Linear-Time Dependency Analysis for Japanese

Manabu Sassano
Fujitsu Laboratories, Ltd.
4-1-1, Kamikodanaka, Nakahara-ku,
Kawasaki 211-8588, Japan
sassano@jp.fujitsu.com

Abstract

We present a novel algorithm for Japanese dependency analysis. The algorithm allows us to analyze dependency structures of a sentence in linear-time while keeping a state-of-the-art accuracy. In this paper, we show a formal description of the algorithm and discuss it theoretically with respect to time complexity. In addition, we evaluate its efficiency and performance empirically against the Kyoto University Corpus. The proposed algorithm with improved models for dependency yields the best accuracy in the previously published results on the Kyoto University Corpus.

1 Introduction

Efficiency in parsing as well as accuracy is one of very important issues in natural languages processing. Although we often focus much on parsing accuracy, studies of its efficiency are also important, especially for practical NLP applications. Improving efficiency without loss of accuracy is a really big challenge.

The main purpose of this study is to propose an efficient algorithm to analyze dependency structures of head final languages such as Japanese and to prove its efficiency both theoretically and empirically. In this paper, we present a novel efficient algorithm for Japanese dependency analysis. The algorithm allows us to analyze dependency structures of a sentence in linear-time while keeping a state-of-the-art accuracy. We show a formal description of the algorithm and discuss it theoretically with respect to time complexity. In addition to this, we evaluate its efficiency and performance empirically against the Kyoto University Corpus (Kurohashi and Nagao, 1998), which is a parsed corpus of news paper articles in Japanese.

The remainder of the paper is organized as follows. Section 2 describes the syntactic characteristics of Japanese and the typical sentence processing of Japanese. In Section 3 previous work of dependency analysis of Japanese as well as of English is briefly reviewed. After these introductory sections, our proposed algorithm is described in Section 4. Next, improved models for estimating dependency of two syntactic chunks called *bunsetsu*s are proposed in Section 5. Section 6 describes experimental results and discussion. Finally, in Section 7 we conclude this paper by summarizing our contributions and pointing out some future directions.

2 Parsing Japanese

2.1 Syntactic Properties of Japanese

The Japanese language is basically an SOV language. Word order is relatively free. In English the syntactic function of each word is represented with word order, while in Japanese postpositions represent the syntactic function of each word. For example, one or more postpositions following a noun play a similar role to declension of nouns in German, which indicates a grammatical case.

Based on such properties, a *bunsetsu*[1] was devised and has been used to analyze syntactically a sentence in Japanese. A *bunsetsu* consists of one or more content words followed by zero or more function words. By defining a bunsetsu like that, we can analyze a sentence in a similar way that is used when analyzing a grammatical role of words in inflecting languages like German.

Thus, strictly speaking, bunsetsu order rather than word order is free except the bunsetsu that contains a main verb of a sentence. Such bunsetsu must be placed at the end of the sentence. For example, the following two sentences have an identical meaning: (1) Ken-ga kanojo-ni hon-wo age-ta. (2) Ken-ga hon-wo kanojo-ni age-ta. (-ga: subject marker, -ni: dative case particle, -wo: accusative case particle. English translation: Ken gave a book to her.) Note that the rightmost bunsetsu 'age-ta,' which is composed of a verb stem and a past tense marker, has to be placed at the end of the sentence.

[1] 'Bunsetsu' is composed of two Chinese characters, i.e., 'bun' and 'setsu.' 'Bun' means a sentence and 'setsu' means a segment. A 'bunsetsu' is considered to be a small syntactic segment in a sentence. A *eojeol* in Korean (Yoon et al., 1999) is almost the same concept as a bunsetsu. Chunks defined in (Abney, 1991) for English are also very similar to bunsetsus.

We here list the constraints of Japanese dependency including ones mentioned above.

C1. Each bunsetsu has only one head except the rightmost one.

C2. Each head bunsetsu is always placed at the right hand side of its modifier.

C3. Dependencies do not cross one another.

These properties are basically shared also with Korean and Mongolian.

2.2 Typical Steps of Parsing Japanese

Since Japanese has the properties above, the following steps are very common in parsing Japanese:

1. Break a sentence into morphemes (i.e. morphological analysis).
2. Chunk them into bunsetsus.
3. Analyze dependencies between these bunsetsus.
4. Label each dependency with a semantic role such as agent, object, location, etc.

We focus on dependency analysis in Step 3.

3 Previous Work

We review here previous work, mainly focusing on time complexity. In English as well as in Japanese, dependency analysis has been studied (e.g., (Lafferty et al., 1992; Collins, 1996; Eisner, 1996)). The parsing algorithms in their papers require $O(n^3)$ time where n is the number of words.[2]

In dependency analysis of Japanese it is very common to use probabilities of dependencies between each two bunsetsus in a sentence. Haruno et al. (1998) used decision trees to estimate the dependency probabilities. Fujio and Matsumoto (1998) applied a modified version of Collins' model (Collins, 1996) to Japanese dependency analysis. Both Haruno et al., and Fujio and Matsumoto used the CYK algorithm, which requires $O(n^3)$ time, where n is a sentence length, i.e., the number of bunsetsus. Sekine et al. (2000) used Maximum Entropy (ME) Modeling for dependency probabilities and proposed a backward beam search to find the best parse. This beam search algorithm requires $O(n^2)$ time. Kudo and Matsumoto (2000) also used the same backward beam search together with SVMs rather than ME.

There are few statistical methods that do not use dependency probabilities of each two bunsetsus.

[2]Nivre (2003) proposes a deterministic algorithm for projective dependency parsing, the running time of which is linear. The algorithm has been evaluated on Swedish text.

	Ken-ga	kanojo-ni	ano	hon-wo	age-ta.
	Ken-subj	to her	that	book-acc	gave.
ID	0	1	2	3	4
Head	4	4	3	4	-

Figure 3: Sample Sentence

Sekine (2000) observed that 98.7% of the head locations are covered by five candidates in a sentence. Maruyama and Ogino (Maruyama and Ogino, 1992) also observed similar phenomena. Based on this observation, Sekine (2000) proposed an efficient analysis algorithm using deterministic finite state transducers. This algorithm, in which the limited number of bunsetsus are considered in order to avoid exhaustive search, takes $O(n)$ time. However, his parser achieved an accuracy of 77.97% on the Kyoto University Corpus, which is considerably lower than the state-of-the-art accuracy around 89%.

Another interesting method that does not use dependency probabilities between each two bunsetsus is the cascaded chunking model by Kudo and Matsumoto (2002) based on the idea in (Abney, 1991; Ratnaparkhi, 1997). They used the model with SVMs and achieved an accuracy of 89.29%, which is the best result on the Kyoto University Corpus. Although the number of dependencies that are estimated in parsing are significantly fewer than that either in CYK or the backward beam search, the upper bound of time complexity is still $O(n^2)$.

Thus, it is still an open question as to how we analyze dependencies for Japanese in linear time with a state-of-the-art accuracy. The algorithm described below will be an answer to this question.

4 Algorithm

4.1 Algorithm to Parse a Sentence

The pseudo code for our algorithm of dependency analysis is shown in Figure 1. This algorithm is used with any estimator that decides whether a bunsetsu modifies another bunsetsu. A trainable classifier, such as an SVM, a decision tree, etc., is a typical choice for the estimator. We assume that we have some classifier to estimate the dependency between two bunsetsus in a sentence and the time complexity of the classifier is not affected by the sentence length.

Apart from the estimator, variables used for parsing are only two data structures. One is for input and the other is for output. The former is a stack for keeping IDs of modifier bunsetsus to be checked. The latter is an array of integers that stores head IDs that have already been analyzed.

Following the presented algorithm, let us parse a

```
// Input: N: the number of bunsetsus in a sentence.
//      w[]: an array that keeps a sequence of bunsetsus in the sentence.
// Output: outdep[]: an integer array that stores an analysis result, i.e., dependencies between
//      the bunsetsus. For example, the head of w[j] is outdep[j].
//
// stack: a stack that holds IDs of modifier bunsetsus in the sentence. If it is empty, the pop
//      method returns EMPTY (−1).
// function estimate_dependency(j, i, w[]):
//      a function that returns non-zero when the j-th bunsetsu should
//      modify the i-th bunsetsu. Otherwise returns zero.
function analyze(w[], N, outdep[])
stack.push(0);      // Push 0 on the stack.
for (int i = 1; i < N; i++) {      // Variable i for a head and j for a modifier.
    int j = stack.pop();      // Pop a value off the stack.
    while (j != EMPTY && (i == N − 1 || estimate_dependency(j, i, w))) {
        outdep[j] = i;      // The j-th bunsetsu modifies the i-th bunsetsu.
        j = stack.pop();      // Pop a value off the stack to update j.
    }
    if (j != EMPTY)
        stack.push(j);
    stack.push(i);
}
```

Figure 1: Pseudo Code for Analyzing Dependencies. Note that "i == N - 1" means the i-th bunsetsu is the rightmost one in the sentence.

```
// indep[]: an integer array that holds correct dependencies given in a training corpus.
//
// function estimate_dependency(j, i, w[], indep[]):
//      a function that returns non-zero if indep[j] == i, otherwise returns zero.
//      It also prints a feature vector (i.e., an encoded example) with a label which is decided to be
//      1 (modify) or -1 (not modify) depending on whether the j-th bunsetsu modifies the i-th.
function generate_examples(w[], N, indep[])
stack.push(0);
for (int i = 1; i < N; i++) {
    int j = stack.pop();
    while (j != EMPTY && (i == N − 1 || estimate_dependency(j, i, w, indep))) {
        j = stack.pop();
    }
    if (j != EMPTY)
        stack.push(j);
    stack.push(i);
}
```

Figure 2: Pseudo Code for Generating Training Examples. Variables w[], N, and stack are the same as in Figure 1.

sample sentence in Figure 3. For explanation, we here assume that we have a perfect classifier as *estimate_dependency()* in Figure 1, which can return a correct decision for the sample sentence.

First, we push 0 (Ken-ga) on the stack for the bunsetsu ID at the top of the sentence. After this initialization, let us see how analysis proceeds at each iteration of the *for* loop. At the first iteration we check the dependency between the zero-th bunsetsu and the 1st (kanojo-ni). We push 0 and 1 because the zero-th bunsetsu does not modify the 1st. Note that the bottom of the stack is 0 rather than 1. Smaller IDs are always stored at lower levels of the stack. Due to this, we do not break the non-crossing constraint (C3. in Section 2.1).

At the second iteration we pop 1 off the stack and

check the dependency between the 1st bunsetsu and the 2nd (ano). Since the 1st does not modify the 2nd, we again push 1 and 2.

At the third iteration we pop 2 off the stack and check the dependency for the 2nd and the 3rd (honwo). Since the 2nd modifies the 3rd, the dependency is stored in *outdep[]*. The value of *outdep[j]* represents the head of the j-th bunsetsu. For example, *outdep[2] = 3* means the head of the 2nd bunsetsu is the 3rd. Then we pop 1 off the stack and check the dependency between the 1st and the 3rd. We push again 1 since the 1st does not modify the 3rd. After that, we push 3 on the stack. The stack now has 3, 1 and 0 in top-to-bottom order.

At the fourth iteration we pop 3 off the stack. We do not have to check the dependency between the 3rd and the 4th (age-ta) because the 4th bunsetsu is the last bunsetsu in the sentence. Now we set *outdep[3] = 4*. Next, we pop 1 off the stack. Also in this case, we do not have to check the dependency between the 1st and the 4th. Similarly the zero-th bunsetsu modifies the 4th. As a result we set *outdep[1] = 4* and *outdep[0] = 4*. Now the stack is empty and we finish the analysis function. Finally, we have obtained a dependency structure through the array *outdep[]*.

4.2 Time Complexity

At first glance, the upper bound of the time complexity of this algorithm seems to be $O(n^2)$ because it involves a double loop; however, it is not. We will show that the upper bound is $O(n)$ by considering how many times the condition part of the *while* loop in Figure 1 is executed. The condition part of the *while* loop fails $N-2$ times because the outer *for* loop will be executed from 1 to $N-1$. On the other hand, the same condition part successes $N-1$ times because *outdep[j] = i* is executed $N-1$ times. For each bunsetsu ID j, *outdep[j] = i* is surely executed once because by executing *j = stack.pop()* the value of j is lost and it is never pushed on the stack again. That is the body of the *while* loop will be executed at most $N-1$ times which is equal to the number of the bunsetsus except the last one. Therefore the total number of execution of the condition part of the *while* loop is $2N-3$, which is obtained by summing up $N-2$ and $N-1$. This means that the upper bound of time complexity is $O(n)$.

4.3 Algorithm to Generate Training Examples

When we prepare training examples for the trainable classifier used with this algorithm, we use the algorithm shown in Figure 2. It is almost the same as the algorithm for analyzing in Figure 1. The differences are that we give correct dependencies to *estimate_dependency()* through *indep[]* and we obviously do not have to store the head IDs to *outdep[]*.

4.4 Summary and Theoretical Comparison with Related Work

The algorithm presented here has the following features:

F1. It is independent on specific machine learning methodologies. Any trainable classifiers can be used.

F2. It scans a sentence just once in a left-to-right manner.

F3. The upper bound of time complexity is $O(n)$. The number of the classifier call, which is most time consuming, is at most $2N-3$.

F4. The flow and the used data structures are very simple. Therefore, it is easy to implement.

One of the most related models is the cascaded chunking model by (Kudo and Matsumoto, 2002). Their model and our algorithm share many features including F1.[3] The big difference between theirs and ours is how many times the input sentence has to be scanned (F2). With their model we have to scan it several times, which leads to some computational inefficiency, i.e., at the worst case $O(n^2)$ computation is required. Our strict left-to-right parsing is more suitable also for practical applications such as real time speech recognition. In addition, the flow and the data strucutres are much simpler (F4) than those of the cascaded chunking model where an array for chunk tags is used and it must be updated while scanning the sentence several times.

Our parsing method can be considered to be one of the simplest forms of shift-reduce parsing. The difference from typical use of shift-reduce parsing is that we do not need several types of actions and only the top of the stack is inspected. The reason for these simplicities is that Japanese has the C2 constraint (Sec. 2.1) and the target task is dependency analysis rather than CFG parsing.

5 Models for Estimating Dependency

In order to evaluate the proposed algorithm empirically, we use SVMs (Vapnik, 1995) for estimating dependencies between two bunsetsus because they have excellent properties. One of them is that combinations of features in an example are automatically considered with polynomial kernels. Excellent performances have been reported for many classification tasks. Please see (Vapnik, 1995) for formal descriptions of SVMs.

[3] Kudo and Matsumoto (2002) give more comprehensive comparison with the probabilistic models as used in (Uchimoto et al., 1999).

At *estimate_dependency()* in Figure 1, we encode an example with features described below. Then we give it to the SVM and receive the estimated decision as to whether a bunsetsu modifies the other.

5.1 Standard Features

By the "standard features" here we mean the feature set commonly used in (Uchimoto et al., 1999; Sekine et al., 2000; Kudo and Matsumoto, 2000; Kudo and Matsumoto, 2002). We employ the features below for each bunsetsu:

1. Rightmost Content Word - major POS, minor POS, conjugation type, conjugation form, surface form (lexicalized form)
2. Rightmost Function Word - major POS, minor POS, conjugation type, conjugation form, surface form (lexicalized form)
3. Punctuation (periods, and commas)
4. Open parentheses and close parentheses
5. Location - at the beginning of the sentence or at the end of the sentence.

In addition, features as to the gap between two bunsetsus are also used. They include: distance, particles, parentheses, and punctuation.

5.2 Local Contexts of the Current Bunsetsus

Local contexts of a modifier and its possible head would be useful because they may represent fixed expressions, case frames, or other collocational relations. Assume that the j-th bunsetsu is a modifier and the i-th one is a possible head. We consider three bunsetsus in the local contexts of the j-th and the i-th: the $(j-1)$-th bunsetsu if it modifies the j-th, the $(i-1)$-th one, and the $(i+1)$-th one. Note that in our algorithm the $(i-1)$-th always modifies the i-th when checking the dependency between the j-th bunsetsu and the i-th where $j < i-1$. In order to keep the data structure simple in the proposed algorithm, we did not consider more distant bunsetsus from both the j-th and the i-th. It is easy to check whether the $(j-1)$-th bunsetsus modifies the j-th one through *outdep[]*. Note that this use of local contexts is similar to the dynamic features in (Kudo and Matsumoto, 2002)[4].

5.3 Richer Features Inside a Bunsetsu

With the standard features we will miss some case particles if the bunsetsu has two or more function words. Suppose that a bunsetsu has a topic marker as well as a case particle. In this case the case particle is followed by the topic marker. Thus we miss the case particle since in the standard features only the rightmost function word is employed. In order to capture this information, we use as features also all the particles in each bunsetsu.

Another important features missed in the standard are ones of the leftmost word of a possible head bunsetsu, which often has a strong association, e.g., an idiomatic fixed expression, with the rightmost word of its modifier. Furthermore, we use as a feature the surface form of the leftmost word of the bunsetsu that follows a possible head. This feature is used with ones in Section 5.2.

5.4 Features for Conjunctive Structures

Detecting conjunctive structures is one of hard tasks in parsing long sentences correctly. Kurohashi and Nagao (1994) proposed a method to detect conjunctive structures by calculating similarity scores between two sequences of bunsetsus.

So far few attempts have been made to explore features for detecting conjunctive structures. As a first step we tried two preliminary features for conjunctive structures. If the current modifier bunsetsu is a *distinctive key bunsetsu* (Kurohashi and Nagao, 1994, page 510), these features are triggered. One is a feature which is activated when a modifier bunsetsu is a distinctive key bunsetsu. The other is a feature which is activated when a modifier is a distinctive key bunsetsu and the content words of both the modifier and its possible head are equal to each other. For simplicity, we limit the POS of these content words to nouns.

6 Experimental Results and Discussion

We implemented a parser and SVM tools in C++ and used them for experiments.

6.1 Corpus

We used the Kyoto University Corpus Version 2 (Kurohashi and Nagao, 1998) to evaluate the proposed algorithm. Our parser was trained on the articles on January 1st through 8th (7,958 sentences) and tested on the article on January 9th (1,246 sentences). The article on January 10th were used for development. The usage of these articles is the same as in (Uchimoto et al., 1999; Sekine et al., 2000; Kudo and Matsumoto, 2002).

6.2 SVM setting

Polynomial kernels with the degree of 3 are used and the misclassification cost is set to 1 unless stated otherwise.

6.3 Results

Accuracy. Performances of our parser on the test set is shown in Table 1. For comparison to pre-

[4]Their model extracts three types of dynamic features from modifiers of the j-th bunsetsu (Type B), modifiers of the i-th bunsetsu (Type A), and heads of the i-th bunsetsu (Type C). Since in our proposed algorithm analysis proceeds in a left-to-right manner, we have to use stacking (Wolpert, 1992) or other techniques to employ the type C features.

	Dependency Acc.(%)	Sentence Acc.(%)
Standard	88.72	45.88
Full	89.56	48.35
w/o Context	88.91	46.33
w/o Rich	89.19	47.05
w/o Conj	89.41	47.86

Table 1: Performance on Test Set. Context, Rich, and Conj mean the features in Sec. 5.2, 5.3, and 5.4, respectively.

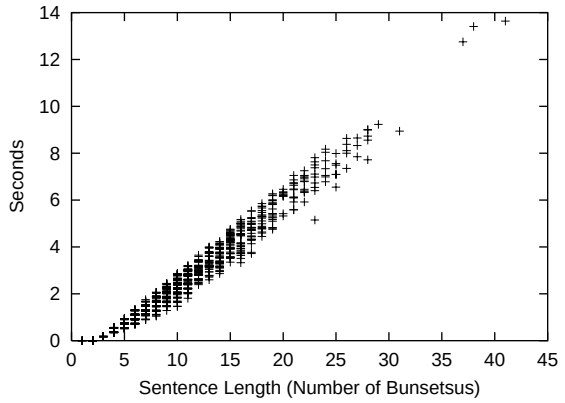

Figure 4: Observed Running Time

vious work we use the standard measures for the Kyoto University Corpus: dependency accuracy and sentence accuracy. The dependency accuracy is the percentage of correct dependencies and the sentence accuracy is the percentage of sentences, all the dependencies in which are correctly analyzed.

The accuracy with the standard feature set is relatively good. Actually, this accuracy is almost the same as that of the cascaded chunking model without dynamic features (Kudo and Matsumoto, 2002). Our parser with the full feature set yields an accuracy of 89.56%, which is the best in the previously published results.

Asymptotic Time Complexity. Figure 4 shows the running time of our parser on the test set using a workstation (Ultra SPARC II 450 MHz with 1GB memory). It clearly shows that the running time is proportional to the sentence length and this observation is consistent with our theoretical analysis in Section 4.2.

One might think that although the upper bound of time complexity is lower than those of previous work, actual processing of our parser is not so fast. Slowness of our parser is mainly due to a huge computation of kernel evaluations in SVMs. The SVM

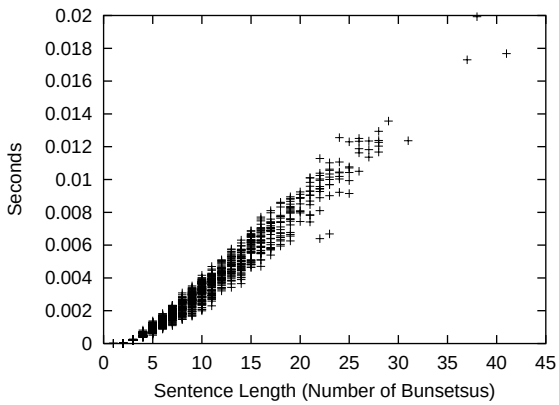

Figure 5: Observed Running Time with Linear Kernel. The misclassification cost is set to 0.0056.

classifiers in our experiments have about forty thousand support vectors. Therefore, for every decision of dependency also a huge computation of dot products is required. Fortunately, solutions to this problem have already been given by Kudo and Matsumoto (2003). They proposed methods to convert a polynomial kernel with higher degrees to a simple linear kernel and reported a new classifier with the converted kernel was about 30 to 300 times faster than the original one while keeping the accuracy. By applying their methods to our parser, its processing time would be enough practical.

In order to roughly estimate the improved speed of our parser, we built a parser with a linear kernel and ran it on the same test set. Figure 5 shows the observed time of the parser with a linear kernel using the same machine. The parser runs fast enough. It can parse a very long sentence within 0.02 seconds. Furthermore, accuracy as well as speed of this parser was much better than we expected. It achieves a dependency accuracy of 87.36% and a sentence accuracy of 40.60%. These accuracies are slightly better than those in (Uchimoto et al., 1999), where combinations of features are manually selected.

6.4 Comparison to Related Work

We compare our parser to those in related work. A summary of the comparison is shown in Table 2. It clearly shows that our proposed algorithm with SVMs has a good property with regard to time complexity and in addition our parser successfully achieves a state-of-the-art accuracy.

Theoretical comparison with (Kudo and Matsumoto, 2002) is described in Section 4.4. Uchimoto et al. (1999) used the backward beam search with ME. According to (Sekine et al., 2000), the analyzing time followed a quadratic curve. In contrast,

	Algorithm/Model	Time Complexity	Acc.(%)
This paper	Stack Dependency Analysis (SVMs)	n	89.56
	Stack Dependency Analysis (linear SVMs)	n	87.36
KM02	Cascaded Chunking (SVMs)	n^2	89.29
KM00	Backward Beam Search (SVMs)	n^2	89.09
USI99	Backward Beam Search (ME)	n^2	87.14
Seki00	Deterministic Finite State Transducer	n	77.97

Table 2: Comparison to Related Work. KM02 = Kudo and Matsumoto 2002, KM00 = Kudo and Matsumoto 2000, USI99 = Uchimoto et al. 1999, and Seki00 = Sekine 2000.

our parser analyzes a sentence in linear time keeping a better accuracy. Sekine (2000) also proposed a very fast parser that runs in linear time; however, accuracy is greatly sacrificed.

7 Conclusion and Future Directions

We have presented a novel algorithm for Japanese dependency analysis. The algorithm allows us to analyze dependency structures of a sentence in linear-time while keeping a state-of-the-art accuracy. We have shown a formal description of the algorithm and discussed it theoretically in terms of time complexity. In addition, we have evaluated its efficiency and performance empirically against the Kyoto University Corpus. Our parser gives the best accuracy, 89.56%, in the previously published results.

In the future, it would be interesting to apply this algorithm to speech recognition in which it is more desirable to analyze a sentence in a left-to-right manner. Another interesting direction would be to explore features for conjunctive structures. Although we found some useful features, they were not enough to improve the performance much. We expect stacking would be useful.

References

S. P. Abney. 1991. Parsing by chunks. In R. C. Berwick, S. P. Abney, and C. Tenny, editors, *Principle-Based Parsing: Computation and Psycholinguistics*, pages 257–278. Kluwer Academic Publishers.

M. Collins. 1996. A new statistical parser based on bigram lexical dependencies. In *Proc. of ACL-96*, pages 184–191.

J. M. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proc. of COLING-96*, pages 340–345.

M. Fujio and Y. Matsumoto. 1998. Japanese dependency structure analysis based on lexicalized statistics. In *Proc. of EMNLP-1998*, pages 88–96.

M. Haruno, S. Shirai, and Y. Ooyama. 1998. Using decision trees to construct a practical parser. In *Proc. of COLING/ACL-98*, pages 505–511.

T. Kudo and Y. Matsumoto. 2000. Japanese dependency structure analysis based on support vector machines. In *Proc. of EMNLP/VLC 2000*, pages 18–25.

T. Kudo and Y. Matsumoto. 2002. Japanese dependency analysis using cascaded chunking. In *Proc. of CoNLL-2002*, pages 63–69.

T. Kudo and Y. Matsumoto. 2003. Fast methods for kernel-based text analysis. In *Proc. of ACL-03*, pages 24–31.

S. Kurohashi and M. Nagao. 1994. A syntactic analysis method of long Japanese sentences based on the detection of conjunctive structures. *Computational Linguistics*, 20(4):507–534.

S. Kurohashi and M. Nagao. 1998. Building a Japanese parsed corpus while improving the parsing system. In *Proc. of the 1st LREC*, pages 719–724.

J. Lafferty, D. Sleator, and D. Temperley. 1992. Grammatical trigrams: A probabilistic model of link grammar. In *Proc. of the AAAI Fall Symp. on Probabilistic Approaches to Natural Language*, pages 89–97.

H. Maruyama and S. Ogino. 1992. A statistical property of Japanese phrase-to-phrase modifications. *Mathematical Linguistics*, 18(7):348–352.

J. Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proc. of IWPT-03*, pages 149–160.

A. Ratnaparkhi. 1997. A linear observed time statistical parser based on maximum entropy models. In *Proc. of EMNLP-1997*, pages 1–10.

S. Sekine, K. Uchimoto, and H. Isahara. 2000. Backward beam search algorithm for dependency analysis of Japanese. In *Proc. of COLING-00*, pages 754–760.

S. Sekine. 2000. Japanese dependency analysis using a deterministic finite state transducer. In *Proc. of COLING-00*, pages 761–767.

K. Uchimoto, S. Sekine, and H. Isahara. 1999. Japanese dependency structure analysis based on maximum entropy models. In *Proc. of EACL-99*, pages 196–203.

V. N. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer-Verlag.

D. H. Wolpert. 1992. Stacked generalization. *Neural Networks*, 5:241–259.

J. Yoon, K. Choi, and M. Song. 1999. Three types of chunking in Korean and dependency analysis based on lexical association. In *Proc. of the 18th Int. Conf. on Computer Processing of Oriental Languages*, pages 59–65.