by combinatorial optimization algorithm. When the dependency tree is projective, the time complexity is $O(n^2)$ for first-order models using a minimum directed spanning tree algorithm [4, 8].

The transition-based methods use a classifier to make greedy decisions of parsing actions locally [18, 12]. Transition-based parsing gives complexities as low as $O(n)$ or $O(n^2)$ for parsing.

Although the transition-based methods usually run fast and performs accurately, they suffer the problems of error propagation. Therefore, their performances are slightly below the graph-based methods[2, 11]. However, recent researches on transition-based dependency parsing have therefore explored different ways of improving their accuracy [19, 1, 17]. With these methods, transition-based parsers have reached state-of-the-art accuracy for a number of languages.

The state-of-the-art transition-based parsers use only one shift action, which obviously loses some information which will be helpful for the dependency parsing. Especially in multi-pass transition-based dependency parsing algorithm, the *shift* actions are usually inconsistent for the same node pair in different passes. Some node pairs have a indeed dependency relation, but the modifier node has not been a complete subtree yet. The bottom-up parsing strategy requires to perform *shift* action for these node pairs.

In this paper, we explore a new approach to improve the accuracy of transition-based parsers by using the enhanced shift actions, which combine the actions and the relations between the target nodes. Our method can give a link between actions and the real relations between the target nodes. While all previous transition systems ignore shift information, our new system distinguishes the shift action by the relation between the two target nodes. With the more detailed actions, the parser can predict more accurately when it has to decide whether the action should be left or right after several *shift* actions between the target nodes. Besides, the previous actions can be helpful for next decision by using these actions as features.

The experiments show that our method leads to significant improvements in parsing accuracy, compared to a baseline parser. Our parser also performs better than Maltparser[12], which is one of the most widely used state-of-the-art transition-based systems.

The rest of the paper is organized as follows: We first describe the multi-pass transition-based parsing in section 2 and give a discussion for the shift action used in current state-of-the-art algorithm in section 3. Then we propose our method on section 4. The experimental results are given in section 5. Finally, we introduce the related work in section 6 and conclude our work in section 7.

2 Multi-pass Transition-Based Dependency Parsing

Our work is based on the Yamada's algorithm [18], which performs multi-pass scans of a partially built dependency structure.

In a transition-based system, an input sentence is usually processed from left to right. Yamada's algorithm uses a deterministic analyzing model based on shift-reduce algorithm. It performs multi-pass scans of a partially built dependency structure. At each point, it focuses on a pair of adjacent nodes in the partial dependency structure and uses a support vector machine to decide whether to create a dependency link between them or to shift the focus to the next pair of heads.

There are three following deterministic actions:

shift No construction of dependencies between these target nodes, and the position of focus simply moves to the right.

left A dependency relation is constructed between two neighboring nodes where the right node of target nodes becomes a child of the left one.

right A dependency relation is constructed between two neighboring nodes where the left node of target nodes becomes a child of the right one.

Yamada' parsing algorithm consists of three steps: (i) Estimate the appropriate parsing actions using contextual information surrounding the target nodes, (ii) constructs a dependency tree by executing the estimated actions, (iii) if there is no construction in one pass, get the left or right action which has the max probability in the dependency tree to construct.

The pseudo-code of our parsing algorithm is shown in Algorithm 1.

We use T to represent the sequence of nodes consisting of m elements $t_m(m \leq n)$, each element of which represents the root node of a sub-tree constructed in the parsing process (initially all t_i is a word w_i).

During the execution, the parsing action $y_i \in \{\mathit{Right}; \mathit{Left}; \mathit{Shift}\}$ is estimated for the focus nodes pair $< t_i, t_{i+1} >$. Meanwhile, we use y'_i to record the substitute action if $y_i = \mathit{shift}$. y'_i is the action with second largest score s'_i given by a classifier.

If every action $y_i = \mathit{shift}$ for $i = 1, \cdots, T-1$ in one pass, no construction is executed and the variable $\mathit{no\_construction} = \mathit{true}$. We select the action y'_j and make the construction. $j = \arg\max_i s'_i$.

The contextual features x are extracted from the context surrounding $< t_i, t_{i+1} >$. Then an appropriate parsing action $y \in \{\mathit{Right}; \mathit{Left}; \mathit{Shift}\}$ is estimated by one or more classifiers.

The complexity of the algorithm is $O(n^2)$ in worst case because it takes up to $n-1$ passes to construct a complete dependency tree. However, it runs in linear time in practice.

3 Discussion of Shift Action in Multi-Pass Transition-based Parsing

Due to the bottom-up regulation, a dependency relation $h \leftarrow d$ can be constructed only after all modifiers of the d have been constructed. Therefore, there are two cases for *shift* action in Yamada's algorithm: (a) there is no dependencies relation between the node pair, (b) there is dependency relation between the node pair, but the modifier node has not been a complete subtree yet. These two cases are shown in Figure 1. ABC is a word sequence.

Shift action is executed for pair of (A, B) for both cases at the first pass. However, for case (b), it need predict to take left action at node pair (A, B) at the second pass. The difference between two predictions is whether C has been attached as a dependent of node B. This might cause a degradation of performance since that the prediction is often made with linear classifier. The features extracted from this two situations have a lot of overlap. So the prediction

```
input  : Input Sentence: $(w_1, w_2 \cdots w_n)$
output: dependency tree T and $|T| = 1$
Initialize:
$i = 1$ ;
$T = w_1, w_2 \cdots w_n$ ;
no_construction = *true* ;
while $|T| \geq 1$ do
    if $i == |T|$ then
        if *no_construction == true* then
            find the substitute action $y'_j$ with largest score $j = \arg\max_i s'_i$
            construct(T, j, $y'_j$);
            no_construction = false ;
            $i = j$ ;
        end
    end
    get the contextual features $x$;
    estimate the substitute action $y$;
    construction(T, i, y) ;
    if  $y ==$ *left or right* then
        no_construction = false ;
    else
        estimate the substitute action $y'_i$ and its score $s'_i$ ;
    end
end
return $T$ ;
```

Algorithm 1: Parsing Algorithm

of (A, B) is very difficult and may cause error propagation. If it predicts *left* action on (A, B) at the first pass, the predict head of C will be never right due to the bottom-up regulation.

The main reason of the problem is that the parser ignores the potential relation between the target node pair when it predicts a *shift* action. In order to cope with this problem, Yamada and Matsumoto [18] suggested to divide the action *shift* into two kinds of actions, *shift'* and *wait* for two cases respectively. These new actions are the same behavior in parsing process. However, they do not report any experimental result for their suggestion.

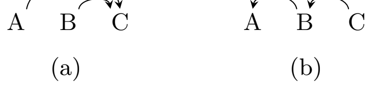

Fig. 1: Two Cases of Shift Action (The arc is from the modifier to its governor.)

4 Dependency Parsing using Enhanced Shift Actions

In this paper, we deal with this problem with two improvements. First, we modify the actions of the transition system. Second, we extend the feature set for parsing based on our improved actions.

4.1 Enhanced Shift Actions

We expand the parsing action by adding two actions: *shift-left* and *shift-right*.

left let the right node become a child of the left one.
right let the left node become a child of the right one.
shift let the point of focus moves to the right. And there is no relation between target nodes.
shift-right let the point of focus moves to the right. And there is a right relation between target nodes but we can just shift because dependent child still have further dependent children in the sentence.
shift-left let the point of focus moves to the right. And there is a left relation between target nodes.

Compared to set of actions of Yamada's algorithm, we split the original *shift* action into *shift*, *shift-left* and *shift-right*, which is used to distinguish the relation between the target node pair. The *shift-left* and *shift-right* actions can be also called *pseudo shift* actions.

We count these three *shift* actions in parsing process on Chinese dataset of the CoNLL 2009 shared task[9]. The *pseudo shift* actions are more than 20%. The detailed information is shown in Table 1.

Table 1: Statistics of Three Shift-related Actions

(a) Training Dataset

	Number	Percent
Shift	694460	77.68%
Shift-Left	156644	17.52%
Shift-Right	42886	4.80%
Total	893990	

(b) Test Dataset

	Number	Percent
Shift	83455	77.77%
Shift-Left	20104	18.50%
Shift-Right	5139	4.73%
Total	108698	

Figure 2 gives real examples *shift-left* and *shift-right* actions.

4.2 Feature Extraction

Let i and $i+1$ be the indexes of the left and right of target nodes in T. The left context is defined as the nodes on the left side of the target nodes: $t_l(l < i)$, and the right context is defined as those on the right: $t_r(i+1 < r)$. Context

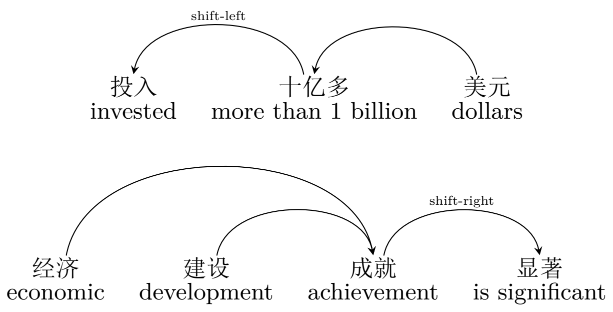

Fig. 2: Examples of *shift-left* and *shift-right* Actions(The arc is from the modifier to its governor.)

lengths (l, r) represents the numbers of nodes within the left and right contexts. We choose the context lengths (2, 2).

A feature can be represented as triplet (p, k, v), in which p is the position from the target nodes, k denotes the feature type, and v is the value of the feature. If $p < 0$, it represents the node in the left context, $p = 0-; 0+$ denotes the left or right node of target nodes. $p > 0$ denotes those in the right context. The feature type k and its value v are summarized in the Table 2.

Besides the traditional features, we also extract the features based on previous action. These features are shown in bold in Table 2. If the last action is *shift*, *shift-left* or *shift-right*(referred as *shift-x*), the parser will extract the shift information of last action including la, la-L-pos, la-L-lex, la-R-pos, la-R-lex. And p is set to 0 to make a formal unity. The information about last action will be a good reference to predict the current action.

4.3 Training

We use online Passive-Aggressive (PA) algorithm [6, 7] to train the model parameters. Following [5], the average strategy is used to avoid the overfitting problem.

First, we need to build instance list of training data sets $(\mathbf{x}, \mathbf{y})$. The algorithm of building instance list is just like the parsing algorithm. Here, x is the contextual features and y is the parsing action. We can get all the instances from training corpus.

Given an example $(\mathbf{x}, \mathbf{y})$, $\hat{\mathbf{y}}$ are denoted as the labels with the highest score

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{z}} \mathbf{w}^T \Phi(\mathbf{x}, \mathbf{z}). \tag{1}$$

where $\mathbf{w}$ is the parameter of score function, and $\Phi(\mathbf{x}, \mathbf{y})$ is the feature vector consisting of lots of overlapping features, which is the chief benefit of discriminative model.

For the training data sets, given an example $(\mathbf{x}_i, \mathbf{y}_i)$, we compare the $\mathbf{y}_i$ and $\hat{\mathbf{y}}_i$ as the hinge-loss function. If $\hat{\mathbf{y}}_i = \mathbf{y}_i$, the resulting algorithm is passive, that

	Type	Value
Traditional Features	pos	POS tag string
	lex	word string
	ch-L-pos	POS tag string of the child node modifying to the parent node from left side
	ch-L-lex	word string of the child node node modifying to the parent node from left side
	ch-R-pos	POS tag string of the child node modifying to the parent node from right side
	ch-R-lex	word string of the child node modifying to the parent node from right side
Action-based Features	**la**	last action if it is *shift-x*
	la-L-pos	POS tag string of the left target node in the last action
	la-L-lex	word string of the left target node in the last action
	la-R-pos	POS tag string of the right target node in the last action
	la-R-lex	word string of the right target node in the last action

Table 2: Summary of the features

is, $\mathbf{w}_{k+1} = \mathbf{w}_k$ as we expected. In contrast, when $\hat{\mathbf{y}}_i \neq \mathbf{y}_i$, the algorithm aggressively forces $\mathbf{w}_{k+1}$ to satisfy the constraints. We define the following constrained optimization problem to update the new weight vector $\mathbf{w}_{k+1}$ based on weight vector $\mathbf{w}_k$. The **margin** $\gamma(\mathbf{w}; (\mathbf{x}, \mathbf{y}))$ is defined as

$$\gamma(\mathbf{w}; (\mathbf{x}, \mathbf{y})) = \mathbf{w}^T \Phi(\mathbf{x}, \mathbf{y}) - \mathbf{w}^T \Phi(\mathbf{x}, \hat{\mathbf{y}}). \tag{2}$$

Thus, we calculate the **hinge loss** $\ell(\mathbf{w}; (\mathbf{x}, \mathbf{y}))$, (abbreviated as ℓ_w) by

$$\ell_w = \begin{cases} 0, & \gamma(\mathbf{w}; (\mathbf{x}, \mathbf{y})) > 1 \\ 1 - \gamma(\mathbf{w}; (\mathbf{x}, \mathbf{y})), & \text{otherwise} \end{cases} \tag{3}$$

In round k, the new weight vector $\mathbf{w}_{k+1}$ is calculated by

$$\mathbf{w}_{k+1} = \arg\min_{\mathbf{w}} \frac{1}{2} ||\mathbf{w} - \mathbf{w}_k||^2 + \mathcal{C} \cdot \xi, \qquad \textbf{s.t. } \ell(\mathbf{w}; (\mathbf{x}_k, \mathbf{y}_k)) \leq \xi \textbf{ and } \xi \geq 0 \tag{4}$$

where ξ is a non-negative slack variable, and $\mathcal{C}$ is a positive parameter which controls the influence of the slack term on the objective function.

Following the derivation in PA [7], we can get the update rule,

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \tau_k(\Phi(\mathbf{x}_k, \mathbf{y}_k) - \Phi(\mathbf{x}_k, \hat{\mathbf{y}}_k)), \tag{5}$$

where

$$\tau_k = \min(\mathcal{C}, \frac{\ell_{w_k}}{\|\Phi(\mathbf{x}_k, \mathbf{y}_k) - \Phi(\mathbf{x}_k, \hat{\mathbf{y}_k})\|^2}) \tag{6}$$

5 Experiments

Our experiments were conducted on Chinese dataset of the CoNLL 2009 shared task[9]. The detailed information is shown in Table 3. FORM/LEMMA OOV (out-of-vocabulary) is the percentage of FORM/LEMMA tokens not found in the respective vocabularies derived solely from the training data.

Table 3: Data Statistics for the Chinese Dataset of CoNLL-2009 Shared Task

Training data size (sentences)	22,277
Training data size (tokens)	609,060
Avg. sentence length (tokens)	27.3
Evaluation data size (sentences)	2,556
Evaluation data size (tokens)	73,153
Evaluation FORM OOV	3.92%
Evaluation LEMMA OOV	3.92%

We compare the accuracy of our model against Maltparser [14] and Yamada's algorithm. The MaltParser is an efficient deterministic dependency parser that is gaining popularity. Similar to Yamada's algorithm, the MaltParser relies on a discriminative classifier to choose its actions. However, by employing the arc-eager algorithm presented in [12], the parser can build the complete parse tree in a single pass and therefore it guarantees $O(n)$ complexity in the worst case.

We trained all parsers up to 50 iterations.

The following measures were used to evaluate parsing performance.

$$UAS = \frac{\langle \sharp \text{ of tokens with correct heads} \rangle}{\langle \sharp \text{ of tokens} \rangle}$$

$$LAS = \frac{\langle \sharp \text{ of tokens with correct heads and labels} \rangle}{\langle \sharp \text{ of tokens} \rangle}$$

$$CM = \frac{\langle \sharp \text{ of correctly parsed sentences} \rangle}{\langle \sharp \text{ of sentences } \rangle}$$

The test results are shown in Table 4.

Considering UAS, our method provides an improvement of 1.66 over the baseline and 0.72 over the maltparser. Considering LAS, we achieve improvements of 1.71 and 0.86, respectively.

The improvement of our method can be ascribed to two contributions.

The first contribution is the enhanced actions, which is introduced in section 4.1. These enhanced actions avoid the inconsistent actions for the same nodes pair in different passes. Furthermore, they also improve the accuracy of classifier.

The second contribution is the new features based on the additional actions by recording last shift action, which introduced in section 4.2. When the previous action is *shift-left*, we can derive that the current action may not be *right*. Otherwise, there would be two heads for one node.

Table 4: Accuracy on test set, excluding punctuation, for unlabeled attachment score (UAS), labeled attachment score (LAS) and Complete Match (CM)

	UAS	LAS	CM
Maltparser	82.31	80.64	28.17
Top CoNLL 2009[15] (Transition-based)	81.22	79.19	-
Che et al. [3] (Graph-based)	-	75.49	-
Baseline(Yamada's algorithm)	81.37	79.79	26.41
Baseline + enhanced actions	82.59	81.07	27.66
Baseline + enhanced actions + action-based features	**83.03**	**81.50**	**29.11**

6 Related Works

Yamada and Matsumoto [18] is in the light of the target nodes, and determines what kind of action to do. This can be treated as a multi-class classification problem. It also uses the the left and right context information of the target nodes and their children's information to exclude ambiguity. However, this method suffers from the inconsistent action problem. It is caused by the strict bottom-up strategy, which requires each node to have found all its dependents before it is combined with its head.

Nivre [13] handles the inconsistent action problem by modifying the parsing strategy. It combines the bottom-up and top-down processing. It has four actions: Left-arc, Right-arc, Reduce and Shift. And it to some extent eases the contradiction between restrict parsing framework and the structure of the dependency tree.

Rush and Petrov [16] propose a multi-pass coarse-to-fine architecture for dependency parsing using linear-time vine pruning and structured prediction cascades. Their first-, second-, and third-order models achieve accuracies comparable to those of their unpruned counterparts, while exploring only a fraction of the search space.

Sartorio et al. [17] present a transition-based, greedy dependency parser which implements a flexible mix of bottom-up and top-down strategies. The new strategy allows the parser to postpone difficult decisions until the relevant information becomes available.

7 Conclusion

In this paper, we investigate to use the enhanced shift actions to improve the transition-based parsing. Our method is based on the multi-pass parsing algorithm. The experiments show the effectiveness of our parser over the baseline and Maltparser.

Our future work is to explore more in the shift information to improve the accuracy of the transition-based dependency parser. We also wish to apply our strategy in the one-pass transition-based parsing algorithm.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable comments. This work was funded by NSFC (No.61003091) and Science and Technology Commission of Shanghai Municipality (14ZR1403200).

References

[1] Bohnet, B.: Very high accuracy and fast dependency parsing is not a contradiction. In: Proceedings of the 23rd International Conference on Computational Linguistics. pp. 89–97. Association for Computational Linguistics (2010)

[2] Buchholz, S., Marsi, E.: Conll-x shared task on multilingual dependency parsing. In: Proceedings of the Tenth Conference on Computational Natural Language Learning. pp. 149–164. Association for Computational Linguistics (2006)

[3] Che, W., Li, Z., Li, Y., Guo, Y., Qin, B., Liu, T.: Multilingual dependency-based syntactic and semantic parsing. In: Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task. pp. 49–54. Association for Computational Linguistics (2009)

[4] Chu, Y.J., Liu, T.H.: On shortest arborescence of a directed graph. Scientia Sinica 14(10), 1396 (1965)

[5] Collins, M.: Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In: Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (2002)

[6] Crammer, K., Singer, Y.: Ultraconservative online algorithms for multiclass problems. Journal of Machine Learning Research 3, 951–991 (2003)

[7] Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., Singer, Y.: Online passive-aggressive algorithms. Journal of Machine Learning Research 7, 551–585 (2006)

[8] Edmonds, J.: Optimum branchings. Journal of Research of the National Bureau of Standards B 71(233-240), 160 (1967)

[9] Hajič, J., Ciaramita, M., Johansson, R., Kawahara, D., Martí, M., Màrquez, L., Meyers, A., Nivre, J., Padó, S., Štěpánek, J., et al.: The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. In: Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task. pp. 1–18. Association for Computational Linguistics (2009)

[10] McDonald, R., Crammer, K., Pereira, F.: Online large-margin training of dependency parsers. In: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics. pp. 91–98 (2005)

[11] Nilsson, J., Riedel, S., Yuret, D.: The conll 2007 shared task on dependency parsing. In: Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL. pp. 915–932. sn (2007)

[12] Nivre, J.: An efficient algorithm for projective dependency parsing. In: Proceedings of the 8th International Workshop on Parsing Technologies (IWPT). Citeseer (2003)

[13] Nivre, J.: Incrementality in deterministic dependency parsing. In: Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together. pp. 50–57. Association for Computational Linguistics (2004)

[14] Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., Kübler, S., Marinov, S., Marsi, E.: Maltparser: A language-independent system for data-driven dependency parsing. Natural Language Engineering 13(2), 95–135 (2007)

[15] Ren, H., Ji, D., Wan, J., Zhang, M.: Parsing syntactic and semantic dependencies for multiple languages with a pipeline approach. In: Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task. pp. 97–102. Association for Computational Linguistics (2009)

[16] Rush, A.M., Petrov, S.: Vine pruning for efficient multi-pass dependency parsing. In: Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. pp. 498–507. Association for Computational Linguistics (2012)

[17] Sartorio, F., Satta, G., Nivre, J.: A transition-based dependency parser using a dynamic parsing strategy. In: Proceeding of the 51st Annual Meeting of the Association for Computational Linguistics (2013)

[18] Yamada, H., Matsumoto, Y.: Statistical dependency analysis with support vector machines. In: Proceedings of the International Workshop on Parsing Technologies (IWPT). vol. 3 (2003)

[19] 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 Empirical Methods in Natural Language Processing. pp. 562–571. Association for Computational Linguistics (2008)