Japanese Dependency Analysis using a Deterministic Finite State Transducer

Satoshi Sekine
Computer Science Department
New York University
715 Broadway, 7th floor
New York, NY 10003, USA

Abstract

A deterministic finite state transducer is a fast device for analyzing strings. It takes O(n) time to analyze a string of length n. In this paper, an application of this technique to Japanese dependency analysis will be described. We achieved the speed at a small cost in accuracy. It takes about 0.17 millisecond to analyze one sentence (average length is 10 bunsetsu, based on PentiumIII 650MHz PC, Linux) and we actually observed the analysis time to be proportional to the sentence length. The accuracy is about 81% even though very little lexical information is used. This is about 17% and 9% better than the default and a simple system, respectively. We believe the gap between our performance and the best current performance on the same task, about 7%, can be filled by introducing lexical or semantic information.

1 Introduction

Syntactic analysis or parsing based on traditional methods, like Chart parsing or the GLR parsing algorithm, takes cubic or greater time in the sentence length to analyze natural language sentences. For Japanese, Sekine et al. (Sekine et al., 2000) proposed a Japanese dependency analyzer which analyzes sentences in time quadratic in the sentence length using a backward search algorithm. Recently, a number of research efforts using Finite State Transducers (FST) have been reported. Roche built an English syntactic analyzer by finding a fixed point in a non-deterministic FST (Roche, 1994). But it still can't analyze a sentence in time linear in the sentence length.

In this paper, we will propose a Japanese dependency analyzer using a Deterministic Finite State Transducer (DFST). The Japanese dependency structure is usually represented by relationships between phrasal units called 'bunsetsu'. A bunsetsu usually contains one or more content words, like a noun, verb or adjective, and zero or more function words, like a postposition (case marker) or verb/noun suffix. A dependency between two bunsetsu has a direction from a dependent to its head. Figure 1 shows examples of bunsetsu and dependencies. Each bunsetsu is separated by "`|`". The first segment "`KANOJO-HA`" consists of two words, `KANOJO` (She) and `HA` (subject case marker). The numbers in the "head" line show the head ID of corresponding bunsetsus. Note that the last segment does not have a head, and it is the head bunsetsu of the sentence. The task of the dependency analysis is to find the head ID for each bunsetsu.

2 Backward beam search algorithm

First, we would like to describe the backward beam search algorithm for Japanese dependency analysis proposed by Sekine et al. (Sekine et al., 2000). Their experiments suggested the method proposed in this paper.

The following characteristics are known for Japanese dependency. Sekine et al. assumed these characteristics in order to design the algorithm [1].

(1) Dependencies are directed from left to right

(2) Dependencies don't cross.

(3) Each bunsetsu except the rightmost one has only one head

(4) Left context is not necessary to determine a dependency.

[1]We know that there are exceptions (Shirai, 1998), but the frequencies of such exceptions are very small. Characteristic (4) is not well recognized, but based on our experiments with humans, it is true more than 90% of the time.

```
-----------------------------------------------------------------------
ID      1          2          3          4          5            6
     KANOJO-HA | KARE-GA | TSUKUTTA | PAI-WO | YOROKONDE | UKETOTTA.
     (She-subj)  (he-subj)  (made)  (pie-obj) (with pleasure) (received)
Head    6          3        4          6          6            -

Translation:  She received the pie made by him with pleasure.
-----------------------------------------------------------------------
```

Figure 1: Example of a Japanese sentence, bunsetsus and dependencies

Sekine et al. proposed a backward beam search algorithm (analyze a sentence from the tail to the head, or right to left). Backward search has two merits. Let's assume we have analyzed up to the $M+1$-st bunsetsu in a sentence of length N $(0 < M < N)$. Now we are deciding on the head of the M-th bunsetsu. The first merit is that the head of the dependency of the M-th bunsetsu is one of the bunsetsus between $M+1$ and N, which are already analyzed. Because of this, we don't have to keep a huge number of possible analyses, i.e. we can avoid something like active edges in a chart parser, or making parallel stacks in GLR parsing, as we can make a decision at this time. Also, we can use the beam search mechanism, by keeping a certain number of candidates of analyses at each bunsetsu. The width of the beam search can be easily tuned and the memory size of the process is proportional to the product of the input sentence length and the beam search width. The other merit is that the possible heads of the dependency can be narrowed down because of the assumption of non-crossing dependencies. For example, if the K-th bunsetsu depends on the L-th bunsetsu $(M < K < L)$, then the M-th bunsetsu can't depend on any bunsetsus between K and L. According to our experiment, this reduced the number of heads to consider to less than 50%.

Uchimoto et al. implemented a Japanese dependency analyzer based on this algorithm in combination with the Maximum Entropy learning method (Uchimoto et al., 2000). The analyzer demonstrated a high accuracy. Table 1 shows the relationship between the beam width and the accuracy of the system. From the table, we can see that the accuracy is not sensitive to the beam width, and even when the width is 1, which means that at each stage, the dependency is deterministically decided, the accuracy is almost the same as the best accuracy. This means that the left context of a dependency is not necessary to decide the dependency, which is closely related to characteristic (4). This result gives a strong motivation for starting the research reported in this paper.

Beam width	Dependency Accuracy	Sentence Accuracy
1	87.14	40.60
2	87.16	40.76
5	87.15	40.60
10	87.20	40.60
20	86.21	40.60

Table 1: Beam width and accuracy

3 Idea

Unfortunately, the fact that the beam width can be 1 by itself can not lead to the analysis in time proportional to the input length. Theoretically, it takes time quadratic in the length and this is observed experimentally as well (see Figure 3 in (Sekine et al., 2000)). This is due to the process of finding the head among the candidate bunsetsus on its right. The time to find the head is proportional to the number of candidates, which is roughly proportional to the number of bunsetsus on its right. The key idea of getting linear speed is to make this time a constant.

Table 2 shows the number of candidate bunsetsus and the relative location of the head among the candidates (the number of candidate bunsetsus from the bunsetsu being analyzed). The data is derived from 1st to 8th of

Num.of	Location of head													
cand.	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	7918													
2	12824	4948												
3	10232	3292	3126											
4	6774	2244	1439	1968										
5	3692	1294	888	660	1114									
6	1843	614	398	331	*291*	551								
7	848	278	176	133	*133*	*125*	264							
8	340	110	83	47	*38*	*57*	*68*	121						
9	137	33	28	17	*19*	*21*	*19*	*24*	46					
10	51	11	10	7	*3*	*10*	*3*	*6*	*8*	22				
11	17	5	2	0	*2*	*2*	*4*	*2*	*4*	*2*	10			
12	5	1	1	2	*0*	*0*	*0*	*0*	*2*	*2*	*1*	3		
13	3	0	1	0	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	0	
14	0	1	1	0	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	2

Table 2: Number of candidate and location of head

January part of the Kyoto corpus, which is a hand created corpus tagged with POS, bunsetsu and dependency relationships (Kurohashi, Nagao, 1997). It is the same portion used as the training data of the system described later. The number of candidates is shown vertically and the location of the head is shown horizontally. For example, 2244 in the fourth row and the second column means that there are 2244 bunsetsu which have 4 head candidates and the 2nd from the left is the head of the bunsetsu.

We can observe that the data is very biased. The head location is very limited. 98.7% of instances are covered by the first 4 candidates and the last candidate combined. In the table, the data which are not covered by the criteria are shown in italics. From this observation, we come to the key idea. We restrict the head candidate locations to consider, and we remember all patterns of categories at the limited locations. For each remembered pattern, we also remember where the head (answer) is. This strategy will give us a constant time selection of the head. For example, assume, at a certain bunsetsu in the training data, there are five candidates. Then we will remember the categories of the current bunsetsu, say "A", and five candidates, "B C D E F", as well as the head location, for example "2nd". At the analysis phase, if we encounter the same situation, i.e. the same category bunsetsu "A" to be analyzed, and the same categories of five candidates in the same order "B C D E F", then we can just return the remembered head location, "2nd". This process can be done in constant time and eventually, the analysis of a sentence can be done in time proportional to the sentence length.

For the implementation, we used a DFST in which each state represents the pattern of the candidates, the input of an edge is the category of the bunsetsu being analyzed and the output of an edge is the location of head.

4 Implementation

We still have several problems which have to be solved in order to implement the idea. Assuming the number of candidates to be 5, the problems are to

(1) define the categories of head bunsetsu candidates,

(2) limit the number of patterns (the number of states in DFST) to a manageable range, because the combination of five categories could be huge

(3) define the categories of input bunsetsus,

(4) deal with unseen events (sparseness problem).

In this section, we present how we implemented the system, which at the same time shows the solution to the problems. At the end, we implemented the system in a very small size; 1200 lines of C program, 188KB data file and less than 1MB processing size.

Structure of DFST

For the categories of head bunsetsu candidates, we used JUMAN's POS categories as the basis and optimized them using held-out data. JUMAN has 42 parts-of-speech (POS) including the minor level POSs, and we used the POS of the head word of a candidate bunsetsu. We also used the information of whether the bunsetsu has a comma or not. The number of categories becomes 18 after tuning it using the held-out data.

The input of an edge is the information about the bunsetsu currently being analyzed. We used the information of the tail of the bunsetsu (mostly function words), and it becomes 40 categories according to the same tuning process.

The output of an edge is the information of which candidate is the head. It is simply a number from 1 to 5. The training data contains examples which represent the same state and input, but different output. This is due to the rough definition of the categories or inherent impossibility of to disambiguating the dependency relationship from the given information only. In such cases, we pick the one which is the most frequent as the answer in that situation. We will discuss the problems caused by this strategy.

Data size

Based on the design described above, the number of states (or number of patterns) is 1,889,568 (18^5) and the number of edges is 75,582,720 as each state has 40 outgoing edges. If we implement it as is, we may need several hundred megabytes of data. In order to keep it fast, all the data should be in memory, and the current memory requirement is too large to implement.

To solve the problem, two ideas are employed. First, the states are represented by the combination of the candidate categories, i.e. states can be located by the 5 candidate categories. So, once it transfers from one state to another, the new state can be identified from the previous state, input bunsetsu and the output of the edge. Using this operation, the new state does not need to be remembered for each edge and this leads to a large data reduction. Second, we introduced the default dependency relationship. In the Japanese dependency relationship, 64% of bunsetsu depend on the next bunsetsu. So if this is the output, we don't record the edge as it is the default. In other words, if there is no edge information for a particular input at a particular state, the output is the next bunsetsu (or 1). This gave unexpectedly a large benefit. For unseen events, it is reasonable to guess that the bunsetsu depends the next bunsetsu. Because of the default, we don't have to keep such information. Actually the majority (more than 99%) of states and edges are unseen events, so the default strategy helps a lot to reduce the data size.

By the combination of the two ideas, there could be a state which has absolutely no information. If this kind of state is reached in the DFST, the output for any input is the next bunsetsu and the next state can be calculated from the information you have. In fact, we have a lot of states with absolutely no information. Before implementing the supplementation, explained in the next section, the number of recorded states is only 1,006 and there are 1,554 edges (among 1,889,568 possible states and 75,582,720 possible edges). After implementing the supplementation, we still have only 10,457 states and 31,316 edges. The data sizes (file sizes) are about 15KB and 188KB, respectively.

Sparseness problem

The amount of training data is about 8,000 sentences. As the average number of bunsetsu in a sentence is 10, there are about 72,000 data points in the training data. This number is very much smaller than the number of possible states (1,889,568), and it seems obvious that we will have a sparseness problem[2].

In order to supplement the unseen events, we use the system developed by Uchimoto et.al (Uchimoto et al., 2000). A large corpus is parsed by the analyzer, and the results

[2]However, we can make the system by using the default strategy and surprisingly the accuracy of the system is not so bad. This will be reported in the Experiment section

are added to the training corpus. In practice, we parsed two years of newspaper articles (2,536,229 sentences of Mainichi Shinbun 94 and 95, excluding January 1-10, 95, as these are used in the Kyoto corpus).

5 Experiment

In this section, we will report the experiment. We used the Kyoto corpus (version 2). The training is done using January 1-8, 95 (7,960 sentences), the test is done using January 9, 95 (1,246 sentences) and the parameter tuning is done using January 10, 95 (1,519 sentences; held-out data). The input sentences to the system are morphologically analyzed and bunsetsu are detected correctly.

Dependency Accuracy

Table 3 shows the accuracy of the systems. The 'dependency accuracy' means the percentage of the bunsetsus whose head is correctly analyzed. The bunsetsus are all but the last bunsetsu of the sentence, as the last bunsetsu has no head. The 'default method' is the system in which the all bunsetsus are supposed to depend on the next bunsetsu. The 'baseline method' is a quite simple rule-based system which was created by hand and has about 30 rules. The details of the system are reported in (Uchimoto et al., 1999). The 'Uchimoto's system' is the system reported in (Uchimoto et al., 2000). They used the same training data and test data. The dependency accuracies of our systems are 81% with supplementation and 78% without supplementation. The result is about 17% and 9% better than the default and the baseline methods respectively. Compared with Uchimoto's system, we are about 7% behind. But as Uchimoto's system used about 40,000 features including lexical features, and they also introduced combined features (up to 5), it is natural that our system, which uses only 18 categories and only combinations of two categories, has less accuracy[3].

System	Dependency Accuracy
Our system (with supp.)	81.231 %
Our system (without supp.)	77.972 %
Default method	64.14 %
Baseline method	72.57 %
Uchimoto's system	87.93 %

Table 3: Dependency Accuracy

Analysis speed

The main objective of the system is speed. Table 4 shows the analysis speed on three different platforms. On the fastest machine, it analyzes a sentence in 0.17 millisecond. Table 5 shows a comparison of the analysis speed of three different systems on SPARC-I. Our system runs about 100 times faster than Uchimoto's system and KNP(Kurohashi, Nagao, 1994).

Platform	Analysis time (millisec./sent.)
PentiumIII 650MHz	0.17
SunEnterprise, 400MHz	0.29
Ultra SPARC-I, 170MHz	1.03

Table 4: Analysis Speed

System	Analysis time (millisec./sent.)
Our system	1.03
Uchimoto's system	170
KNP	86

Table 5: Comparison of Analysis Speed

Figure 2 shows the relationship between the sentence length and the analysis time. We used the slowest machine (Ultra SPARC-I, 170MHz) in order to minimize observational errors. We can clearly observe that the analysis time is proportional to the sentence length, as was predicted by the algorithm.

The speed of the training is slightly slower than that of the analysis. The training on the smaller training data (about 8000 sentences) takes about 10 seconds on Ultra SPARC-I.

[3]However, our system uses context information of five bunsetsus, which is not used in Uchimoto's system.

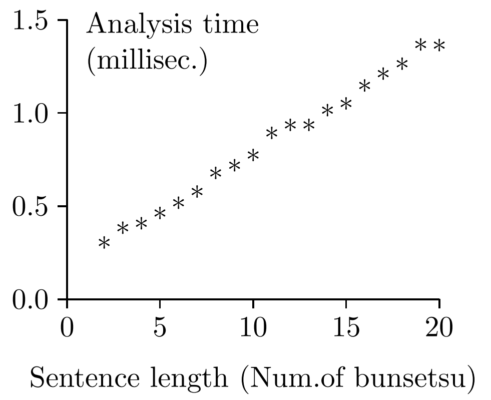

Figure 2: Analysis time and Sentence length

6 Discussion

6.1 The restriction

People may strongly argue that the main problem of the system is the restriction of the head location. We completely agree. We restrict the candidate to five bunsetsus, as we described earlier, and we theoretically ignored 1.3% of accuracy. Obviously, this restriction can be loosened by widening the range of the candidates. For example, 1.3% will be reduced to 0.5% if we take 6 instead of 5 candidates. Implementation of this change is easy. However, the problem lies with the sparseness. As shown in Table 2, there are fewer examples with a large number of bunsetsu candidates. For example, there are only 4 instances which have 14 candidates. It may be impossible to accumulate enough training examples of these kinds, even if we use a lot of untagged text for the supplementation. In such cases, we believe, we should have other kinds of remedies. One of them is a generalization of a large number candidates to a smaller number candidates by deleting unimportant bunsetsus. This remains for future research.

We can easily imagine that other systems may not analyze accurately the cases where the correct head is far from the bunsetsu. For example, Uchimoto's system achieves an accuracy of 41% in the cases shown in italics in Table 2, which is much lower than 88%, the system's overall accuracy. So, relative to the Uchimoto's system, the restriction caused a loss of accuracy of only 0.5% (1.3% x 41%) instead of 1.3%.

6.2 Accuracy

There are several things to do in order to achieve better accuracy. One of the major things is to use the information which has not been used, but is known to be useful to decide dependency relationships. Because the accuracy of the system against the training data is only 81.653% (without supplementation), it is clear that we miss some important information,

We believe the lexical relationships in verb frame element preference (VFEP) is one of the most important types of information. Analyzing the data, we can find evidence that such information is crucial. For example, there are 236 examples in the training corpus where there are 4 head candidates and they are bunsetsus whose heads are noun, verb, noun and verb, and the current bunsetsu ends with a `kaku-joshi`, a major particle. Out of the 236 examples, the number of cases where the first, second, third and last candidate is the head are 60, 142, 3 and 31, respectively. The answer is widely spread, and as the current system takes the most frequent head as the answer, 94 cases out of 236 (40%) are ignored. This is due to the level of categorization which uses only POS information. Looking at the example sentences in this case, we can observe that the VFEP could solve the problem.

It is not straightforward to add such lexical information to the current framework. If such information is incorporated into the state information, the number of states will become enormous. We can alternatively take an approach which was taken by augmented CFG. In this approach, the lexical information will be referred to only when needed. Such a process may slow down the analyzer, but since the number of invocation of the process needed in a sentence may be proportional to the sentence length, we believe the entire process may still operate in time proportional to the sentence length.

7 Conclusion

We proposed a Japanese dependency analysis using a Deterministic Finite State Transducer (DFST). The system analyzes a sentence with fairly good accuracy of about 81% in 0.17 millisecond on average. It can be applied to a wide range of NLP applications, including real time Machine Translation, Information Extraction and Speech Recognition. There are a number of efforts to improve the accuracy of Japanese dependency analysis (Haruno et.al, 1997) (Fujio, Matsumoto, 1998) (Kanayama et.al, 2000). In particular, it is interesting to see that Kanayama's method uses a similar idea, limiting the head candidates, in order to improve the accuracy. We are planning to incorporate the fruit of these works to improve our accuracy. We believe our research is complementary to these research. We also believe the method proposed in this paper may be applicable to other languages which share the characteristics explained earlier, for example, Korean, Turkish and others.

8 Acknowledgment

The experiments in this paper were conducted using the Kyoto corpus. We really appreciate their effort and their kindness in making it public domain. We thank Dr. Kurohashi and other people who worked in the project. Mr.Uchimoto helped us to prepare the data and the experiment, and we would like to thank him.

References

Masakazu Fujio, Yuuji Matsumoto. 1998 : "Japanese Dependency Structure Analysis based on Lexicalized Statistics", *Proceedings of Third Conference on EMNLP* pp87-96

Hiroshi Kanayama, Kentaro Torisawa, Yutaka Mitsuishi, Jun'ichi Tsujii. 2000 : "A hybrid Japanese Parser with Hand-crafted Grammar and Statistics", *Proceedings of COLING-00, this proceedings*

Masahiko Haruno, Satoshi Shirai, Yoshifumi Ooyama. 1998 : "Using Decision Trees to Construct a Practical Parser", *Proceedings of COLING-ACL-98* pp505-511

Sadao Kurohashi, Makoto Nagao. 1994 : "KN Parser : Japanese Dependency/Case Structure Analyzer", *Proceedings of the Workshop on Sharable Natural Language Resources* pp48-55

Sadao Kurohashi, Makoto Nagao. 1997 : "Kyoto University text corpus project", *Proceedings of the ANLP-97, Japan* pp115-118

Emmanuel Roche. 1994 : "Two Parsing Algorithms by Means of Finite State Transducers", *Proceedings of COLING-94* pp431-435,

Satoshi Sekine, Kiyotaka Uchimoto, Hitoshi Isahara. 2000 : "Statistical Dependency Analysis using Backward Beam Search", *Proceedings of COLING-00, this proceedings*

Satoshi Shirai. 1998 : "Heuristics and its limitation", *Journal of the ANLP, Japan* Vol.5 No.1, pp1-2

Kiyotaka Uchimoto, Satoshi Sekine, Hitoshi Isahara. 1999 : "Japanese Dependency Structure Analysis Based on Maximum Entropy Models", *Journal of Information Processing Society of Japan* Vol.40, No.9, pp3397-3407

Kiyotaka Uchimoto, Masaki Murata, Satoshi Sekine, Hitoshi Isahara. 2000 : "Dependency Model Using Posterior Context", *Proceedings of the IWPT-00*