Two-Phase Shift-Reduce Deterministic Dependency Parser of Chinese

Meixun Jin, Mi-Young Kim and Jong-Hyeok Lee
Div. of Electrical and Computer Engineering,
Pohang University of Science and Technology (POSTECH)
Advanced Information Technology Research Center (AITrc)
{Meixunj, colorful, jhlee}@postech.ac.kr

Abstract

In the Chinese language, a verb may have its dependents on its left, right or on both sides. The ambiguity resolution of right-side dependencies is essential for dependency parsing of sentences with two or more verbs. Previous works on shift-reduce dependency parsers may not guarantee the connectivity of a dependency tree due to their weakness at resolving the right-side dependencies. This paper proposes a two-phase shift-reduce dependency parser based on SVM learning. The left-side dependents and right-side nominal dependents are detected in Phase I, and right-side verbal dependents are decided in Phase II. In experimental evaluation, our proposed method outperforms previous shift-reduce dependency parsers for the Chine language, showing improvement of dependency accuracy by 10.08%.

1 Introduction

Dependency parsing describes syntactic structure of a sentence in terms of links between individual words rather than constituency trees. The fundamental relation in dependency parsing is between head and dependent. Robinson[1] formulates four axioms to the well-formed dependency structures, known as *single headed, acyclic, connective and projective*.

In this paper, we present a dependency parsing strategy that produces one dependency structure that satisfies all these constraints.

This paper is organized as follows. Related works are introduced in section 2. In section 3, detailed analysis of the work of Nivre[2] and Yamada[3] are given. Then our parsing strategy is introduced. In section 4, experiments and results are delivered. Finally a conclusion will be given in section 5.

2 Overview of Related Works

Most nature language grammars tend to assign many possible syntactic structures to the same input utterance. A parser should output a single analysis for each sentence. The task of selecting one single analysis for a given sentence is known as disambiguation.

Some of the parsing strategies first produce all possible trees for a sentence. The disambiguation work is done in the end by searching the most probable one through parsing tree forest. Statistical parsers employ probability as a disambiguation measure and output the tree with the highest probability[4,5]. However, in the work of Collins [6], 42% of the correct parse trees were not in the candidate pool of ~30-best parses. Disambiguation work by searching throughout the parsing tree forest has limitations.

The alternative way is to disambiguate at each parsing step and output the parsing result deterministically. Nivre[2] and Yamada[3] suggest a shift-reduce like dependency parsing strategy. In section 3.1 we give a detailed analysis of their approach.

There are several approaches for dependency parsing on Chinese text. Ma[5] and Cheng[18] are examples of these approaches. The training and test set Ma[5] used, are not sufficient to prove the reliability of Ma's[5] approach. On the frame of parsing Chinese with CFG, there are several approaches to apply the original English parsing strategies to Chinese [7,8,9]. The potential purposes of these works are to take advantage of state-of-art English parsing strategy and to find a way to apply it to Chinese text. Due to the differences between Chinese and English, the performance of the system on Chinese is about 10% lower comparing the performance of the original system.

3 Two-Phase Dependency Parsing

3.1 Review of Previous Shift-Reduce Dependency Parsers

Nivre[3] presented a shift-reduce dependency parsing algorithm which can parse in linear time. The Nivre's parser was represented by a triples <*S, I, A*>, where *S* is a stack, *I* is a list of (remaining) input tokens, and *A* is the set of determined dependency relations. Nivre defined four transitions: *Left-Arc*, *Right-Arc*, *Reduce,* and *Shift*. If there is a dependency relation between the top word of the stack and the input word, according to the direction of the dependency arc, it can be either *Left-Arc* or *Right-Arc*. Otherwise, the transition can be either *shift* or *reduce*. If the head of the top word of the stack is already determined, then the transition is *reduce,* otherwise *shift*. The action of each transition is shown in **Fig.1**. For details, please refer to Nivre[3,10]. **Fig.2** gives an example[1] of parsing a Chinese sentence using Nivre's algorithm.

Nivre's[3,10] approach has several advantages. First, the dependency structure produced by the algorithm is projective and acyclic[3]. Second, the algorithm performs very well for deciding short-distance dependences. Third, at each parsing step, all of the dependency relations on the left side of the input word are determined. Also as the author emphasizes, the time complexity is linear.

However, wrong decision of *reduce* transition, like early *reduce,* cause the word at the top of the stack loses the chance to be the head of others. Some words lose the chance to be the head of other following words. As a result, the dependents of this word will have a wrong head or may have no head.

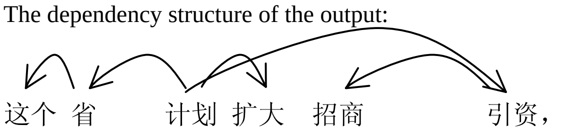
The parsing steps of a Chinese sentence using Nivre's[3] algorithm are given in **Fig.2.** At step-5 of **Fig.2,** after *reduce*, the top of the stack was popped. The algorithm doesn't give a chance for the word 扩大 to be the head of other words. Therefore, word '引资' cannot have word '扩大' as its head. In the final dependency tree of example-1 in **Fig.2**, the arc from 计划 to 引资 is wrong. **Fig.3** gives the correct dependency tree. Here, 扩大 is the head of word 引资.

```
If there is a dependency relation between top.stack and input
    If the dependency relation is Left_arc
      Insert (input, top.stack) pair into set A      ]  ⟹  Left_arc
      pop(stack);
    Else
      Insert (top.stack, input) pair into set A      ]  ⟹  Right_arc
      push(input);
Else
    If the head of top of the stack is determined    ]  ⟹  Reduce
      pop(stack);
    Else
      push(input);                                   ]  ⟹  shift
```

Fig. 1. Transitions defined by Nivre[3]

```
这个 省      计划 扩大  招商          引资,
This province plan extend attract merchants attract investments.
The province plans to expand attracting merchants and investments.
                stack , input             relation set A
                  ⇩       ⇩                     ⇩
Step-0:     <nil,这个省计划扩大招商引资,{}>
Step-1:→S   <这个,省计划扩大招商引资,{}>
Step-2:→LA <省,计划扩大招商引资,{(省,这个)}>
Step-3:→LA <计划,扩大招商引资,{(省,这个),(计划,省)}>
Step-4:→RA <扩大 计划,招商引资,{(省,这个),(计划,省),
                                  (计划, 扩大)}>
Step-5:→R <计划,招商引资,{(省,这个),(计划,省),
                             (计划,扩大)}>
Step-6:→S <招商 计划,引资,{(省,这个),(计划,省),
                             (计划,扩大)}>
Step-7:→LA <计划,引资,{(省,这个),(计划,省),
                         (计划,扩大),(引资,招商)}>
Step-8:→LA <计划,nil,{(省,这个),(计划,省),
                 (计划,扩大),(引资,招商),(计划, 引资)}>
```

The dependency structure of the output:



S:Shift LA:Left-arc RA:Right-arc R:reduce

Fig. 2. Example-1: Parsing using Nivre's algorithm

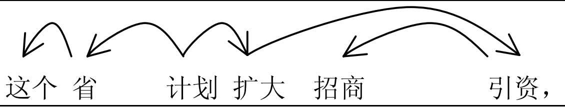


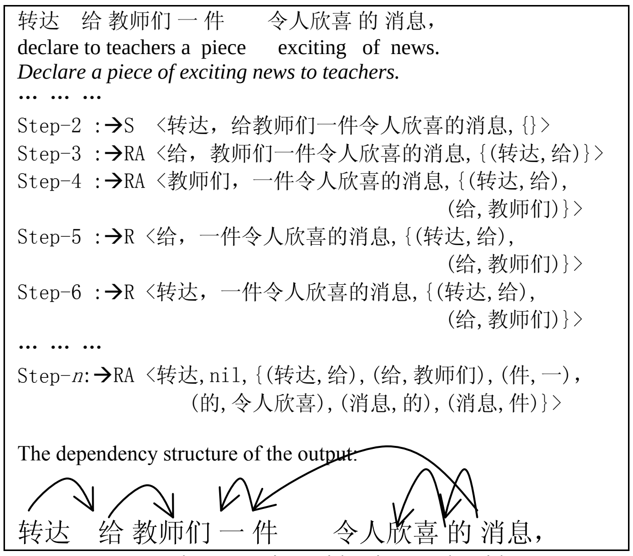
Fig. 3. The correct parse tree of Example-1

Fig.4. gives the parsing step of another example. As the final dependency tree in **Fig.4** shows, there is no head for word 消息。After Step-5, the top of the stack is word 给 and input word is 一. There is no dependency relation between these two words. Since the head of the word 给 is already determined in step-2, the next transition is R(educe). As a result, word 给 loses the chance to be the head of word 消息. So, there is no head assigned to word 消息 in **Fig.4**. Therefore, Nivre's algorithm causes some errors for determining the right-side dependents.

Yamada's[4] approach is similar to Nivre's[3].

[1] All the example sentences are from CTB.

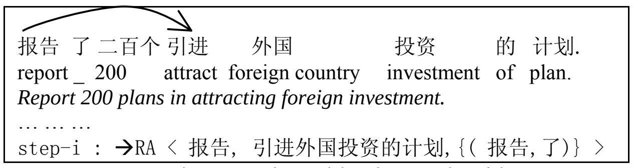
Yamada’s algorithm define three actions: left, right and shift, which were similar to those of Nivre’s. Yamada parsed a sentence by scanning the sentence word by word from left to right, during the meantime, left or right or shift actions were decided. For short dependents, Yamada’s algorithm can cope with it easily. For long dependents, Yamada tried to solve by increasing the iteration of scanning the sentences. As Yamada pointed out, ‘shift’ transition was executed for two kinds of structure. This may cause wrong decision while deciding the action of transition. Yamada tried to resolve it by looking ahead for more information on the right side of the target word.



```
转达   给 教师们 一 件       令人欣喜 的 消息，
declare to teachers a  piece     exciting  of  news.
Declare a piece of exciting news to teachers.
… … …
Step-2 :→S  <转达，给教师们一件令人欣喜的消息,{}>
Step-3 :→RA <给，教师们一件令人欣喜的消息,{(转达,给)}>
Step-4 :→RA <教师们，一件令人欣喜的消息,{(转达,给),
                                  (给,教师们)}>
Step-5 :→R  <给，一件令人欣喜的消息,{(转达,给),
                                  (给,教师们)}>
Step-6 :→R  <转达，一件令人欣喜的消息,{(转达,给),
                                  (给,教师们)}>
… … …
Step-n:→RA <转达,nil,{(转达,给),(给,教师们),(件,一),
                (的,令人欣喜),(消息,的),(消息,件)}>

The dependency structure of the output:
```

Fig. 4. Example-2: Parsing with Nivre’s algorithm



```
报告 了 二百个 引进      外国              投资        的  计划.
report _ 200    attract  foreign country   investment  of  plan.
Report 200 plans in attracting foreign investment.
… … …
step-i : →RA < 报告, 引进外国投资的计划,{( 报告,了)} >
```

Fig. 5. Example-3: Parsing with Nivre’s algorithm

When applying to Chinese parsing, the determination of dependency relation between two verbs is not effective. In the example-3 of **Fig.5**, at step-i, the parser decides whether the dependency relation between 报告 and 引进 is either *Left-arc* or *Right-arc*. The actual head of the verb 引进 is 的, which is distant. By looking only two or three right side words ahead, to decide the dependency relation between these verbs at this moment is not reliable. Yamada’s algorithm is not a clear solution to determine the right side dependents either.

3.2 Two-Phase Dependency Parsing

For the head-final languages like Korean or Japanese, Nivre[3] and Yamada’s[4] approaches are efficient. However, being applied to Chinese text, the existing methods cannot correctly detect various kinds of right-side dependents involved in verbs. All wrong decisions of *reduce* transition mainly occur if the right dependent of a verb is also a verb, which may have right-side dependents.

For the correct detection of the right-side dependents, we divide the parsing procedure into two-phase. Phase I is to detect the left-side dependents and right-side nominal dependents. Although some nominal dependents are right-side, they don’t have dependents on the right side, and will not cause any ambiguities related to right-side dependents. In Phase II, the detection of right-side verbal dependents, are performed.

3.2.1 Phase I

In Phase I, we determine the left-side dependents and right-side nominal dependents. We define three transitions for Phase I: *Shift, Left-Arc, Right-Arc*. The actions of transition *shift* and *Left-Arc* are the same as Nivre[3] defines. However, in our method, the transition of *Right-Arc* does not push the input token to the stack. The original purpose for pushing input to stack after *right-arc*, is to give a chance for the input to be a potential head of the following words. In Chinese, only verbs and prepositions have right-side dependents. For other POS categories, the action of pushing into stack is nonsense. In case that the input word is a preposition, there is no ambiguities we describe. Only the words belong to various verbal categories may cause problems. The method that we use is as follows. When the top word of the stack and the next input word are **verbs**, like VV, VE, VC or VA[2] [11], the detection of the dependency relation between these two verbs is delayed by transition of *shift*. To differentiate this *shift* from original *shift*, we call this *verbal-shift*. The determination of the dependency relation between these two verbs will be postponed until phase II. The transitions are summarized as **Fig.6.**

If there is no more input word, phase I terminates. The output of the phase I is a stack, which

[2] VV, VE, VC and VA are Penn Chinese Treebank POS categories related to verbs. For details, please refer to [11].

contains verbs in reverse order of the original appearance of the verbs in the sentence. Each verb in the stack may have their partial dependents, which are determined in Phase I.

```
If the action is Verbal-shift
        : push the input to the stack
else if the action is Shift
        push the input to the stack
else if the action is Left-arc
        set the dependency relation for two words; pop the top of the stack
else if the action is Right-arc
        set the dependency relation for two words
```

Fig. 6. Types of transitions in the phase I

The type of transition is determined by the top word of the stack, input word and their context. Most of the previous parsing models[4,12,13] use lexical words as features. Compared to Penn English Treebank, the size of Penn Chinese Treebank (version 4.0, abbreviated as CTB) is rather small. Considering the data sparseness problem, we use POS tags instead of lexical words itself. As **Fig.7.** shows, the window for feature extraction is the **top word of the stack, input word, previous word of the top of the stack, next word of the input.** The left-side nearest dependent of these is also taken into consideration. Besides, we use two more features, if_adjoin, and Punc. The feature vector for Phase I is shown in **Fig.7.**

3.2.2 Phase II

After Phase I, only verbs remain in the stack. In Phase II, we determine the right-side verbal dependents. We take the output stack of Phase I as input. Some words in the stack will have right-side dependents as shown in **Fig.8.** For Phase II, we also define three transitions: *shift, left-arc, right-arc*. The operations of these three transitions are the same as Phase I, but there are no *verbal-shifts*. **Fig.9** shows the output of Phase I and parsing at Phase II of example given in **Fig.8**.

The window for feature extraction is the same as that of Phase I. The right-side nearest dependent is newly taken as features for Phase II. The feature vector for Phase II is shown in **Fig.10.**

The two-phase parsing will output a projective, acyclic and connective dependency structure. Nivre[10] said that the time complexity of his parser is 2 times the size of the sentence. Our algorithm is 4 times the size of the sentence, so the time complexity of our parser is still linear to the size of the sentence.

Windows for feature extraction :
t.stack : top word of the stack
p.stack: previous word of top of the stack
input : input word
n.input: next word of the input word

x.pos : POS tag of word x
x.left.child : the left-side nearest dependent of word x

punc : the surface form of punctuation between top word of the stack and input word, if there is any
if_adjoin : a binary indicator to show if the top word of the stack and input word are adjoined

The feature vector for Phase I is :
<p.stack.pos t.stack.pos input.pos n.input.pos p.stack.left.child.pos t.stack.left.child.pos input.left.child.pos punc if_adjoin>

Fig. 7. Feature vector for Phase I

这位官员说，四川将奉行更加开放的政策，不断改善投资环境，引进更多的海外资金、先进的技术和管理经验。
(The official said that Sichuan will pursue a more open door policy, continuously improve the investment environments and attract more capitals from overseas, advanced techniques and experiences of administration.)

The contents of stack after Phase I: 〈引进，改善，奉行，说〉.
(attract, improve, pursue, said)

The dependents of each verb in the stack

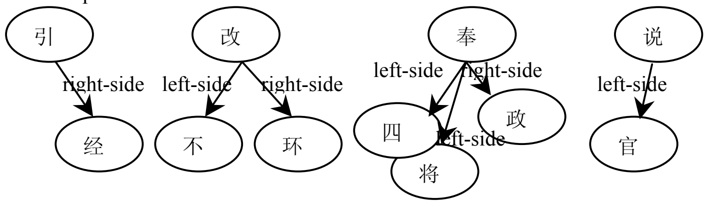


Fig. 8. Dependents of each verb after Phase I

```
step-0      <nil, 引进 改善 奉行 说, {}>
step-1→S    < 引进, 改善 奉行 说, {}>
step-2→RA  < 引进, 奉行 说, {(引进,改善)}>
step-3→RA  < 引进, 说, {(引进,改善),(引进,奉行)}>
step-4→LA  < nil, 说, {(引进,改善),(引进,奉行),
                                          (说,引进)}>
step-5 →S  < 说, nil, {(引进,改善),(引进,奉行),
                                          (说,引进)}>
```

Fig. 9. Example of parsing at Phase II

The feature vector for Phase II is :
<p.stack.pos t.stack.pos input.pos n.input.pos p.stack.left.child.pos t.stack.left.child.pos input.left.child.pos p.stack.right.child.pos t.stack.right.child.pos input.right.child.pos n.input.right.child.pos punc if_adjoin>

Fig. 10. Feature vector for Phase II.

4 Experiments and Evaluation

Our parsing procedure is sequentially performed from left to right. The feature vectors for

Phase I and Phase II are used as the input for the parsing model. The model outputs a parsing action, *left-arc, right-arc or shift*. We use SVM as the model to obtain a parsing action, and use CTB for training and test the model.

4.1 Conversion of Penn Chinese Treebank to Dependency Trees

Annotating a Treebank is a tedious task. To take the advantage of CTB, we made some heuristic rules to convert CTB into dependency Treebank. This kind of conversion task has been done on English Treebank[14,10,4]. We use the dependency formalism as Zhou[15] defined.

CTB contains 15,162 newswire sentences (including titles, fragments and headlines). The contents of CTB are from Xinhua of mainland, information services department of HKSAR and Sinorama magazine of Taiwan. For experiments, 12,142 sentences are extracted, excluding all the titles, headlines and fragments.

For the conversion task, we made some heuristic rules. CTB defines total 23 syntactic phrases and verb compounds[11]. A phrase is composed of several words accompanied to a head word. The head word of each phrase is used as an important resource for PCFG parsing[12,13]. According to the position of the head word with respect to other words, a phrase[3] can be categorized into head-final, head-initial or head-middle set. **Table.1** shows the head-initial, head-final and head-middle groups.

For VP, IP and CP, these phrases have a verb as its head word. So we find a main verb and regard the verb the head word of the phrase. If the head word for each phrase is determined, other words composing the phrase simply take the head word of the phrase as its head. In the case of BA/LB[4], we take a different view from what is done in CTB. Zhou[15] regards BA/LB as the dependent of the following verb. We follow Zhou's[15] thought. For sentences containing BA/LB, we converted them into dependency trees manually. With above heuristics, we converted the original CTB into dependency Treebank.

[3] We use the label of phrases as CTB has defined. We exclude FRAG, LST, PRN. For each definition of the phrase please refer to [11].
[4] BA, LB are two POS categories of CTB. For details, see [11].

4.2 Experiments

SVM is one of the binary classifiers based on maximum margin strategy introduced by Vapnik[16]. SVM has been used for various NLP tasks, and gives reasonable outputs. For the experiments reported in this paper, we used the software package SVM^{light} [17].

For evaluation matrix, we use **Dependency Accuracy** and **Root Accuracy** defined by Yamada[4]. An additional evaluation measure, **None Head** is defined as following.

```
None Head: the proportion of words whose
head is not determined.
```

GROUP	PHRASES
Head-initial	PP; VRD; VPT;
Head-final	ADJP; ADVP; CLP; DNP; DVP; DP; LCP; NP; QP; VCD; VCP; UCP; VSB; VNV;
Head-middle	CP; IP; VP;

Table 1. Cluster of CTB syntactic phrases

	Dependency accuracy	*Root accuracy*	*None head*
Nivre's algorithm[10]	73.34%	69.98%	1.53%
Ours	84.42%	83.33%	----

Table 2. Comparison of dependency accuracy with Nivre's

We construct two SVM binary classifiers**,** **Dep vs. N_Dep** and **LA vs. RA**, to output the transition action of *Left-arc, Right-arc or Shift*. **Dep vs. N_Dep** classifier determines if two words have a dependency relation. If two words have no dependency relation, the transition action is simply *Shift*. If there is a dependency relation, the second classifier will decide the direction of it, and the transition action is either *Left-arc* or *Right-arc*.

We first train a model along the algorithm of Nivre[10]. The training and test sentences are randomly selected. **Table.2** shows that 1.53% of the words cannot find their head after parsing. This result means that the original Nivre's algorithm cannot guarantee a connective dependency structure.

With our two-phase parsing algorithm, there is no *none head*. Then, the *dependency accuracy* and *root accuracy* are increased by 10.08% and 13.35% respectively.

4.3 Comparison with Related Works

Compared to the original works of Nivre[10] and Yamada[4], the performance of our system is lower. We think that is because the target language is different.

	Average sentence length	*Dependency accuracy*	*Root accuracy*
Ma[5]	9	80.25%	83.22%
Cheng[18]	5.27	94.44%	--
Ours	34	84.42%	83.33%

Table 3 Comparison of the parsing performances between Ma[5], Cheng[18] and ours

Table 3 gives the comparison of the performances between Ma[5], Chen[18] and ours. The training and test domain of Ma[5] is not clear. Cheng[18] used CKIP corpus in his experiments. The average length of sentence in our test set is 34, which is much longer than that in Ma[5] and Cheng[18]. The performance of our system is still better than Ma[5] and less than Cheng[8].

5 Conclusion

To resolve the right-side long distance dependencies, we propose two-phase shift-reduce parsing strategy. The parsing strategy not only guarantees the connectivity of dependency tree, but also improves the parsing performance. As the length of sentences increases, the ambiguities for parsing increase drastically. With our two-phase shift-reduce parsing strategy, the performance of syntactic parsing of long sentences is also reasonable.

The motivation of this paper is to design a well-formed dependency parser for Chinese. We believe that there're rooms to improve the performance. We plan to work further to explore the optimal features. We also plan to parse English text with our algorithm to see if it can compete with the state-of-art dependency parsers on English. We believe that our parsing strategy can apply to other languages, in which head position is mixed, as Chinese language. We think that it is the main contribution of our approach.

References

1. Robinson, J.J.: Dependency structures and transformation rules. Language 46 (1970) 259-285
2. Nivre, J.: An efficient algorithm for projective dependency parsing. In Proceedings of IWPT (2003) 149-160
3. Yamada, H. and Matsumoto, Y.: Statistical dependency analysis with support vector machines. In Proceedings of IWPT (2003) 195-206
4. Eisner, J.M.:Three new probabilistic models for dependency parsing: An exploration. In Proceedings of ACL.(1996) 340-345
5. Ma,J., Zhang,Y. and Li,S.: A statistical dependency parser of Chinese under small training data. IJCNLP-04 Workshop : Beyond Shallow Analyese-Formalisms and Statistical Modeling for Deep Analyses (2004)
6. Collins,M.: Discriminative reranking for natural language parsing. In proceedings of ICML 17.(2000) 175-182
7. Fung,P., Ngai,G, Yang,Y.S and Chen,B.: A maximum-entropy Chinese parser augmented by transformation-based learning. ACM transactions on Asian language information processing. Volume 3. Number 2.(2004) 159-168
8. Levy,R. and Manning,C.: Is it harder to parse Chinese, or the Chinese Treebank? In Proceedings of ACL. (2003) 439-446
9. Bikel, D.M. and.Chiang, D.: Two Statistical Parsing models applied to the Chinese Treebank. In proceedings of the second Chinese language processing workshop. (2000)
10. Nivre,J, Hall,J and Nilsson,J.: Deterministic dependency parsing of English text. In Proceedings of COLING. (2004) 23–27
11. Xue,N and Xia,F.: The bracketing guidelines for the Penn Chinese Treebank(3.0). IRCS Report 00-08, University of Pennsylvania (2000)
12. Collins,M.: Three generative lexicalised models for statistical parsing. In Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics, Madrid (1997) 16-23
13. Charniak,E.: A maximum-entropy-inspired parser. In Proceedings of NAACL. Seattle (2000) 132–139,
14. Collins,M.: A new statistical parser based on bigram lexical dependencies. In Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics, philladelphia (1996) 184–191
15. Zhou,M. and Huang,C.: Approach to the Chinese dependency formalism for the tagging of corpus. Journal of Chinese information processing.(in Chinese), Vol. 8(3) (1994) 35-52
16. Joachims,T.: Making large-scale SVM learning practical. Advances in Kernel Methods-Support Vector Learning, B.Scholkopf and C.Burges and A.Smola(Eds.), MIT-Press (1999)
17. Vapnik, V.N.: The nature of statistical learning theory. Springer, New York. (1995)
18. Cheng, Y.C, Asahara,M and Matsumoto Y.: Deterministic dependency structure analyzer for Chinese. In proceedings of the first IJCNLP(2004) 135-140