Efficient Word Lattice Generation for Joint Word Segmentation and POS Tagging in Japanese

Nobuhiro Kaji[*] and **Masaru Kitsuregawa**[*†]
[*]Institute of Industrial Science, The University of Tokyo
[†]National Institute of Informatics
{kaji, kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract

This paper investigates the importance of a word lattice generation algorithm in joint word segmentation and POS tagging. We conducted experiments on three Japanese data sets to demonstrate that the previously proposed pruning-based algorithm is in fact not efficient enough, and that the pipeline algorithm, which is introduced in this paper, achieves considerable speed-up without loss of accuracy. Moreover, the compactness of the lattice generated by the pipeline algorithm was investigated from both theoretical and empirical perspectives.

1 Introduction

Many approaches to joint word segmentation and POS tagging can be interpreted as reranking with a word lattice (Jiang et al., 2008), wherein a small lattice is generated for an input sentence, and then the lattice paths are reranked to obtain the optimal one. Examples of such a method include (Asahara and Matsumoto, 2000; Kudo et al., 2004; Kruengkrai et al., 2006; Jiang et al., 2008).

In such a framework, it is crucial to develop an efficient lattice generation algorithm. Since there are ${}_{n+1}C_2 = O(n^2)$ word candidates, where n is the number of characters in the sentence, to be included in the lattice, it is prohibitively expensive to check all of them exhaustively. Such a naive method constitutes a severe bottleneck in a reranking system. Accordingly, in practice, it is necessary to resort to some technique to speed-up lattice generation.

It is, however, not straightforward to speed-up lattice generation for reranking, because there are requirements that the lattice has to satisfy and it is necessary to achieve a speed-up while satisfying those requirements. Most importantly, the lattice should contain a sufficient amount of correct words; otherwise, the accuracy of the reranking system will be seriously degraded. Moreover, the lattice should be small: an excessively large lattice spoils the efficiency of the reranking system because it is expensive to find the optimal path of such a lattice.

For the reasons stated above, it is not readily obvious what sort of technique is effective for lattice generation. Despite its practical importance, this question, however, has not been well studied. For example, (Kudo et al., 2004) used a dictionary to filter word candidates. While indeed efficient, such a method is obviously prone to removing out-of-vocabulary (OOV) words from a lattice and degrade accuracy (Uchimoto et al., 2001). Jiang et al. (2008) employed a pruning-based algorithm to reduce the $O(n^2)$ cost, but they did not investigate computational time required.

Given the above issues, the present study revisits lattice reranking by exploring the effectiveness of the lattice generation algorithm. Specifically, large-scale experiments were conducted on three Japanese data sets. The results of the experiments show that the pruning-based algorithm (Jiang et al., 2008) in fact incurs a non-negligible computational cost, which constitutes a bottleneck in the reranking system. Moreover, a pipelined lattice generation algorithm (see Section 3) was investigated as an alternative to the pruning-based one, and it was demonstrated that the reranking system using the pipeline algorithm speeds up the reranking more than 10 times without loss of accuracy. After that, the compactness of the lattice generated by the pipeline algorithm was examined from not

International Joint Conference on Natural Language Processing, pages 153–161,
Nagoya, Japan, 14-18 October 2013.

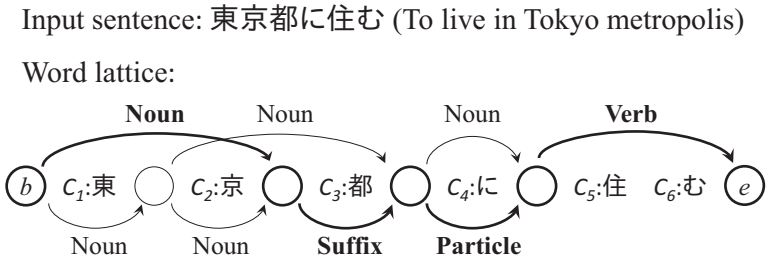

Figure 1: Example lattice (Kudo et al., 2004). The circle and arrow represent the node and edge, respectively. The bold edges represent the correct analysis.

only theoretical but also empirical perspectives.

The first contribution of this study is to shed light on the importance of the lattice generation algorithm in lattice reranking. As mentioned earlier, past studies paid little attention to elaborating the lattice generation algorithm. On the contrary, the results of our experiments reveal that the design of the lattice generation algorithm crucially affects the performance of the reranking system (including speed, accuracy, and lattice size).

The second contribution is to provide clear empirical evidence concerning the effectiveness of the pipeline algorithm. Although the pipeline algorithm itself is a simple application of well-known techniques (Xue, 2003; Peng et al., 2004; Neubig et al., 2011) and does not have much novelty, its effectiveness has been left unexplored in the context of lattice reranking. Consequently, its merits (or demerits) in relation to the pruning-based algorithm have also been unknown.

The third contribution is to develop an accurate reranking system based on the pipeline algorithm. The developed system achieved considerably higher F_1-score than three software tools that are widely used in Japanese NLP (JUMAN[1], MeCab[2], and Kytea[3]), while achieving high speed close to two of the three.

2 Preliminaries

As a preliminary, a word lattice and lattice reranking for joint word segmentation and POS tagging are explained in Sections 2.1 and 2.2, respectively. After that, the pruning-based lattice generation algorithm proposed by Jiang et al. (2008) is introduced in Section 2.3.

[1]http://nlp.ist.i.kyoto-u.ac.jp/EN/index.php?JUMAN
[2]http://code.google.com/p/mecab
[3]http://www.phontron.com/kytea

2.1 Word lattice

A word lattice, or lattice for short, is a data representation that compactly encodes an exponentially large number of word segmentations and POS tagging results (Kudo et al., 2004; Jiang et al., 2008).

An example lattice is illustrated in Figure 1. A lattice is formally a directed acyclic graph. A node (a circle in Figure 1) corresponds to the position between two characters, representing a possible word boundary. Moreover, two special nodes, b and e, represent the beginning and ending of the sentence. An edge (an arrow) represents a word-POS pair (w, t), where w is a word defined by two nodes, and t is a member of the predefined POS tag set.

Since every path from node b to e represents one candidate analysis of the sentence, the task of joint word segmentation and POS tagging can be seen as locating the most probable path amongst those in the lattice. Dynamic programming is usually used to locate the optimal path.

For later convenience, notations that will be used throughout this paper are introduced as follows. x and y are used to denote an input sentence and a lattice path. It is presumed that sentence x has n characters, and c_i is used to denote the i-th character ($1 \leq i \leq n$). w and t are used to denote a word and a POS tag, respectively.

2.2 Lattice reranking

Lattice reranking is an approximate inference technique for joint word segmentation and POS tagging (Jiang et al., 2008). In this approach, a small lattice is generated for an input sentence, and the paths of the lattice are then reranked to obtain the optimal one. The advantage of this approach is that the search space is greatly reduced in the same manner as conventional list-based reranking (Collins, 2000), while an exponentially large num-

ber of candidates is maintained in the lattice (Jiang et al., 2008).

In this framework, the task of joint word segmentation and POS tagging can be formalized as

$$\hat{y} = \arg\max_{y \in L(x)} \textsc{Score}(x, y) \quad (1)$$

where $\hat{y}$ is the optimal path, $L(x)$ is the lattice created for sentence x, and $\textsc{Score}(x, y)$ is a function for scoring path y of lattice $L(x)$. For notational convenience, lattice $L(x)$ is treated as a set of paths.

In this paper we explore the algorithm for generating the lattice $L(x)$. A naive approach requires $O(n^2)$ time to determine which word candidate to include in $L(x)$, as mentioned in Section 1, and constitutes a bottleneck. Although additional time is required to perform the $\arg\max$ operation, it is practically negligible because the lattice generated in this framework is generally small.

2.3 Pruning-based algorithm

Jiang et al. (2008) proposed a pruning-based lattice generation algorithm for reranking. Here, we briefly describe their algorithm. Interested readers may refer to (Jiang et al., 2008) for its details.

The pruning-based algorithm generates a lattice, specifically the edge set E constituting a lattice, by considering each character in a left-to-right fashion (Algorithm 1). The algorithm enumerates word-POS pairs (w, t), or edges, that end with the current character, c_i, and stores them in the candidate list, C (line 5-10). Top-scored k edges in C are then moved to E (line 11). Note that the word length l is limited to, at most, K characters (line 5).

This algorithm can be understood as pruning $O(n^2)$ candidate space by setting threshold K on the maximum word length. Although this method is much more efficient than exhaustively searching over the entire candidates, it still incurs non-negligible computational overhead, as we will demonstrate in the experiments.

An additional issue involving the pruning-based algorithm is how to determine the value of K. Although a smaller value of K reduces computational cost more, it is prone to remove more correct word-POS pairs from the search space. While this trade-off was not investigated by Jiang et al. (2008), it is examined in our experiment (see Section 5).

Algorithm 1 Pruning-based lattice generation algorithm.

```
1: T ← a set of all POS tags
2: E ← ∅
3: for i = 1 ... n do
4:    C ← ∅
5:    for l = 1 ... min(i, K) do
6:       w ← c_{i−l+1} c_{i−l+2} ... c_i
7:       for t ∈ T do
8:          C ← C ∪ (w, t)
9:       end for
10:   end for
11:   add top-k edges in C to E.
12: end for
13: return E
```

Algorithm 2 Pipelined lattice generation algorithm.

```
1: E ← ∅
2: W ← WordGenerator(x)
3: for w ∈ W do
4:    T ← PosTagGenerator(x, w)
5:    for t ∈ T do
6:       E ← E ∪ (w, t)
7:    end for
8: end for
9: return E
```

3 Pipeline Algorithm

As an alternative to the pruning-based algorithm, a pipelined lattice generation algorithm, which generates words and POS tags independently, is proposed here. In a nutshell, this method first generates the word set W constituting the lattice (Algorithm 2 line 2), and it then generates POS tags for each of the words (line 4).

The advantage of this approach is that it can naturally avoid searching the $O(n^2)$ candidate space by exploiting a character-based word segmentation model (Xue, 2003; Peng et al., 2004; Neubig et al., 2011) to obtain the word set W. This algorithm has linear-time complexity in the sentence length and hence is efficient.

This section proceeds as follows. Sections 3.1 and 3.2 describe how to generate words and POS tags, respectively. The computational complexity is then examined in Section 3.3.

3.1 Word generation

The character-based word segmentation model (Xue, 2003; Peng et al., 2004; Neubig et al., 2011) is used to generate word set W (Figure 2 line 2). This model performs segmentation by assigning tag sequence $\mathbf{b}$ to the input sentence:

$$\mathbf{b} = \arg\max_{\mathbf{b}} \mathbf{\Lambda}_w \cdot \mathbf{F}_w(x, \mathbf{b})$$

Name	Template
Char. n-gram	$\langle c_{i-1}, b_i\rangle$, $\langle c_i, b_i\rangle$, $\langle c_{i+1}, b_i\rangle$, $\langle c_{i-2}, c_{i-1}, b_i\rangle$, $\langle c_{i-1}, c_i, b_i\rangle$, $\langle c_i, c_{i+1}, b_i\rangle$, $\langle c_{i+1}, c_{i+2}, b_i\rangle$, $\langle c_{i-3}, c_{i-2}, c_{i-1}, b_i\rangle$, $\langle c_{i-2}, c_{i-1}, c_i, b_i\rangle$, $\langle c_{i-1}, c_i, c_{i+1}, b_i\rangle$ $\langle c_i, c_{i+1}, c_{i+2}, b_i\rangle$, $\langle c_{i+1}, c_{i+2}, c_{i+3}, b_i\rangle$
Char. type n-gram	$\langle c'_{i-1}, b_i\rangle$, $\langle c'_i, b_i\rangle$, $\langle c'_{i+1}, b_i\rangle$, $\langle c'_{i-2}, c'_{i-1}, b_i\rangle$, $\langle c'_{i-1}, c'_i, b_i\rangle$, $\langle c'_i, c'_{i+1}, b_i\rangle$, $\langle c'_{i+1}, c'_{i+2}, b_i\rangle$, $\langle c'_{i-3}, c'_{i-2}, c'_{i-1}, b_i\rangle$, $\langle c'_{i-2}, c'_{i-1}, c'_i, b_i\rangle$, $\langle c'_{i-1}, c'_i, c'_{i+1}, b_i\rangle$, $\langle c'_i, c'_{i+1}, c'_{i+2}, b_i\rangle$, $\langle c'_{i+1}, c'_{i+2}, c'_{i+3}, b_i\rangle$
Dictionary	$\langle$BEGIN, $b_i\rangle$,$\langle$END, $b_i\rangle$, $\langle$INSIDE, $b_i\rangle$, $\langle$BEGIN, $s, b_i\rangle$, $\langle$END, $s, b_i\rangle$, $\langle$INSIDE, $s, b_i\rangle$

Table 1: Feature templates of word generation. c_i and c'_i represent the target character and its type, respectively. c'_i specifically takes one of the following values: (1) Roman alphabet, (2) Chinese *kanji* characters, (3) Japanese *hiragana* characters, (4) Japanese *katakana* characters, (5) numerical symbols, or (6) others. The neighboring characters and their types are similarly referred to as c_{i-1}, c_{i+1}, c'_{i+1}, and so on. b_i is the tag (B or I) given to the target character. BEGIN and END represent whether a word in a dictionary begins with or ends before the target character, respectively. INSIDE means that the target character is inside the word. s denotes the length (1, 2, 3, 4, or 5$\leq$) of the word registered in the dictionary.

Name	Template
Word	$\langle w, t\rangle$
Word length	$\langle$LENGTH$(w), t\rangle$
Affix	$\langle c_i, t\rangle$, $\langle c_i, c_{i+1}, t\rangle$, $\langle c_{j-1}, t\rangle$, $\langle c_{j-2}, c_{j-1}, t\rangle$
Neighboring string	$\langle c_{i-1}, t\rangle$, $\langle c_{i-2}, c_{i-1}, t\rangle$, $\langle c_{i-3}, c_{i-2}, c_{i-1}, t\rangle$, $\langle c_j, t\rangle$, $\langle c_j, c_{j+1}, t\rangle$, $\langle c_j, c_{j+1}, c_{j+2}, t\rangle$
Dictionary	$\langle$DICT$(w, t)\rangle$, $\langle$DICT$(w, t), t\rangle$

Table 2: Feature templates of POS tag generation. $w = c_i c_{i+1} \dots c_{j-1}$ represents the word string, and t represents the target POS tag. LENGTH(w) returns the length of the word w in the number of characters: 1, 2, 3, 4, or 5$\leq$. DICT(w, t) is an indicator representing that word w with POS tag t is registered in a dictionary. The features in the last row are fired only when the target word is found in a dictionary.

where $\mathbf{b} = b_1 \dots b_n$ is the character-based tag sequence that encodes the segmentation results; $b_i = B$ and $b_i = I$ represent whether the i-th character is the beginning or inside of a word, respectively. $\mathbf{\Lambda}_w$ and $\mathbf{F}_w(x, \mathbf{b})$ are weight and feature vectors, respectively.

The model is trained with the averaged structured perceptron (Collins, 2002) due to its simplicity and efficiency. The features illustrated in Table 1, as well as tag bigrams, were used for the training. The features in Table 1 is basically taken from (Neubig et al., 2011). The first two rows represent character strings surrounding the target character; the last row represents dictionary-based features similar to those described in (Neubig et al., 2011). The dictionary-based features are fired if a string in a sentence is registered as a word in a dictionary, and they encode whether the string begins with or ends before the target character, or includes the target character.

α-best outputs of this segmentation model are used to obtain word set W:

$$W = \cup_{i=1\dots\alpha} W_i$$

where W_i is a word set included in the i-th best output. Hyperparameter α controls the size of word set $|W|$ and is tuned by using development data.

3.2 POS tag generation

To generate POS tags for each word (Figure 2 line 4), a linear model was used. Given sentence x and word w, it assigns the following score to each POS tag t (Neubig et al., 2011):

$$\mathbf{\Lambda}_t \cdot \mathbf{F}_t(x, w, t)$$

where $\mathbf{\Lambda}_t$ and $\mathbf{F}_t(x, w, t)$ are weight and feature vectors, respectively. Averaged perceptron was used for training (Freund and Schapire, 1999).

Table 2 shows the feature templates. Word string, word length, prefixes and suffixes up to length two were used, and the adjacent strings of the word up to length three were used. We also check the presence of the word in a dictionary.

For each word, top-β tags were used as the POS tag set T (line 4). Hyperparameter β is also tuned by using development data.

3.3 Computational complexity

Unlike the pruning-based algorithm, the pipeline algorithm can generate words of arbitrary lengths. Nevertheless, it still only needs $O(n)$ time. This

can be proved as follows. First, the word segmentation model takes $O(n)$ time to output word set W, since this step can be efficiently performed by dynamic programming. In addition, since $O(|W|) = O(n)$, the outer loop of the algorithm requires $O(n)$ time. This can be verified as

$$|W| = |\cup_{i=1...\alpha} W_i| \leq \sum_{i=1...\alpha} |W_i| \leq \alpha\, n$$

where $|W_i| \leq n$. Since the process in lines 4-7 is independent of n, the pipeline algorithm requires $O(n)$ time.

It also follows from the above discussion that the lattice size, that is, the number of edges, is also linear in the sentence length, i.e., $O(|E|) = O(n)$. Consequently, since the node degree is at most α (i.e., not dependent on n), the lattice path can be efficiently reranked in $O(n)$ time by using dynamic programming.

4 Perceptron-based Reranker

This section presents our reranker. Since the main focus of this study is in not reranking but lattice generation, a perceptron-based reranker was developed by simply following the procedure proposed by (Huang, 2008).

The scoring function $\text{SCORE}(x, y)$ in equation (1) is defined as follows:

$$\begin{aligned} \hat{y} &= \arg\max_{y \in L(x)} \text{SCORE}(x, y) \\ &= \arg\max_{y \in L(x)} \mathbf{\Lambda} \cdot \mathbf{F}(x, y) \end{aligned}$$

where $\mathbf{\Lambda}$ is the weight vector and $\mathbf{F}(x, y)$ is the feature vector.

4.1 Training

The averaged perceptron algorithm was used to train weight vector $\mathbf{\Lambda}$ (Huang, 2008). Note here two minor technical issues that have to be addressed before the perceptron algorithm can be used for training the reranker.

First, the generated lattice $L(x)$ might not include the oracle path. This possibility is avoided by simply adding all the nodes and edges in the oracle lattice to $L(x)$. This approach worked reasonably well in our experiments, while having the advantage of being simpler than the alternative (Huang, 2008; Jiang et al., 2008).

Second, the same data should not be used for training the lattice generator (i.e., the two models described in Sections 3.1 and 3.2) and reranker. If the same data were used, we will end up using injuriously better lattices when training the reranker than testing. To meet this requirement, the training data were split into ten subsets. During training of the reranker, the lattices of each subset were provided by the lattice generator trained by using the remaining nine subsets. During testing, on the other hand, the lattice generator trained by using the entire training data was used.

4.2 Features

The features used for training the reranker include those listed in Table 1 and Table 2, as well as POS tag bigrams. For the features in Table 1, BIES encoding (Nakagawa, 2004) is used. Since all those features can be factorized, the optimal path is located by using dynamic programming.

5 Experiment

The effectiveness of the lattice generation algorithm was investigated in the experiment described in the following. Sections 5.1, 5.2, and 5.3 explain our experimental setting: data sets, lattice generation algorithms to be compared, and hyperparameter tuning. The experimental results are reported in Section 5.4. The experiments were performed on a computer with 3.2 GHz Intel® Xeon™ CPU and 32 GB memory.

5.1 Data sets

Three evaluation data sets were developed from three corpora: Kyoto Corpus (KC) version 4.0 (Kurohashi and Nagao, 1998), Kyoto university NTT Blog Corpus (KNBC) version 1.0 (Hashimoto et al., 2011), and Balanced Corpus of Contemporary Written Japanese (BCCWJ) (Maekawa, 2008). Each corpus was randomly split into three parts: training, development, and test set. The size of each data set is listed in Table 3.

JUMAN dictionary version 7.0[4] was used to extract the dictionary-based features in the experiments using KC and KNBC. Because BCCWJ adopts word segmentation criteria and a POS tag set different from those of the other two corpora, a different dictionary, UniDic version 1.3.12[5], was used in the experiment using BCCWJ.

[4] http://nlp.ist.i.kyoto-u.ac.jp/EN/index.php?NLPresources
[5] http://www.tokuteicorpus.jp/dist

	KC					KNBC					BCCWJ				
	Time		#Cand.	F_1	Size	Time		#Cand.	F_1	Size	Time		#Cand.	F_1	Size
Pruning (K=5)	20	22	212	†97.25	356	1.2	1.3	137	†92.72	235	25	26	163	†97.33	276
Pruning (K=10)	31	32	400	97.92	88.9	2.0	2.1	250	93.48	235	43	44	301	†98.08	69.0
Pruning (K=20)	62	63	702	**97.94**	88.9	3.6	3.8	413	93.42	235	88	90	516	**98.18**	69.0
Pipeline	**1.8**	**2.6**	**30.4**	**97.94**	**60.8**	**0.12**	**0.18**	**24.8**	**93.92**	**99.2**	**2.3**	**3.1**	**23.3**	98.10	**46.6**

Table 4: Comparison of the reranking systems with the different lattice generation algorithms. Best-performing results in each metric are highlighted in bold font.

	Training	Development	Testing
KC	30,608	4028	3764
KNBC	3453	385	348
BCCWJ	47,547	6144	5741

Table 3: The number of sentences included in the three data sets.

5.2 Lattice generation algorithms

Two types of rerankers were implemented: one uses the pruning-based lattice generation algorithm, and the other uses the pipeline algorithm. All the rerankers were trained in the same manner as described in Section 4.

Although Jiang et al. (2008) fixed pruning threshold K as 20, $K \in \{5, 10, 20\}$ was tested to examine the effect of this parameter. As a result, three rerankers that use the pruning-based algorithm were thus created.

The pruning-based algorithm uses a character-based model[6] to obtain top-k edges (Figure 1 line 11). Although Jiang et al. (2008) proposed several features to train this model, they are simplistic compared with those used in the pipeline algorithm (i.e., Table 1 and 2). To make the comparison as fair as possible, the feature listed in Table 1 and BIES encoding were used (c.f., Section 4.2) were used. The features listed in Table 2 were not used, because they are not usable in a character-based model. It is considered that this feature set is comparable with that used by the pipeline algorithm, because the reranker using the pruning-based algorithm achieved comparable F_1-score with the one using the pipeline algorithm when K is large (see Section 5.4).

5.3 Hyperparameter tuning

Hyperparameter k of the pruning-based algorithm was tuned with the development data. The tuning was done by searching over $\{1, 2, 4, 8, 16, \ldots, 256\}$ and selecting k that generated the lattice with the fewest edges amongst those covering at least θ% of the correct edges.

Since the pipeline algorithm also has hyperparameters (α, β), the hyperparameters were tuned in a similar manner by performing a grid search over $\{1, 2, 4, 8, 16, \ldots, 256\} \times \{1, 2, 4, 8, 16, \ldots, 256\}$.

The value of θ was set as 99, 97, and 99 for the three data sets, respectively. A smaller value of θ was used for KNBC because over 99% coverage could not be achieved in this data set.

5.4 Results

Table 4 summarizes the time in seconds spent on lattice generation, overall processing time spent on reranking, average number of candidates per sentence (see below), word-level F1- score in the joint task, and average lattice size per sentence, where lattice size refers to the number of edges in a lattice.

As for the pruning-based algorithm, the number of candidates refers to the number of words to be considered (Figure 1 line 6). As for the pipeline algorithm, it refers to the size of word set W (Figure 2). This number serves as an estimation of the computational cost. Notice that it corresponds to the time consumed by the two outer loops in Figure 1 or by the outer loop in Figure 2.

The symbol † is used to represent that the difference in F_1-score from the best-performing system is statistically significant ($p < 0.01$). Bootstrap resampling with 1,000 samples was used to test the statistical significance.

5.4.1 Runtime

Table 4 reveals that the reranking system using the pruning-based algorithm consumes the vast majority of the time for lattice generation. In other words, the pruning-based algorithm is not efficient enough. This inefficiency was not pointed out in previous studies, e.g., (Zhang and Clark, 2010; Sun, 2011).

[6]Not detailed this model in this paper; refer to (Jiang et al., 2008) for details.

The results in Table 4 also demonstrate that the reranker using the pipeline algorithm is an order of magnitude faster than the pruning-based algorithms. It is significantly faster than even the case that $K = 5$. This result indicates the importance of using an efficient lattice generation algorithm in the reranking system.

Table 4 also indicates that the number of the candidates roughly correlates with the actual computation time spent on lattice generation. This correlation confirms that the speed-up is achieved mainly by reducing the number of word candidates to be considered.

5.4.2 F_1-score

F_1-score of the reranking systems was investigated next. The pipeline algorithm achieved comparable or higher F_1-score than the pruning-based algorithm. This result shows that the speed-up does not come at the cost of accuracy.

It is crucial for the pruning-based algorithm to select an appropriate threshold value, K. If the value is too small, F_1-score will significantly drop. In case that $K = 5$, F_1-score was statistically significantly worse than that attained by the best-performing system for all three data sets ($p < 0.01$). On the other hand, an excessively large value ($K = 20$) does not contribute to the increase of F_1-score so much, while it considerably degrades the speed.

5.4.3 Lattice size

Table 4 shows that the pipeline algorithm usually generates smaller lattices than the pruning-based algorithm. This is because the pruning-based algorithm has no mechanisms to prune nodes (Jiang et al., 2008). To be more specific, the pruning-based algorithm always produces $n + 1$ nodes for a sentence with n characters; hence, the lattice size is prone to grow large. The pipeline algorithm is, on the other hand, free from such a problem.

The coverage of the correct edges as the function of the average lattice size was investigated as follows (Figure 2). For the pruning-based algorithm, which has only one hyperparameter, k, the graph was drawn by changing k over $\{1, 2, 4, 8, 16\}$. Note that the graph for $K = 10$ is omitted, because almost the same lattices are generated for $K = 10$ and $K = 20$. For the pipeline algorithm, $\alpha = 32$ is fixed and β is changed over $\{1, 2, 4, 8, 16\}$ to draw the two-dimensional graphs. It is clear that the lattice generated by the pipeline algorithm generally achieves higher coverage, while having a smaller number of edges than the pruning-based algorithm.

As discussed in Section 3.3, the size of word set $|W|$ is linear in the sentence length. This analysis empirically justified as follows. The number of words is illustrated in Figure 3 as a function of sentence length. The three graphs in the figure clearly illustrate that the number of words grows linearly with increasing sentence length.

	KC	KNBC	BCCWJ
JUMAN	†95.37	93.85	N/A
MeCab	†95.45	†91.60	†96.31
Kytea	†96.95	†90.91	†97.10
Our reranker	**97.94**	**93.92**	**98.10**

Table 5: Comparison of F_1-score with that achieved by the existing software.

6 Comparison with Existing Software

As an additional experiment, the proposed pipeline-algorithm-based reranking system was compared with three software tools popular in Japanese NLP: JUMAN, MeCab (Kudo et al., 2004), and Kytea (Neubig et al., 2011).

Table 5 compares the F_1-score of the proposed system with that attained by the three tools. Bootstrap resampling with 1,000 samples was used for the statistical significance test. The symbol † indicates that the F_1-score is significantly lower than that achieved by the proposed system ($p < 0.01$). It is clear that the proposed system outperforms the existing tools in the case of two of the three data sets, while performing comparably with JUMAN in the case of KNBC. Note that JUMAN is a rule-based system and is not applicable to BCCWJ because of the discrepancy in the definition of the segmentation criteria and POS tag set.

The speeds of the algorithms were also investigated. The proposed system processed 1400 sentences in a second, while JUMAN, MeCab, and Kytea processed 2100, 29000, and 3200 sentences, respectively. This result demonstrates that the proposed reranking system using the pipeline algorithm successfully achieved speed close to the two of the three tools, while keeping considerably higher F_1-score.

7 Related Work

Several methods, other than the pruning-based algorithm (Jiang et al., 2008), have been developed

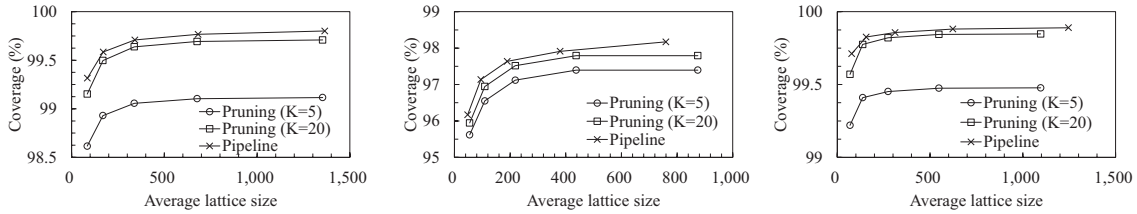

Figure 2: Coverage as the function of average lattice size (left: KC; middle: KNBC; right: BCCWJ).

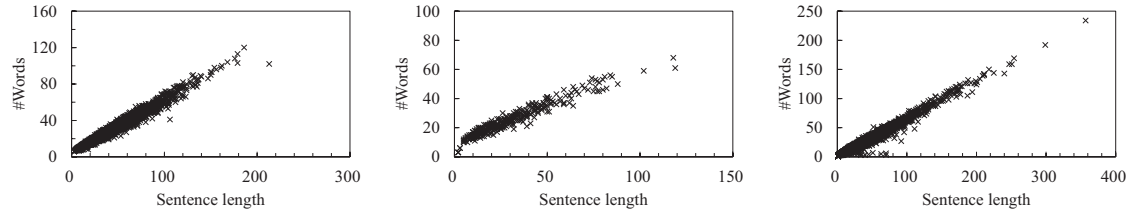

Figure 3: Number of words as a function of sentence length (left: KC; middle: KNBC; right: BCCWJ).

for lattice generation. However, they are dependent on an external dictionary and have limitations in handling OOV words. For example, Kudo et al. (2004) built a lattice based on dictionary-lookup. While efficient, such a method is prone to remove OOV words from a lattice and degrade accuracy (Uchimoto et al., 2001). Other researchers (Nakagawa and Uchimoto, 2007; Kruengkrai et al., 2009) used a word-character hybrid model, which combines dictionary-lookup and character-based modeling of OOV words. This method still has difficulty in using word-level information of OOV words.

The techniques utilized by the pipelined lattice generation algorithm have also been used elsewhere (Sassano, 2002; Peng et al., 2004; Shi and Wang, 2007; Neubig et al., 2011; Wang et al., 2011). However, the present study is the first to investigate the effectiveness of such a technique in the context of lattice reranking. Empirical studies similar to the ones made in this study are not found in the other work.

Zhang and Clark (2008) and Zhang and Clark (2010) proposed a fast decoding algorithm for joint word segmentation and POS tagging. The present study is largely complementary with theirs, since it did not investigate to improve decoding algorithm. Their algorithm should be useful for the decoding of our reranker especially when dynamic programming is not effective; for example, nonlocal features are used.

8 Conclusion

The effectiveness of the lattice generation algorithms used in joint word segmentation and POS tagging was investigated. While lattice generation has not been paid much attention to in previous studies, the present study demonstrated that the design of a lattice generation algorithm has a significant impact on the performance of a reranking system. It was showed that the simple pipeline algorithm outperforms the pruning-based algorithm. We hope that the pipeline algorithm serves as a simple but effective building block of future researches.

Acknowledgments

This work was supported by the FIRST program. The authors thank the anonymous reviewers for their helpful comments.

References

Masayuki Asahara and Yuji Matsumoto. 2000. Extened models and tools for high-performance part-of-speech tagger. In *Proceedings of COLING*, pages 21–27.

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proceedings of ICML*, pages 175–182.

Michael Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*, pages 1–8.

Yoav Freund and Robert E. Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296.

Chikara Hashimoto, Sadao Kurohashi, Daisuke Kawahara, Keiji Shinzato, and Masaaki Nagata. 2011. Construction of a blog corpus with syntactic, anaphoric, and semantic annotations (in Japanese). *Journal of Natural Language Processing*, 18(2):175–201.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of ACL*, pages 586–594.

Wenbin Jiang, Haitao Mi, and Qun Liu. 2008. Word lattice reranking for Chinese word segmentation and part-of-speech tagging. In *Proceedings of Coling*, pages 385–392.

Canasai Kruengkrai, Virach Sornlertlamvnich, and Hitoshi Isahara. 2006. A conditional random field framework for Thai morphological analysis. In *Proceedings of LREC*, pages 2419–2424.

Canasai Kruengkrai, Kiyoaki Uchimoto, Jun'ichi Kazama, Yiou Wang, Kentaro Torisawa, and Hitoshi Isahara. 2009. An error-driven word-character hybrid model for joint Chinese word segmentation and POS tagging. In *Proceedings of ACL*, pages 513–521.

Taku Kudo, Kaoru Yamamoto, and Yuji Matsumoto. 2004. Applying conditional random fields to Japanese morphological analysis. In *Proceedings of EMNLP*, pages 230–237.

Sadao Kurohashi and Makoto Nagao. 1998. Building a Japanese parsed corpus while improving the parsing system. In *Proceedings of LREC*, pages 719–724.

Kikuo Maekawa. 2008. Balanced corpus of contemporary written Japanese. In *Proceedings of the 6th Workshop on Asian Language Resources*, pages 101–102.

Tetsuji Nakagawa and Kiyoaki Uchimoto. 2007. A hybrid approach to word segmentation and POS tagging. In *Proceedings of ACL, Demo and Poster Sessions*, pages 217–220.

Tetsuji Nakagawa. 2004. Chinese and Japanese word segmentation using word-level and character-level information. In *Proceedings of Coling*, pages 466–472.

Graham Neubig, Yousuke Nakata, and Shinsuke Mori. 2011. Pointwise prediction for robust adaptable Japanese morphological analysis. In *Proceedings of ACL*, pages 529–533.

Fuchun Peng, Fangfang Feng, and Andrew McCallum. 2004. Chinese segmentation and new word detection using conditional random fields. In *Proceedings of Coling*, pages 562–568.

Manabu Sassano. 2002. An empirical study of active learning with support vector machines for Japanese word segmentation. In *Proceedings of ACL*, pages 505–512.

Yanxin Shi and Mengqiu Wang. 2007. A dual-layer CRFs based joint decoding method for cascaded segmentation and labeling tasks. In *Proceedings of IJCAI*, pages 1707–1712.

Weiwei Sun. 2011. A stacked sub-word model for joint Chinese word segmentation and part-of-speech tagging. In *Proceedings of ACL*, pages 1385–1394.

Kiyotaka Uchimoto, Satoshi Sekine, and Hitoshi Isahara. 2001. The unknown word problem: a morphological analysis of Japanese using maximum entropy aided by a dictionary. In *Proceedings of EMNLP*, pages 91–99.

Yiou Wang, Jun'ichi Kazama, Yoshimasa Tsuruoka, Wenliang Chen, Yujie Zhang, and Kentaro Torisawa. 2011. Improving Chinese word segmentation and POS tagging with semi-supervised methods using large auto-analyzed data. In *Proceedings of IJCNLP*, pages 309–317.

Nianwen Xue. 2003. Chinese word segmentation as character tagging. *Computatioinal Linguistics and Chinese Language Processing*, 8(1):29–48.

Yue Zhang and Stephen Clark. 2008. Joint word segmentation and POS tagging using a single perceptron. In *Proceedings of ACL*, pages 888–896.

Yue Zhang and Stephen Clark. 2010. A fast decoder for joint word segmentation and POS tagging using a single discriminative model. In *Proceedings of EMNLP*, pages 843–852.