*Journal of Chinese Language and Computing 15 (1)*: (13-24)

# Machine Learning-based Dependency Analyzer for Chinese

Yuchang CHENG and Masayuki ASAHARA and Yuji MATSUMOTO
*Graduate School of Information Science*
*Nara Institute of Science and Technology,*
*8916-5 Takayama, Ikoma, Nara 630-0192, Japan*
{yuchan-c, masayu-a, matsu}@is.naist.jp

**Abstract**

*In this paper, we present a deterministic dependency structure analyzer for Chinese. This analyzer implements two algorithms – Yamada and Nivre algorithms – and two sorts of classifiers – Support Vector Machines and Maximum Entropy models. We compare the performance of these 2x2 combinations. We evaluate the methods on a dependency tagged corpus derived from the CKIP Treebank corpus. Then, we analyze the errors in the experiments and found that some errors are caused by mistakes of nominal compound analysis. Therefore we adopt a noun phrase chunker to overcome this problem.*

**Keywords**

*Natural Language Processing, Chinese, Parsing, Parsing algorithm, Dependency Analysis, Machine Learning*

## 1. Introduction

Many syntactic analyzers for English have been implemented and have demonstrated good performance (Charniak, 2001; Collins, 2004; Ratnaparkhi, 1999). However, implementation of Chinese syntactic structure analyzers is still limited, since the structure of the Chinese language is quite different from other languages. We are currently developing a Chinese syntactic structure analyzer. In this paper, we present our machine learning-based syntactic structure analyzer for Chinese.

Our analyzer produces word dependency structures rather than phrase structures. The reason is that dependency structures are simpler and more comprehensible than phrase structures. Moreover, construction of a word dependency annotated corpus is easier than construction of a phrase structure annotated corpus so as to be used as training data. Consistency among annotators can also be more easily achieved using the simpler dependency structure.

We utilize a deterministic method for dependency relation construction. Deterministic methods of dependency structure analysis are proposed for Japanese (Kudo, 2002), for English (Yamada, 2003; Nivre, 2004a) and for Norwegian (Nivre, 2004b). First, we adopt Yamada's method to implement Chinese dependency analysis. However, Yamada's method

still has a crucial ambiguity in estimating parsing actions. To resolve the problem, we implement another bottom-up algorithm proposed by (Nivre, 2004a). Both in Yamada's and Nivre's algorithms, the dependency relations are constructed by a bottom-up schema with machine learners. While Nivre's algorithm is implemented with memory-based learning, Yamada's algorithm is implemented with Support Vector Machines. We use Support Vector Machines (hereafter SVMs) and Maximum Entropy (hereafter MaxEnt) models into their algorithms.

There are many nominal compounds in Chinese. Some nominal compounds include more than three words. Although such long nominal compounds are few, they can influence the performance of our analyzer, since such long nominal compounds connot be analyzed well in the same framework of general parsing. This becomes an error source in our experiment. Therefore, we tried to solve this problem using a noun phrase (hereafter NP) chunker. We adopt an NP-chunker based on SVMs to extract nominal compounds from the input sentence prior to dependency analysis.

We compare the accuracy of two algorithms. The aim is to implement a variation of the deterministic methods and to show how the methods are applicable to Chinese dependency analysis. Our analyzers are experimented on the CKIP Chinese Treebank (K-J Chen et al, 1999), which is a phrase structured and head annotated corpus. Phrase structures are converted into word dependency structures according to the head information. We perform experimental evaluation in several settings on the converted corpus.

In the next section, we describe two deterministic dependency structure analysis algorithms. Section 3 reports experimental evaluation and comparison with related work. Section 4 discusses the effect of the NP-chunker in the reduction of errors. Section 5 shows our future direction. Finally, we summarize our findings and conclude.

## 2. Parsing Models

This section presents two parsing algorithms proposed by (Yamada, 2003) and (Nivre, 2004a) and the machine learning methods which are used in our analyzer. Both algorithms are deterministic approaches, in which the dependency relations are constructed by bottom-up deterministic schemata. The algorithms use two major procedures:

(i) Extract the surrounding features for the focused node pair.
(ii) Estimate the dependency relation operation for the focused node by a machine learning method.

In section 2.1, we describe these algorithms and then we discuss the differences between two algorithms. In section 2.2, we present the machine learning methods—Support Vector Machines and Maximum Entropy method.

### 2.1 Parsing Algorithm

#### *2.1.1 Yamada's algorithm*

In Yamada's algorithm(Yamada, 2003), a dependency relation for each word position is represented by the following three operations: **Shift**, **Left** and **Right**. The operation is determined by a classifier, SVMs, based on the surrounding features. The determination is iterated until the classifier cannot make any further dependency relation on the whole sentence. The details of the three operations are as follows:

**Shift** means that there is no relation between the focused node and the succeeding node. In this case, the focused node moves to the succeeding node. The left figure in Figure 1 illustrates the **Shift** operation, in which the focused node is shown in a round box. In this operation, no dependency relation is constructed.

**Right** means that the focused node (*n*) becomes a child (dependent) of the succeeding node (*n+1*). The middle figure in Figure 1 illustrates the **Right** operation.

**Left** means that the focused node (*n*) becomes a parent of the succeeding node (*n+1*). The bottom figure in Figure 1 illustrates the **Left** operation.

Note that once either **Left** or **Right** operation is applied, either the focus or succeeding node becomes the child of the other node and will never be considered in future analysis. In other words, the dependent node cannot receive any more nodes as its child. To check the child to be complete, the classifier utilizes surrounding features which will be discussed in Section 3.1.2.

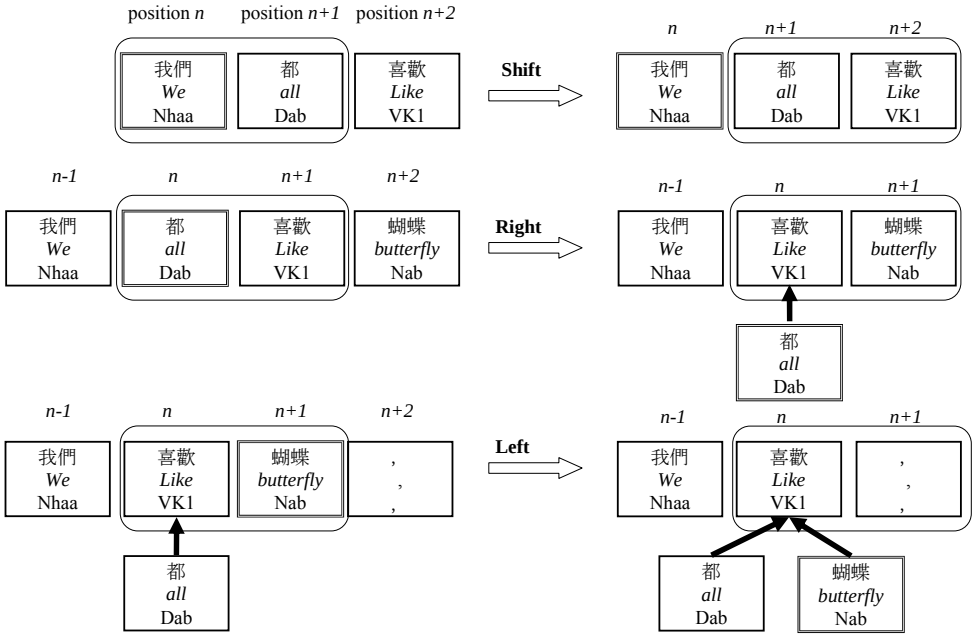



Figure 1: Three operations of Yamada's model

### *2.1.2 Nivre's algorithm*

In Nivre's algorithm, the analyzer's configurations are represented by a triple $\langle S, I, A \rangle$. *S* and *I* are stacks, *S* keeps the words being in consideration. *I* keeps input tokens yet to be analyzed. *A* is a list of dependency relations that are determined during the parsing process. Given an input token sequence *W*, the analyzer is initialized by the triple $\langle nil, W, \phi \rangle$. The analyzer estimates the dependency relation between two tokens (the top token *t* in *S* and the first token *n* in *I*). The algorithm iterates until the list *I* becomes empty. When the list *I* becomes empty, the analyzer stops the iteration and outputs the word dependency relation accumulated in *A*.

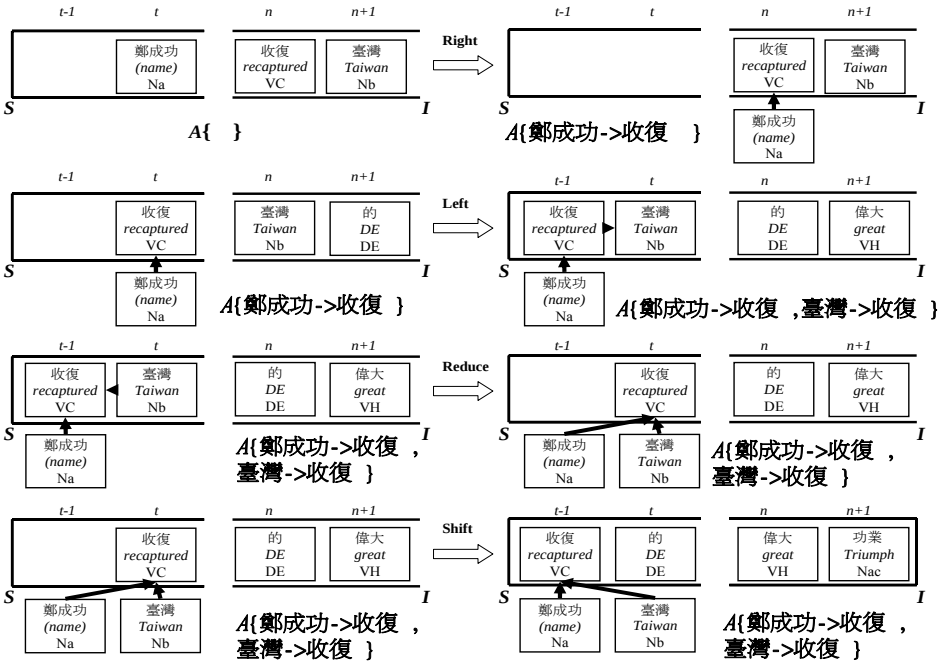



Figure 2: Four operations of Nivre's algorithm. Example: 鄭成功收復臺灣的偉大功業(**"The great triumph that Cheng Kheng-Koug recaptured Taiwan."**)

There are four possible operations to the next configuration:

**Right:** In the current triple $\langle t \mid S, n \mid I, A \rangle$ ($t$ is the top element and $S$ is the remaining elements in the stack), this operation indicates a dependency relation that the word $t$ depends on word $n$. Then, the analyzer extends $A$ with $(t \rightarrow n)$ and, removes $t$. The configuration becomes the triple $\langle S, n \mid I, A \cup \{(t \rightarrow n)\} \rangle$.

**Left:** In the current triple $\langle t \mid S, n \mid I, A \rangle$, this operation indicates that there is a dependency relation that the word $n$ depends on the word $t$. Then, the analyzer extends $A$ with $(n \rightarrow t)$, and pushes $n$ onto the stack. The configuration becomes the triple $\langle n \mid t \mid S, I, A \cup \{(n \rightarrow t)\} \rangle$.

**Reduce**: In the current triple $\langle t \mid S, n \mid I, A \rangle$, this operation indicates that there is no dependency relation between $n$ and $t$; no more words $n'$ ($n' \in I$) which may depend on $t$; and $t$ already has a parent on its left side. Then, the analyzer removes $t$ from the stack $S$, and the configuration now becomes the triple $\langle S, n \mid I, A \rangle$.

**Shift:** In the current triple $\langle t \mid S, n \mid I, A \rangle$, this operation indicates that there is no dependency relation between $n$ and $t$. Then the analyzer push $n$ onto the stack $S$, and the configuration now becomes the triple $\langle n \mid t \mid S, I, A \rangle$.

These operations are depicted in Figure 2. Given an input sentence of length $N$ (words), the analyzer is guaranteed to terminate after at most $2N$-1 actions. The dependency structure given at the termination is well-formed if and only if the subtrees are connected (Nivre, 2004).

It should be noted that the definition above is slightly different from the original algorithm (Nivre, 2004a). Similar to Yamada's algorithm, each word of input sentence becomes a token. The token includes the word, the POS, the information of its children, and other useful information. These become the features for the classifiers.

### *2.1.3 Comparison between the two algorithms*

In the original Yamada's algorithm[1], the operation **Shift** means either "There is no dependency relation between the focused node and its succeeding node" or "There is a dependency relation between the focused node and its succeeding node, but the operation should be delayed since there is a possibility that some nodes on the right side may depend on the node. Therefore the focused node should remain in token sequence". If the succeeding node has a potential child on the right side and the child hasn't been analyzed yet, the current node should wait for the child to be analyzed. However, if the succeeding node depends on the left node and it becomes a child of the focused node, the latter nodes may lose the correct dependencies. In our experiment, we arranged for the training data to let classifier predict correct operation (**Left** or **Shift**). However, there is high ambiguity in **Shift** operation. The ambiguity of **Shift** is a crucial error source in the experiment.

In contrast, there are four operations in Nivre's algorithm. If the current triple has a dependency relation, it is registered in *A*. Even if the relation is $(n \rightarrow t)$, the word *n* will be pushed onto *S*. This approach guarantees that the succeeding words can depend on *n* and doesn't lose the relation $(n \rightarrow t)$. This algorithm can resolve some problems concerning the ambiguity of **Shift** operation (in Yamada's algorithm). However, if the word *n* has no more child in the succeeding words, it should select **Reduce** operation after the word *n* being pushed onto *S*. If it doesn't select **Reduce** operation, the succeeding words in *I* will eventually depend on a wrong word. This is a major error source in our experiment discussed in Section 4. However, the result shows that the Nivre's algorithm can get better performance.

## 2.2 Machine Learning Models

In Yamada's method (Yamada, 2003), the operation is determined by SVMs (Vapnik, 1998). On the other hand, Nivre's method uses memory-based learning to determine the operations. In this paper, we use SVMs and MaxEnt in both algorithms to compare the models' performance.

SVMs are binary classifiers based on a maximum margin strategy. In our experiment, we use the polynomial kernel: $K(\boldsymbol{x},\boldsymbol{z}) = (1 + \boldsymbol{x} \cdot \boldsymbol{z})^d$ . To extend binary classifiers to multi-class classifiers, we use the pair-wise method which utilizes ${}_n C_2$ binary classifiers between all pairs of the classes (Kreel, 1998). We use Libsvm (Lin et al., 2001) in our experiments.

MaxEnt model is a statistical approach that has been applied to many classification tasks in NLP, such as prepositional phrase attachment classification, part-of-speech tagging, etc. (Berger et al., 1996; Ratnaparkhi et al. 1999).

---

[1] There are several versions of Yamada's algorithm, but we only consider the first version in this paper.

MaxEnt model combines evidence from different features without the independence assumption. Using this model in our task, we can assign consistent probabilities to an action conditioned on the context *c*. For an action *a*, the conditional probability $p(a \mid c)$ is calculated as below :

$$p(a \mid c) = \frac{1}{Z(c)} \exp\left( \sum_{j=1}^{k} \lambda_j f_j(a,c) \right) \qquad (1)$$

In equation (1), *a* and *c* represent an action and a context respectively. *Z(c)* is the normalization factor. $f_j(a,c)$ represents the *j*th feature for the action and *k* is the total number of features used. The features are binary-valued. MaxEnt model calculates the maximum likelihood by training with annotated training data. In our experiments, we used Opennlp-Maxent Package (Baldridge et al, 2001), which implements the GIS (Darroch, 1972) algorithm. The feature selection is presented in the next section.

(Yamada, 2003) proposed that they divided training examples according to the POS-tags of focused node to reduce the computational cost for training. We also make a set of pair-wise SVM classifiers or a MaxEnt classifier for each divided example group.

## 3. Experiments

### 3.1 Corpus and Features

#### *3.1.1 Corpus*

We use the CKIP Chinese Treebank Version 2.0 (K-J Chen et al., 1999) to train our analyzer. The corpus includes 54,902 phrase structure trees and 290,114 words in 23 files. The corpus has the following three properties:

(i) Word segmented, POS-tagged, parsed (phrase structure) and head annotated

(ii) Balanced corpus

(iii) Clause segmented by punctuation marks (commas and full stops)

Table 1 shows a sample phrase structure in the CKIP Treebank and its conversion to dependency structure to be used in our experiment data.

NP(property:S的(head:S(agent:NP(Head:Nba:鄭成功)| Head:VC31:收復 |theme:NP(Head:Nca:臺灣))|Head:DE:的)|property:VH11:偉大|Head:Nac:功業)

| Word | 鄭成功 | 收復 | 臺灣 | 的 | 偉大 | 功業 |
|---|---|---|---|---|---|---|
| POS | Nba | VC31 | Nca | DE | VH11 | Nac |
| Node ID | 0 | 1 | 2 | 3 | 4 | 5 |
| Parent node | 1 | 3 | 1 | 5 | 5 | -1 |

Table 1: Tree structure conversion from phrase structure(the upper part) to dependency structure(the lower part). (*"The great triumph that Cheng Kheng-Koug recaptured Taiwan."*)

#### *3.1.2 Features*

The features in Yamada's algorithm are the words and their POS tags of 5 local nodes (2 preceding nodes *n-2, n-1*, the focused node, and 2 succeeding nodes *n+1, n+2*) and their child nodes. The detailed description is in Figure 3.

In Nivre's algorithm, the analyzer considers the dependency of two nodes (*n*,*t*) which are in current triple. Similar to Yamada's algorithm, we select these features: 2 preceding nodes of node *t, t* itself, and 2 succeeding nodes in stack *I* , together with their child nodes.

In SVMs, the polynomial kernel enables the analyzer to consider combination of relevant features automatically. Figure 4 presents the feature of Nivre's model with SVMs.

The MaxEnt, however, cannot estimate combination of features. We select some useful combined features and add them in MaxEnt. We select other features which are independent to basic features, for instance: the distance between node *n* and *t*, the preceding action…etc.

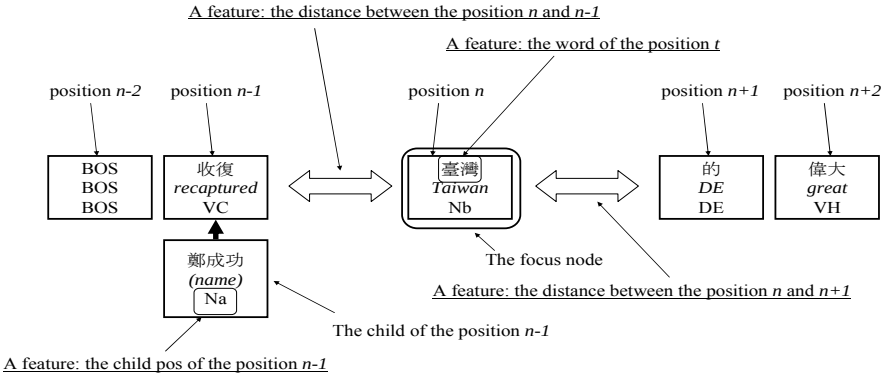



Figure 3: Features in Yamada's algorithm with SVMs and MaxEnt:

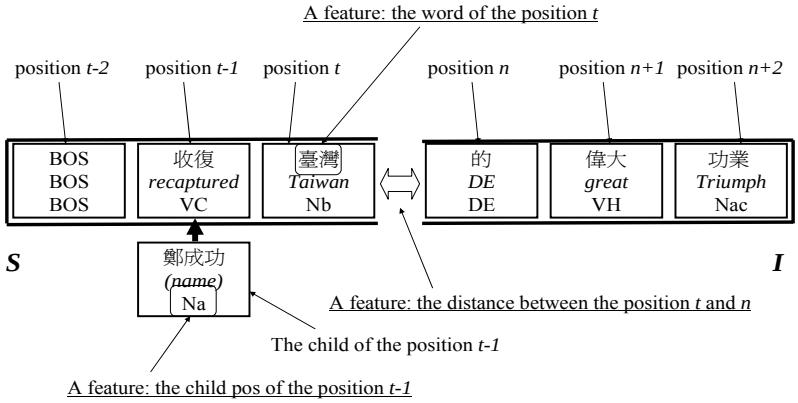



Figure 4: Features in Nivre's algorithm with SVMs and MaxEnt:

## 3.2 Experiments and results

### *3.2.1 Experiment Setting*

In our experiments, we verify the accuracy of Yamada's and Nivre's algorithms. Since CKIP Treebank is a balanced corpus, there are multi-source of articles in corpus. We select four testing data corresponding to the sources.

**Experiment 1**: We implement the Yamada's and Nivre's algorithms with SVMs and MaxEnt. There are four associations of results in this experiment. These associations are: Nivre algorithm with SVMs, Nivre algorithm with MaxEnt, Yamada algorithm with SVMs, and Yamada algorithm with MaxEnt. The training data has 229,764 words (41,057 sentences) and the result is shown in Table 2.

**Experiment 2:** After Experiment 1, we analyze the errors in Experiment 1 and found that some errors are caused by the mistake of nominal compound analysis. Therefore we try to solve this problem by extracting nominal compounds. The definition of nominal compound (base-NP) in our experiment is a noun phrase in which each noun except for the last noun should depends on the last noun. For instance, the head word of the base-NP "行政院 /經濟 / 建設 /委員會(Council for Economic Planning and Development)" is the last word(noun) "委員會" and all other nouns in it depend on the head.

We use two methods to extract the nominal compounds. First, we use the NP-chunker composed by "YamCha"(Kudo, 2004). YamCha is a customizable text chunker based on SVMs. Second, we extract the base-NP based on the annotation of the test data. The reason that we extracted nominal compounds from the testing data is that we want to discuss the true impact of extracting nominal compounds. Before the analysis starts, we use the NP-chunker to chunk the nominal compounds in the testing data. Table 3 described these results.

All these experiments are implemented on a Linux machine with XEON 2.4GHz dual CPUs and 4.0GB memory.

#### *3.2.2 Results*

The performance of our dependency structure analyzer is evaluated by the following three measures:

Dependency Accuracy: $(Dep.\ Acc.) = \frac{\#\ of\ \ correctly\ analyzed\ dependency\ relation}{\#\ of\ \ dependency\ relation}$

Root Accuracy: $(Root.\ Acc.) = \frac{\#\ of\ \ correctly\ analyzed\ root\ nodes}{\#\ of\ \ clauses}$

Sentence Accuracy: $(Sent.\ Acc.) = \frac{\#\ of\ \ fully\ correctly\ analyzed\ clause}{\#\ of\ \ clauses}$

Table 2 shows the result of Experiment 1. This table compares the performance of two algorithms and two machine learning methods. The first column shows four testing setting and rows show the accuracy measures described above. The result shows that Nivre's algorithm achieves better performance than Yamada's algorithm both with SVMs and MaxEnt.

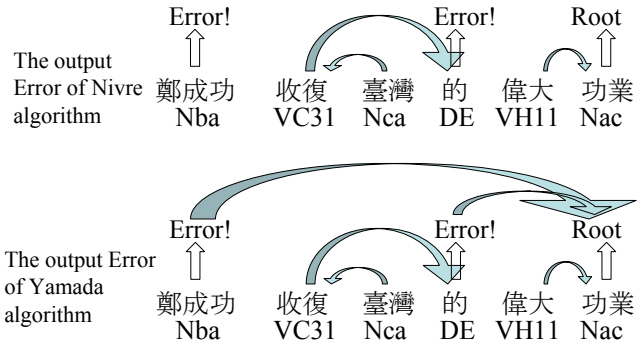
Alternatively, comparing the performance of different machine learning methods, the result shows that SVMs shows better performance than MaxEnt both in Nivre's algorithm and Yamada's algorithm.

According to the results, in some testing data, the root accuracy of Yamada's algorithm is better than Nivre's algorithm (in SVMs). This is because that the Yamada's algorithm iterates the analysis process until all of the remaining words are analyzed as ***Shift***. However, the ending condition of Nivre algorithm is that the stack *I* becomes empty. Considering the example in Figure 5, there are three words which do not have head word (including the root word "功業 (*triumph*)", error words "鄭成功 (*Cheng Cheng-Kung*)"and "的 (DE)"). The Nivre algorithm terminates the analysis process

when the stack *I* becomes empty. On the other hand, Yamada algorithm iterates the process again and tends to let the words depend on the root word. Therefore, Yamada algorithm analyzes the sentence as a whole dependency structure but includes some errors (the words "鄭成功 (*Cheng Cheng-Kung*)"and "的 (DE)"). Therefore, although the dependency accuracy and sentence accuracy are worse than Nivre algorithm, the root accuracy of Yamada algorithm is better than Nivre algorithm.

| Test setting | | Textbook | Newspaper | Anthology | Magazine |
|---|---|---|---|---|---|
| Nivre, SVMs | Dep. | **94.61** | **87.86** | **95.06** | **87.71** |
| | Root | 95.01 | 89.91 | **96.46** | 88.34 |
| | Sent. | **89.09** | **72.16** | **88.18** | **69.97** |
| Yamada, SVMs | Dep. | 94.37 | 86.40 | 94.80 | 87.19 |
| | Root | **96.18** | 88.98 | 96.24 | **89.33** |
| | Sent. | 87.56 | 68.40 | 87.07 | 67.35 |
| Nivre, MaxEnt | Dep. | 93.86 | 86.70 | 92.63 | 85.74 |
| | Root | 95.71 | **90.13** | 95.47 | 88.79 |
| | Sent. | 87.09 | 69.20 | 84.32 | 65.57 |
| Yamada, MaxEnt | Dep. | 91.83 | 82.61 | 91.63 | 83.10 |
| | Root | 95.77 | 89.02 | 94.92 | 89.16 |
| | Sent. | 81.87 | 59.77 | 80.46 | 58.43 |

Table 2: The result of experiment 1 (Among 4 association)




(*The great triumph that Cheng Cheng-Kung recaptured Taiwan.* )

Figure 5: The error processing of Yamada algorithm and Nivre algorithm

Table 3 shows the result of Experiment 2. This experiment conducts for Nivre algorithm with the SVM-based NP-chunker. Comparing two methods (Nivre, SVMs+NP and Nivre, SVMs), the performance decreases after using the NP-chunker. It seems that the NP-chunker gives bad effect, because the accuracy of our NP-chunker is as low as 88.21 (Recall) and 87.78 (Precision). There are many nominal compounds which cannot be extracted correctly.

To see this effect, we extracted correct nominal compounds in the testing data. The results are shown in Table 3 bottom. Comparing the results between the two experiments (Nivre algorithm +SVMs and Nivre algorithm +SVMs+true NP), the results show that extracting nominal compounds can improve the analysis performance on testing data "Newspaper" and "Magazine", especially in the sentence accuracy.

The training data size of our experiments is 229,764 words (41,057 sentences). According to our results, it is possible that if we have more training data, the accuracy will improve. When we change the data size, the parsing accuracy is improved by extracting correct nominal compounds. These results of experiments suggest the necessity of extracting nominal compounds in Chinese dependency analysis.

| Test data | | Textbook | Newspaper | Anthology | Magazine |
|---|---|---|---|---|---|
| Nivre, SVMs+NP | Dep. | 94.44 | 87.72 | 95.04 | 87.48 |
| | Root | 94.95 | 89.33 | **96.46** | 87.83 |
| | Sent. | 88.68 | 71.81 | 88.07 | 69.15 |
| Nivre, SVMs | Dep. | 94.61 | 87.86 | **95.06** | 87.71 |
| | Root | 95.01 | **89.91** | **96.46** | 88.34 |
| | Sent. | 89.09 | 72.16 | **88.18** | 69.97 |
| Nivre, SVMs+true NP | Dep. | **94.62** | **88.02** | **95.06** | **87.95** |
| | Root | 94.83 | 89.38 | 96.45 | 87.53 |
| | Sent. | **89.20** | **72.56** | 88.17 | **70.42** |

Table 3: The result of experiment 2 (effect of NP-chunker)

## 4. Discussion--Base NP

We use the NP-chunker, since there are many nominal compounds in Chinese text, especially in newspaper. Such as: 國際/貨幣/基金會(International Monetary Fund.), 太空/衛星/計畫 (satellite project)…etc. Most of nominal compounds are composed by two or three noun words. Table 4 shows the length distribution of nominal compounds in the testing data. The nominal compounds which include two words are 70~90% in different text categories. Therefore, our model analyzes a long noun sequence – namely a nominal compound, the model tends to separate the sequence into two or three base-NPs. For instance: base-NP "行政院 /經濟 /建設 /委員會" is an organization name. Each word should depend on the last word "委員會", but our analyzer without base NP chunker separates this base-NP into two base-NPs: "行政院" and "經濟 /建設 /委員會". Therefore, we should use the NP-chunker to avoid such errors.

Although we use the NP-chunker in our analyzer, the parsing performance is not improved. This is because the performance of the NP-chunker is not perfect and the nominal compounds composed by more than 3 words rarely occur in Chinese texts (See Table 4). However, these long nominal compounds will influence the performance of analyzer. In the Experiment 2, the results show that the perfect NP-chunker can improve the performance in testing data "Newspaper" and "Magazine". According to the result of McNemar test which compares the cases with and without using perfect NP-chunker, we can trust that perfect NP-chunker is effective (P-value = 0.0019 < 0.05 in testing data "Newspaper" and P-value = 0.0042 < 0.05 in testing data "Magazine").

The other reason why the NP chunker cannot improve the analyzer may come from the restricted definition of base-NPs in our experiment. There are many base-NPs that include not only nouns but also other part-of-speech words. For example, the NP "今年(Nd)/一(Nd)/到(Ca)/ 十月(Nd)" cannot be extracted by our NP-chunker. However these can be analyzed easily by some simple rules. We should extend the definition of base-NP and improve the performance of the NP-chunker.

| Length / Category | 2 words | 3 words | 4 words | More than 4 words | Title Nominal compounds |
|---|---|---|---|---|---|
| Anthology and Textbook | 461(92.2%) | 36(7.2%) | 3(0.6%) | 0(0%) | 500(100%) |
| Newspaper | 779(71.0%) | 239(22.0%) | 53(4.9%) | 15(1.3%) | 1086(100%) |
| Magazine | 1411(80.3 %) | 282(16.0%) | 55(3.1%) | 8(0.4%) | 1756(100%) |

Table 4: The length distribution of nominal compounds in different category

## 5. Future Work--PP Extraction

To analyze the prepositional phrases is another problem in our experiment. The structure of prepositional phrases is varied according to the scope of PP such as "prep.+ NP", "prep.+VP", or "prep.+S". It is difficult to identify the scope of PP. Sometimes PP scope will include a long sentence. However, our analyzer tends to let the preposition governing a partial sub-structure of a full sentence. For example, in the clause—"*直到*(until)prep./[主管(administer)/官署(government)/正視(serious)/此(this)/一(one)/問題(problem)]S (Until the government conducts this problem seriously)…", the prep. (*直到*) with clause(主管……問題) becomes a PP. But the output of our analyzer is the prep. (*直到*) with NP(主管/官署) as a PP, therefore the analysis after the word (官署) ─(正視/此/一/問題)─becomes an error. If the analyzer has the information about the scope of PP in advance, these errors can be solved. To deal with this problem, we should construct a useful PP chunker to extract the PP from input sentence.

## 6. Conclusion

In this paper, we presented a deterministic dependency structure analyzer for Chinese. This analyzer implements two algorithms with two machine learning methods, SVMs and MaxEnt. We compared the performance of the two models on a dependency tagged corpus derived from the CKIP Treebank corpus. The result shows that Nivre's method can resolve some problems in Yamada's method and the SVMs can obtain good performance. Next, we tried to add an NP-chunker in our model. Although the result of the NP-chunker experiment didn't show better performance, we showed that perfect NP-chunking has a potential to give better performance.

Future work includes three parts. First, as discussed in Sections 4 and 5, we should consider a good process for NP-chunker and PP-chunker. Second, in Nivre algorithm, the action **Reduce** needs the condition that the node *n* should have no more child in *I*. However, it is difficult to identify this condition. In some long sentences, the children of the focused node *n* may be far away from *n*. Therefore, the ambiguity between two actions – **Shift** and **Reduce** – should be resolved by long-distance dependency resolution model. Finally, for applying to wide variation of Chinese texts, we will evaluate our method on the Penn Chinese Treebank which contains articles from the newspapers in China.

**References**

Adam Berger, Stephen Della Pietra, and Vincent Della Pietra, 1996. *A Maximum Entropy Approach to Natural Language Processing,* Computational Linguistics, vol. 22, no. 1.

Chih Jen Lin, 2001.*A practical guide to support vector classification* , http://www.csie.ntu.edu.tw/~cjlin/libsvm/

Jason Baldridge, Tom Morton, and Gann Bierner, 2001, http://maxent.sourceforge.net/.

Eugene Charniak, 2001. *Immediate-Head Parsing for Language Models* NAACL-2001.

Keh-Jiann Chen, Chin-Ching Luo, Zhao-Ming Gao, Ming-Chung Chang, Feng-Yi Chen, Chao-Jan Chen, 1999, *The CKIP Tree-bank: Guidelines for Annotation,* Presented at ATALA Workshop, Paris, June 18-19.

Michael Collins, Brian Roark, 2004, *Incremental parsing with the Perceptron algorithm.* ACL-2004.

Darroch, J. Ratcli , D. 1972. *Generlized Iterative Scaling for Log-linear Models*. In Annals of Mathematical Statistics, volume 43, Issue 5, pages 1470-1480.

Ulrich. H.-G. Kreβel, 1998. *Pairwise classification and support vector machines.* In Advances in Kernel Methods, pages 255–268. The MIT Press.

Taku Kudo, Yuji Matsumoto, 2002. *Japanese Dependency Analyisis using Cascaded Chunking*, CONLL-2002.

Taku Kudo, Yuji Matsumoto, 2003. *Fast Methods for Kernel-Based Text Analysis,* ACL-2003.

Joakim Nivre, 2004a. *Incrementality in Deterministic Dependency Parsing.* ACL-2004.

Joakim Nivre, 2004b. *Inductive Dependency Parsing.* MSI Report 04070.

Adwait Ratnaparkhi, 1999. *Learning to parse natural language with maximum entropy models.* Machine Learning, 34(1-3):151–175.

Vladimir N. Vapnik, 1998. *Statistical Learning Theory.* A Wiley-Interscience Publication.

Hiroyasu Yamada and Yuji Matsumoto, 2003. Statistical Dependency Analysis with Support Vector Machines, IWPT 2003.