Layer-Based Dependency Parsing*

Ping Jian and Chengqing Zong

Institute of Automation, Chinese Academy of Sciences
No. 95 Zhongguancun East Road, Beijing, 100190, China
{pjian, cqzong}@nlpr.ia.ac.cn

Abstract. In this paper, a layer-based projective dependency parsing approach is presented. This novel approach works layer by layer from the bottom up. Inside the layer the dependency graphs are searched exhaustively while between the layers the parser state transfers deterministically. Taking the dependency layer as the parsing unit, the proposed parser has a lower computational complexity than graph-based models which search for a whole dependency graph and alleviates the error propagation that transition-based models suffer from to some extent. Furthermore, our parser adopts the sequence labeling models to find the optimal sub-graph of the layer which demonstrates that the sequence labeling techniques are also competent for hierarchical structure analysis tasks. Experimental results indicate that the proposed approach offers desirable accuracies and especially a fast parsing speed.

Keywords: dependency parsing, dependency layer, sequence labeling

1 Introduction

Graph-based models (McDonald et al., 2005; McDonald and Pereira, 2006) and transition-based models (Yamada and Matsumoto, 2003; Nivre and Scholz, 2004) are two dominant paradigms in the dependency parsing community. McDonald and Nivre (2007) have made elaborate analyses about the very different theoretical properties of these two kinds of models and the corresponding experimental behaviors.

Generally, graph-based approaches learn a model for scoring possible dependency graphs of an input sentence and apply exhaustive search algorithms to find the one that maximizes the score. The unit graph-based models calculate is the whole sentence (the whole dependency graph) both in training and inference procedures, which results in a cubic computational complexity (in projective case). By contrast, transition-based approaches train a classifier to greedily choose the best parsing action under the current parser state. They make decisions at a configuration which is usually composed by a couple of focus tokens and the parsing contexts. Therefore, these two kinds of dependency parsing methods represent the two extremes when they seek the best dependency structure of the input sentence. In this paper, we adopt a moderate structural granularity to calculate the parser: a *dependency layer*.

The dependency layer we mean here is a set of tokens whose dependency depth (the depth of the dependency tree) is at most one. Inside the layer the dependency graphs can be searched exhaustively while between the layers the parser state transfers deterministically. On one hand, this design will decrease the computational cost for searching the whole tree like graph-based models do; on the other hand, it may alleviate the error propagation resulting from the complete no "search" outside the parsing configuration in transition-based models.

* The research work has been partially funded by the Natural Science Foundation of China under grant No.60736014, 60723005 and 90820303, the National Key Technology R&D Program under grant No. 2006BAH03B02, the Hi-Tech Research and Development Program (863 Program) of China under grant No. 2006AA010108-4, and also supported by the China-Singapore Institute of Digital Media as well.

It is well known that chunking, which is deemed to be a useful and tractable precursor to full parsing, has been successfully handled by sequence labeling techniques (Kudo and Matsumoto, 2001; Sha and Pereira, 2003). Inspired by this scheme, we adopt the globally optimal sequence labeling to search the best depth-one sub-graph in the dependency layer. We believe that the line-typed sequential models are potent complementarities to the tree-typed hierarchical ones or even the latent substitutes.

The experiments show that our layer-based parser yields comparable dependency attachment accuracies to the state-of-the-art dependency parsers on both English and Chinese datasets. Especially, it is quite efficient due to the layer-based search and sequence typed analysis. The remainder of the paper is organized as follows: Section 2 describes the details of the algorithm and feature set. Section 3 presents the experimental results. The related work is discussed in Section 4. Conclusion and future work comprise Section 5.

2 Layer-based Parsing Approach

2.1 Algorithms

Wu et al. (2007) designed a neighbor parser to identify the neighboring parent-child relations between two consecutive tokens in the input sentence. Following their framework we label the dependency relations in our parsing layer. An example is shown in Figure 1(a). The first and second columns represent the words and part-of-speech (POS) tags respectively. The third column implies whether the token modifies its left neighbor (LH, left-headed) or right neighbor (RH, right-headed) or neither (O). The string behind the character "_" indicates the dependency type of the neighboring link.

The	DT	RH_NMOD
chair	NN	O
of	IN	LH_NMOD
the	DT	RH_NMOD
conference	NN	O
declared	VBD	O
that	DT	RH_NOMD
decision	NN	O

(a)

The	DT	RH_NMOD	r
chair	NN	O	o
of	IN	LH_NMOD	o
the	DT	RH_NMOD	r
conference	NN	O	o
declared	VBD	O	o
that	DT	RH_NOMD	r
decision	NN	O	o

(b)

Figure 1: Example of (a) the sequential neighboring relation labeling, (b) the *reduce* decision labeling.

Wu et al. (2007) employed linear chain conditional random fields (CRFs) as the labeling algorithm to capture the higher order features and avoid the greedy search when labeling with sequential classifiers (Cheng et al., 2006). To prevent the error propagation, they regarded the labeling results as features of the subsequent parsing stage instead of reducing the child words. However, this weakens the strength that neighboring parsing can provide. In our approach, besides the CRF-based relation labeler, an additional tagger is introduced to examine whether a dependent child can be reduced, i.e., whether it has found its head and has already been a complete sub-tree. The *reduce* tagger tries to guarantee safe reductions and ensures the parsed structures can be formed into a tree after several passes of analysis. In Figure 1(b), the letter "r" in the rightmost column implies that the corresponding token will be reduced while others are reserved for the next stage.

The *reduce* tagger is also trained by linear chain CRFs to fulfill the globally optimal property of the layer-based labeling. Specially, when continuous attachments happen in the same direction, only the lowest child token is reduced although other tokens in this chain are complete sub-trees after the current labeling. This enables the tokens to change their

attachments when the context is refreshed in the next layer. For example, in Figure 2, if the parent word "of" and the child word "conference" are far from each other in the early parsing stage, the child "conference" may be wrongly attached to the word "declared" (Figure 2(a)) because long distance interrelations are difficult to be caught in sequence labeling models. If the tagger is learned to only reduce the lowest child token each time, i.e., the leftmost word "the", the word "conference" has the chance to adjoin "of" and be attached correctly at last (Figure 2(b)).

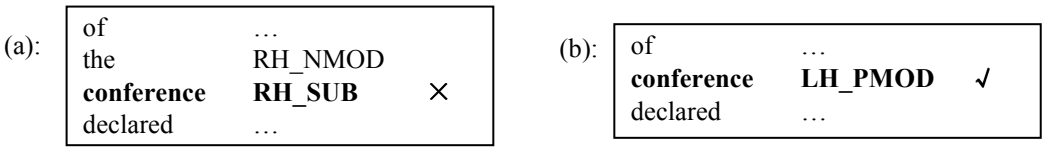

Figure 2: Long dependency attaching error in neighboring relation labeling

As the dependency relations only exist between adjacent tokens and all the survivals will be relabeled in the next layer, the dependency depth of the layer is at most one.

Pseudo-code of the parsing algorithm is given as follows:

```
Input sentence: w1, w2, …, wn
Initialize:
  L = {w1, w2, …, wn};
  have_reduce = false;
Start:
While |L|>1 do begin
  x = get_feature (L);
  y1 = estimate_relation (model1, x);
  y2 = estimate_reduce (model2, x, y1);
  have_reduce = sign(count_reduce(y2));
  if(have_reduce == true)
    reduce (L, y2);
    have_reduce = false;
  else break;
  end;
end.
```

At each processing stage, two functions, *estimate_relation* and *estimate_reduce*, are employed to label the sequence L with neighboring dependency relations ($\mathbf{y}_1$) and *reduce* decisions ($\mathbf{y}_2$). $model_1$ and $model_2$ are the pre-trained models accordingly. Then the parser reduces the "r" tagged tokens and transfers them as the children features for the next labeling stage. This process is repeated until there is no token to be reduced or the size of L equals 1. The remaining parsing process for the example sentence in Figure 1 is illustrated step by step in Figure 3 to give a more specific description of the algorithm.

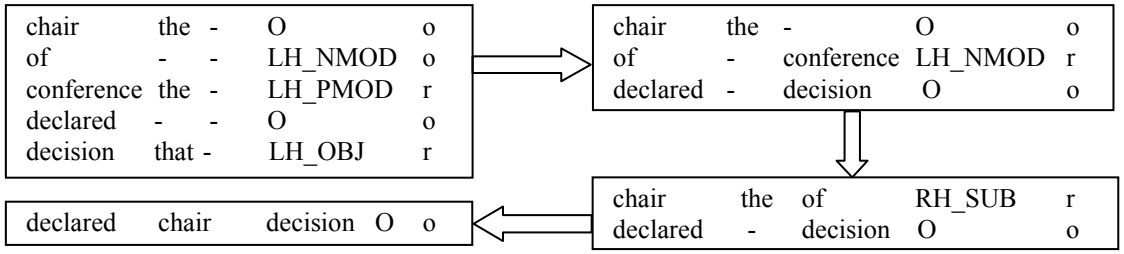

Figure 3: The parsing process following the stage showed in Figure 1(b). The second column lists the left child of the current token attached in the latest analysis and the third column is the right one.

Together with the initial labeling stage showed in Figure 1(b), the layer-based algorithm spends five iterations, i.e., five layers to get the final dependency graph of the input sentence. In each layer, the neighboring dependency relations and *reduce* decisions are traded off at different

sequence positions to obtain a globally optimal depth-one dependency sub-graph. Between the layers, the pre-built structure is handed on through the surviving tokens as well as their children. Since dependency relations only exist between two consecutive tokens, the child appearing in the observation sequence is always the leftmost or rightmost one of the parent token. Previous work based on deterministic models (Nivre and Scholz, 2004; Hall et al., 2007) has verified that the information of the children at these positions is more useful than that of others.

For training, the parsing process described above is repeated on each sentence in the training set to pick up instances on different layers.

In addition, the *reduce* examiner in the two-time labeling algorithm described above relies too much on the relation labeling results since it takes the relation labels as features. Therefore, a one-time labeling framework is introduced to be an alteration of the two-time labeling one. Figure 4 shows an example. The strings in the third column are the integrated symbols of the dependency relation labels and the *reduce* labels. Because the token whose head is not found will not be tagged with "r", a unique symbol "O" is enough to express this case.

The	DT	RH_NMOD_r
chair	NN	O
of	IN	LH_NMOD_o
the	DT	RH_NMOD_r
conference	NN	O
declared	VBD	O
that	DT	RH_NOMD_r
decision	NN	O

Figure 4: Integration of the relation and *reduce* labels

2.2 Usage of *N*-best Searching Results

The algorithm described above stops the parsing process if there is no *reduce* label "r" in the current layer. However, sometimes the fact is that the parser quits so early while the tree is not well formed yet at that point. One reason is that the *reduce* tagger is more prone to assign an "o" than an "r" due to the unbalanced training instances. Taking this into account, we use the *n*-best searching results produced by the CRF-based labeler to amend.

Taking the two-time labeling for example, although there is no "r" assigned in the current stage, the parsing process still continues if there is a relation annotated between the neighbors. The parser will ask for the next best relation label sequence ($\mathbf{y}_1$') and consequently estimate the *reduce* labels based on it. But if $\mathbf{y}_1$' is not assigned with relations, the parser will fall back on the initial best labels ($\mathbf{y}_1$) and further request the next best *reduce* labels for $\mathbf{y}_1$. In our experiments, only 2-best outputs of the labeler are utilized and the experimental results show that it works well.

2.3 Feature Design

The features used in our labelers are summarized in Table 1. Features of the tokens and children are prepared to parameterize the dependency attachment model. The relation features are added when tagging the *reduce* decisions in two-time labeling case.

As a typical sequence labeling task, the features chosen for our parser are similar to those adopted in (Sha and Pereira, 2003) for shallow parsing, and a first-order Markov dependency between labels is considered.

Cheng et al. (2006) argued that the features and the strategies for parsing in the early stage are different from parsing in the upper stages in bottom-up deterministic parsing approaches. Because the initial stage parses "words" while the upper stages parse "phrases". For this reason, we improve the proposed parser to a model-divided one in which one model is only for the first parsing layer and the other takes charge of the higher layers. The children features listed in Table 1 will not be used to parameterize the first layer model.

Table 1: Feature set for the neighboring parsing. *w* is the word and *p* is the POS tag of the token. *lc* and *rc* represent the leftmost and rightmost child, and the dependency relation type of them uses *typ*. The relation features like "RH_SUB" are denoted by *rel*. Digit bracketed marks the position of the token where the feature is sampled, negative for the left and positive for the right. "•" denotes the combination.

Tokens	*w*[-3], *w*[-2], *w*[-1], *w*[0], *w*[1], *w*[2], *w*[3] *p*[-3], *p*[-2], *p*[-1], *p*[0], *p*[1], *p*[2], *p*[3] *p*[-2] • *p*[-1], *p*[-1] • *p*[0], *p*[0] • *p*[1], *p*[1] • *p*[2], *p*[-1] • *p*[0] • *p*[1] *w*[-1] • *p*[-1], *w*[0] • *p*[0], *w*[1] • *p*[1]
Children	*w_lc*[0], *w_rc*[0] *p_lc*[-1], *p_rc*[-1], *p_lc*[0], *p_rc*[0], *p_lc*[1], *p_rc*[1] *p*[-1] • *p_lc*[-1], *p*[-1] • *p_rc*[-1], *p*[0] • *p_lc*[0], *p*[0] • *p_rc*[0], *p*[1] • *p_lc*[1], *p*[1] • *p_rc*[1] *typ_lc*[-1], *typ_rc*[-1], *typ_lc*[0], *typ_rc*[0], *typ_lc*[1], *typ_rc*[1]
Relations	*rel*[-3], *rel*[-2], *rel*[-1], *rel*[0], *rel*[1], *rel*[2], *rel*[3]

3 Experiments

To evaluate the effectiveness and efficiency of the layer-based approach, we conducted dependency parsing experiments on both English and Chinese datasets.

The English experiments were carried out on the WSJ part of Penn Treebank (Marcus et al., 1993). To match the previous work (Nivre and Scholz, 2004; Hall et al., 2006; McDonald and Pereira, 2006), we used sections 02-21 for training, section 22 for development and section 23 (about 56,684 words) for testing. The head-finding rules employed by Yamada and Matsumoto (2003) were adopted here to convert the constituent structures to dependency ones and a set of 12 dependency types was utilized as what Hall et al. (2006) did. [1] The POS tags for the development and testing set were automatically assigned by MXPOST (Ratnaparkhi, 1996). A tagging accuracy 97.05% was achieved on the testing set.

The Chinese experiments were evaluated on the Penn Chinese Treebank (CTB) version 5.0 (Xue et al., 2005). The corpus was split into training, development, and testing data as Duan et al. (2007) did to balance the different resources. 16,079 sentences were for training, 803 for development, and 1,905 (about 50,319 words) for testing. The head-finding rules and dependency type set also followed Hall et al. (2006). [2] Gold standard POS tags were used.

Eight parsers involved in our main experiments are concisely introduced as following:

MaltParser (Nivre et al., 2006): adopts transition-based model described in (Nivre, 2004). Here, MaltParser version 1.1 is employed.

Yamada03: our implementation of another typical transition-based model proposed in (Yamada and Matsumoto, 2003).

MSTParser1: The first-order paradigm of MSTParser[3] which implements the graph-based models described in (McDonald et al., 2005; McDonald and Pereira, 2006). Version 0.2 is used.

MSTParser2: The second-order paradigm of MSTParser.

Duan07: A probabilistic parsing action model proposed by Duan et al. (2007) which globally seeks the optimal action sequence above the transition-based model described in (Yamada and Matsumoto, 2003) with beam search algorithm and employs SVMs for learning.

LDParser1: One of the layer-based dependency parsers which labels the relations and *reduce* decisions at one time.

LDParser2: One of the layer-based dependency parsers which labels the relations and *reduce* decisions separately.

[1] The tree conversion and the arc labeling were implemented by Penn2Malt (http://w3.msi.vxu.se/~nivre/research/Penn2Malt.html) with the "Malt" hard-coding setting.

[2] It was realized by Penn2Malt with the head-finding rules it provided for Chinese and the hard-coding setting.

[3] http://www.seas.upenn.edu/~strctlrn/MSTParser/MSTParser.html

LDP1div: LDParser1 using divided models. In our experiments, the first layer instances in the training set are used to train the first layer model while the instances on all of the layers are trained for the higher layer model.

The first five models are taken as the baselines in the experiments and the last three ones are the proposed parsers to be compared.

The results are evaluated by the unlabeled attachment score (**UAS**), labeled attachment score (**LAS**), root accuracy (**RA**) and complete match (**CM**) according to Nivre and Scholz (2004) except that **RA** is the proportion of sentences in which the root word is correctly identified. All the metrics are calculated excluding the punctuations besides **CM**. We also present the detailed comparisons with the baselines in aspects of the computational complexity and the testing time (the CPU time). All the experiments were done on a 32-bit Intel Xeon 2.33GHz processor.

3.1 English Results

In the English experiments, all the parsers listed above except Duan07 were compared. For MaltParser, we chose the *arc-eager* algorithm (Nivre, 2004) and the feature set which got the best performance for English in (Hall et al., 2006) (the feature model Φ_5 in their work). Hall et al. (2006) reported that the SVMs learning algorithm outperformed memory-based learning (MBL) on this feature set and could parse faster. It is the same case for Chinese. Therefore, SVMs were used for both our English and Chinese experiments. We also compared the split MaltParser which utilizes the efficient *Classifiers Splitting* in the experiment where the POS tag of the next input token was selected for splitting and the split threshold was 1,000. For Yamada03, the optimal feature context window size six was chosen and the dependency relation type of the child tokens was added into the feature set. The model was also trained dividedly according to the POS tag of the left target token. For MSTParser, we tried to reproduce the results in (McDonald et al., 2005) and (McDonald and Pereira, 2006) by using the 5-best projective parsing algorithm and not including punctuations in Hamming loss calculation.

Considering the training cost, only the features that occur more than twice were modeled in LDParser1 and LDP1div. The combination of the children features and the combination between the word and POS features in Table 1 were also omitted.

The final results are compiled in Table 2. n denotes the length of the input sentence and R is the number of the dependency types appearing in the corpus. The complexity of LDParser is a constant multiple of R^2n^2 according to the labeling strategies (one or two times labeling).

Table 2: Parsing results on the English testing set

Parser	UAS (%)	LAS (%)	RA (%)	CM (%)	Complexity	Testing time
MaltParser	89.68	88.48	84.73	33.69	$O(n)$	2hour 46min
MaltParser (split)	89.52	88.19	84.81	33.77	$O(n)$	10min 20sec
Yamada03 (split)	89.59	88.72	85.11	34.15	$O(n^2)$	20min 8sec
MSTParser1	91.03	89.78	94.21	35.72	$O(n^3)$	6min 58sec
MSTParser2	91.72	90.46	94.41	39.53	$O(n^3)$	9min 44sec
LDParser2	88.60	87.34	87.96	31.13	$O(R^2n^2)$	1min 18sec
LDParser1	89.16	87.91	88.70	32.62	$O(R^2n^2)$	2min 6sec
LDP1div	89.68	88.43	89.16	33.90	$O(R^2n^2)$	1min 58sec

Among the three proposed parsers, LDParser1 outperforms LDParser2 and LDP1div is the best one. Concerning the terms for parent-prediction accuracies and sentence complete matching, the LDParsers perform similarly to the transition-based models but exceed them more in root accuracy. Thanks to the global search over the whole dependency tree the graph-based models realized by MSTParser gain the best performance among the competitors on the English dataset. However, considering the parsing efficiency, the LDParsers are quite competitive. They have

lower complexity than graph-based models and accordingly parse faster than them under the current implementations in projective case. Transition-based models can be implemented in linear time but SVMs which have been proved to achieve the highest performance in parser learning (Cheng et al., 2005; Wang et al., 2006) are not regarded as fast algorithms especially when the number of classes is large. The Classifier Splitting heuristic strategy and SVM speeding up methods (Goldberg and Elhadad, 2008) are gold choices to accelerate these implementations. However, even considering these cases, the parsing speed of the proposed LDParsers (up to 480 English words per second) is still desirable. Moreover, the speed boosting of SVMs is usually accompanied with the decrease of the accuracies or more memory consumption.

3.2 Chinese Results

We compared LDParser (LDP1div) with MaltParser, Yamada03, MSTParser and Duan07 in the Chinese experiments. *Arc-standard* algorithm (Nivre, 2004) is adopted in MaltParser because the experiments on the development set revealed that it got a higher performance than the *arc-eager* one. We also used the best Φ_5 feature set in Hall et al. (2006) for Chinese and the setting of classifier splitting was kept the same as what it was for English. So were the feature model and splitting for Yamada03. All the settings for Chinese experiments of MSTParser were not changed from English ones except the 1-best parse set size. The results on the development set indicated that the k-best (k>1) models did not surpass the 1-best one remarkably.

Only the features that appear more than once were utilized in LDP1div. Table 3 illustrates the parsing accuracies and speeds.

Table 3: Parsing results on the Chinese testing set. The complexity of Duan07 is $O(BKn^2)$, where B is the beam size of beam-search algorithm and K is the number of action steps in PAPM (Duan et al., 2007)

Parser	UAS (%)	LAS (%)	RA (%)	CM (%)	Testing time
MaltParser (split)	83.82	82.15	73.54	32.55	22min 42sec
Yamada03 (split)	83.91	82.44	70.38	31.32	27min
MSTParser1	83.39	81.75	70.76	26.30	10min 28sec
MSTParser2	85.23	83.47	75.70	31.81	15min 40sec
Duan07	84.38	82.94	71.28	32.17	9hour 57min
LDP1div	83.44	81.89	70.29	29.66	1min 53sec

The scores in Table 3 imply that LDParser is comparable to first-order MSTParser for Chinese parsing and a little weaker than transition-based approaches. The reason is that the transition-based models are more suitable for Chinese parsing than English because of the richer feature representations. This is also the reason why LDParser catches up with MSTParser on Chinese dataset. The optimal sub-graphs are delivered deterministically between the layers in LDParser which makes the parser be able to use the dependency graph pre-built. Duan07 which added global search to Yamada03 obtains further better performance.[4]

Similar to the experiments for English，LDParser spends the shortest time. It parses Chinese sentences about 450 words per second. Moreover, the gaps between the speeds of LDParser and others' consistently increase. For example, LDP1div is about 8 times faster than MSTParser2 and 15 times faster than split MaltParser while it was both 5 times faster in the English experiments. We think it is partially due to the character encoding mechanism in the Java implementation of MaltParser and MSTParser. Another reason is that the average sentence length of the Chinese testing set is 26.4 words, which is longer than that of English (23.5). Profiting from the layer-based search and sequence typed analysis, LDParser handles long

[4] The rank of the parsers under the metrics of parsing accuracies in Table 3 is not quite the same as what was in Duan et al. (2007). It is because the dependency structures of the data were differently converted in our experiments.

sentences more efficiently. The global search of the transitions adopted in Duan07 makes the parser the most laggard one.

3.3 Additional Results

To further study the character of the layer-based parser, we present two additional results in this section. Table 4 illustrates the unlabeled attachment scores (UAS) of different dependency lengths in the English parsing experiment. The dependencies are calculated separately according to their length, equal to 1 (the neighboring relations), shorter than 3 or longer than 3. The threshold is chosen in terms of the average dependency length of the corpus which is 3.28.

Table 4: Unlabeled attachment scores of different dependency lengths on the English dataset

Parser	=1	≤ 3	> 3
MaltParser	94.24	93.09	73.92
MSTParser2	94.67	93.59	83.23
LDP1div	94.56	93.07	74.53

The moderate behavior of LDParser in neighboring attachment accuracy demonstrates that the globally optimal sequence labeling is competent for neighboring relation parsing compared with the tree-typed hierarchical ones. It even exceeds the transition-based parser. For the long dependencies, LDParser also does well than the transition-based one which verifies that the global search inside the parsing layer lightens the error propagation in transition-based models.

By keeping partial parsing history through factoring over adjacent edge pairs of the dependency tree, the second-order MSTParser performs the best both for short and long dependencies. Making use of the pre-built structures, LDParser achieved a similar performance as MSTParser for short dependencies but gets worse for long ones. It is because LDParser is still a deterministic model in nature, the error propagation is unavoidable when the dependencies grow long. Another reason is that the higher layers are not modeled separately from each other in the current LDParser and it depresses the disambiguation ability of the model for higher layer parsing.

We further examined the behaviors of the parsers on long sentences. 171 sentences with more than 40 words in the English testing set were tested and the results are listed in Table 5. The percentages represent the decrease of the speed when parsing the long sentences. Taking both the dependency accuracy and root accuracy into account, LDParser is almost the same as MaltParser. Although MSTParser is still the best, the parsing speed has dropped a lot (57%) when sentences grow long. Contrarily, there is only 8% slower for LDParser to parse these sentences which further implies that the layer-based approach is not sensitive to the length of the sentences and can be more efficient for long sentences than other parsers compared.

Table 5: Results for sentences longer than 40 words. 8,019 words were analyzed in the experiment.

Parser	UAS	RA	Testing time
MaltParser (split)	87.34	71.92	1min 45sec (17%)
MSTParser2	89.84	94.74	3min 11sec (57%)
LDP1div	86.58	78.95	18sec (8%)

4 Related Work

Actually, as a bottom-up framework the proposed approach is a little similar to the model proposed by Yamada and Matsumoto (2003) which employed a shift-reduce algorithm with multiple passes over the input. The transitions in this model are greedily selected at each parser state, i.e., configuration, from the left to the right during the parsing pass. To remove the greedy properties in the transition-based models, Johansson and Nugues (2007) and Duan et al. (2007) added a global search over the transition sequences. Our approach also uses multiple passes

(layers) to form the dependency tree, and integrates global search like Johansson and Nugues (2007) and Duan et al. (2007) did to find the optimal combination of dependency relations. However, they scored all the graph space as what graph-based models do while we focalized it in a parsing layer for the sake of efficiency. In addition, they inherited the hierarchical analyzing mechanism used in transition-based models but our parser introduced the sequence labeling technique.

Some existing work tried to combine the graph-based and transition-based models for dependency parsing. Sagae and Lavie (2006) built a graph-based model to reparse a dependency graph of which the arc scores were created by the outputs of three transition-based parsers. Hall et al. (2007) followed Sagae's methodology and blended six transition-based parsers. This kind of combinations can be seen as a structural voting of the graph-based and transition-based models. A more effective integration was developed by Nivre and McDonald (2008) who treated the outputs of one model as features for the other. However, all these combination approaches just made use of the outputs of the component parsers without modifying their structures or parsing algorithms. It is quite different from ours. Our parsing framework inherits the benefits of the graph-based and transition-based models with both new structure and algorithm.

Cheng et al. (2006) and Wu et al. (2007) used neighboring dependency attachment taggers to improve the performance of the deterministic parser. In Cheng's method, neighboring relations were decided by greedy sequential SVM-based classifiers and the tagging results were delivered directly to continue the subsequent parsing. Wu et al. (2007) adopted CRFs as the dependency learner and accepted the results of the neighboring parsing as features to increase the original feature set. In their parsers, the neighboring relation tagger was just a preprocessor, while ours works throughout the whole parsing process. We have proved that it can also work well just relying on the neighboring relation taggers for parsing.

5 Conclusion

In this paper, we proposed a novel bottom-up layer-based dependency parsing approach. It takes a dependency layer as the parsing unit and works as a discriminative "graph-based" parser inside the layer while a deterministic "transition-based" parser between the layers. The CRF-based sequence labeling algorithm is adopted entirely to build the dependency structures. The experimental results confirm the effectiveness and efficiency of the proposed parser and they also proved that the sequence labeling techniques can be good substitutes to tree-typed ones for hierarchical structure analysis tasks.

Efforts are going to be made to improve the parsing accuracy of the proposed parser. More sophisticated labeling models for the higher layers and finer feature combinations are in investigating. As only consecutive words are considered, nonprojective case is not yet well dealt with in our approach. It is left to be another future work.

References

Cheng, Y., M. Asahara and Y. Matsumoto. 2005. Machine learning-based dependency analyzer for Chinese. *Proceedings of the International Conference on Chinese Computing*, pp. 66-73.

Cheng, Y., M. Asahara and Y Matsumoto. 2006. Multi-lingual dependency parsing at NAIST. *Proceedings of the Conference on Computational Natural Language Learning,* pp. 191-195.

Duan, X., J. Zhao and B. Xu. 2007. Probabilistic models for action-based Chinese dependency parsing. *Proceedings of the European Conference on Machine Learning and theEuropean Conference on Principles and Practice of Knowledge Discovery in Databases,* pp. 559-566.

Goldberg, Y. and M. Elhadad. 2008. SplitSVM: fast, space-efficient, non-heuristic, polynomial kernel computation for NLP applications. *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pp. 237-240.

Hall, J., J. Nivre and J. Nilsson. 2006. Discriminative classifiers for deterministic dependency parsing. *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics,* pp. 316-323.

Hall, J., J. Nilsson, J. Nivre, G. Eryiğit, B. Megyesi, M. Nilsson and M. Saers. 2007. Single Malt or blended? A study in multilingual parser optimization. *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning,* pp. 933-939.

Johansson, R. and P. Nugues. 2007. Incremental dependency parsing using online learning. *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pp.1134-1138.

Kudo, T. and Y.Matsumoto. 2001. Chunking with support vector machines. *Proceedings of the North America Chapter of the Association for Computational Linguistics*, pp. 192-199.

Marcus, M. P., B. Santorini and M. A. Marcinkiewicz. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2): 313-330.

McDonald, R., K. Crammer and F. Pereira. 2005. Online large-margin training of dependency parsers. *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pp. 91-98.

McDonald, R. and F. Pereira. 2006. Online learning of approximate dependency parsing algorithms. *Proceedings of the European Chapter of the Association for Computational Linguistics*, pp. 81-88.

McDonald, R. and J. Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pp. 122-131.

Nivre, J. 2004. Incrementality in deterministic dependency parsing. *Proceedings of the Annual Meeting of the Association for Computational Linguistics,* pp.50-57.

Nivre, J. and M. Scholz. 2004. Deterministic dependency parsing of English text. *Proceedings of the International Conference on Computational Linguistics,* pp. 64-70.

Nivre, J., J. Hall and J. Nilsson. 2006. MaltParser: a data-driven parser-generator for dependency parsing. *Proceedings of the International Conference on Language Resources and Evaluations*, pp. 2216-2219.

Nivre, J. and R. McDonald. 2008. Integrating graph-based and transition-based dependency parsers. *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pp. 950-958.

Ratnaparkhi, A. 1996. A maximum entropy model for part-of-speech tagging. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 133-142.

Sagae, K. and A. Lavie. 2006. Parser combination by reparsing. *Proceedings of the North America Chapter of the Association for Computational Linguistics*, pp. 129-132.

Sha, F. and F. Pereira. 2003. Shallow parsing with conditional random fields. *Proceedings of the North America Chapter of the Association for Computational Linguistics*, pp. 213-220.

Wang, M., K. Sagae and T. Mitamura. 2006. A fast, accurate deterministic parser for Chinese. *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pp.425-432.

Wu, Y., J. Yang and Y. Lee. 2007. Multilingual deterministic dependency parsing framework using modified finite Newton method support vector machines. *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pp. 1175-1181.

Xue, N., F. Xia, F. Chiou and M. Palmer. 2005. The Penn Chinese Treebank: phrase structure annotation of a large corpus, *Natural Language Engineering*, *Cambridge University Press*, 11(2): 207-238.

Yamada, H. and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. *Proceedings of the International Conference on Parsing Technologies*, pp.195-206.