A Two-Pass Search Algorithm for Thai Morphological Analysis

Canasai Kruengkrai and Hitoshi Isahara

Graduate School of Engineering, Kobe University
1-1 Rokkodai-cho, Nada-ku, Kobe 657-8501 Japan
National Institute of Information and Communications Technology
3-5 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0289 Japan
{canasai,isahara}@nict.go.jp

Abstract. Considering Thai morphological analysis as a search problem, the approach is to search the most likely path out of all candidate paths in the word lattice. However, the search space may not contain all possible word hypotheses due to the unknown word problem. This paper describes an efficient algorithm called the *two-pass search algorithm* that first recovers missing word hypotheses, and then searches the most likely path in the expanded search space. Experimental results show that the two-pass search algorithm improves the performance of the standard search by 3.23 F_1 in word segmentation and 2.92 F_1 in the combination of word segmentation and POS tagging.

1 Introduction

Morphological analysis has been recognized as a fundamental process in Thai text analysis. In Thai, words are written continuously without word boundaries like other non-segmenting languages (e.g., Chinese and Japanese). However, the Thai writing system has certain unique characteristics. For example, in order to form a smallest linguistic unit, a character cluster, including a consonant, a vowel, and/or a tonal mark, must be formed.

Thai morphological analysis generally involves two tasks: segmenting a character string into meaningful words, and assigning words with the most likely part-of-speech (POS) tags. Considering Thai morphological analysis as a search problem, the approach is to search the most likely path out of all candidate paths in a word lattice. Figure 1 illustrates the word lattice of a string analysis, consisting of possible word hypotheses and their connections. The path indicated by the bold line is the correct segmentation.

Discriminating such path from other candidate paths is a difficult problem itself, and requires a dynamic programming search (e.g., the Viterbi algorithm). The problem becomes more difficult, since the search space is often imperfect. Some word hypotheses are missing due to the unknown word problem. Unknown words are words that do not occur in the system dictionary or the training corpus. The system has no knowledge to use in generating hypotheses for unknown words.

Received 15/10/07
Accepted 07/12/07
Final Version 21/01/08

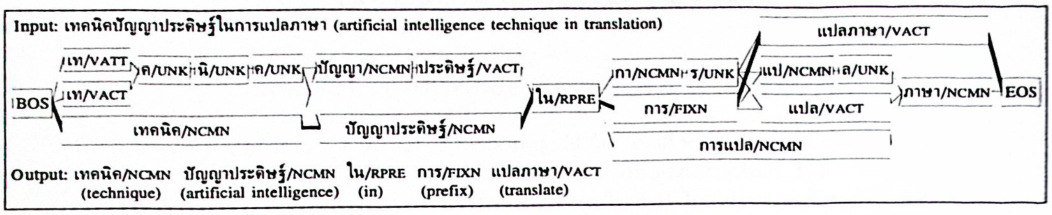

Input: เทคนิคปัญญาประดิษฐ์ในการแปลภาษา (artificial intelligence technique in translation)
แปลภาษา/VACT
เท/VATT
ค/UNK
นิ/UNK
ค/UNK
ปัญญา/NCMN
ประดิษฐ์/VACT
กา/NCMN
ร/UNK
แป/NCMN
ล/UNK
BOS
เท/VACT
ใน/RPRE
การ/FIXN
แปล/VACT
ภาษา/NCMN
EOS
เทคนิค/NCMN
ปัญญาประดิษฐ์/NCMN
การแปล/NCMN
Output: เทคนิค/NCMN ปัญญาประดิษฐ์/NCMN ใน/RPRE การ/FIXN แปลภาษา/VACT
(technique) (artificial intelligence) (in) (prefix) (translate)

Fig. 1. Lattice produced from an unsegmented string

In this paper, we propose a new statistical model for Thai morphological analysis that jointly tackles word segmentation, POS tagging, and unknown word problems. Our assumption is that the system could analyze a given string accurately (both known and unknown words), if the search space contains reasonably potential word hypotheses. Such space can be generated by using the so-called *two-pass search algorithm*. In the first pass, we use a dictionary and writing rules to build an initial lattice for a string input. We then identify suspicious word hypotheses in the lattice and expand new word hypotheses from all possible substrings within uncertainty ranges. Finally, in the second pass, we search the optimal path in the expanded search space by applying a lattice-based Viterbi algorithm. Experimental results show that the two-pass search algorithm improves the performance of the standard search by 3.23 F_1 in word segmentation and 2.92 F_1 in the combination of word segmentation and POS tagging.

2 Framework for Thai Morphological Analysis

2.1 Problem Formulation

We formulate the problem of Thai morphological analysis on word lattice representation. A lattice is an ordered graph that efficiently represents rich information about sequences of word hypotheses. Nodes in the graph are word hypotheses with their morphological information (e.g., POS tags and character types), and arcs are transitions between word hypotheses.

Given an unsegmented string, the lattice is produced by expanding possible word hypotheses from left to right. If a sequence of characters can be matched with a word surface in the system dictionary, word hypotheses are then generated corresponding to lexical entries. If no word can be found in the system dictionary, Thai writing rules are applied to generate character clusters. These character clusters are assigned with a set of open-class POS tags (e.g., noun and verb) to be candidate word hypotheses.

Based on word lattice representation, we define the process of Thai morphological analysis as a search problem by the following equation:

$$y^* = \text{argmax}_{y \in \mathcal{Y}(x)} p(y|x) = \text{argmax}_{y \in \mathcal{Y}(x)} p(y) \ , \tag{1}$$

where x is a character sequence $c_1 c_2 \ldots c_{|x|}$, y is a sequence of word hypotheses, and $\mathrm{Y}(x)$ is a lattice containing a set of whole-sentence analyses for x. Note that we can drop $\boldsymbol{x}$ in the last step, since it is a fixed character sequence and does not affect the argmax. Our objective is to search the most likely path $\boldsymbol{y}^*$ in the lattice $\mathrm{Y}(\boldsymbol{x})$.

2.2 Discriminative Learning

In our model, we apply discriminative learning to estimate model parameters. The idea is to learn how to discriminate the correct path from other candidate paths with features. This is in contrast to most previous research for Thai morphological analysis, which estimates model parameters using only the correct path in a given training corpus. Discriminate learning estimates model parameters based on the search space that the system will expect to see in practice. Our problem formulation can be advantageously used for discriminative learning, since the search space is formed in the word lattice. Both learning and decoding are processed in the same problem structure.

We now factorize the probability $p(\boldsymbol{y})$ further by using an instance of the log-linear models with bigram features [7]. Thus, we obtain:

$$p(\boldsymbol{y}) = \frac{1}{Z} \exp\left\{ \sum_{t=1}^{\#\boldsymbol{y}} \sum_{k=1}^{K} \lambda_k f_k(y_{t-1}, y_t) \right\} , \tag{2}$$

where $\#\boldsymbol{y}$ is a number of word hypotheses in a path $\boldsymbol{y}$ which varies among candidate paths, f_k is a feature operated on bigram transition between y_{t-1} and y_t, λ_k is a parameter to be estimated. To ensure that $\sum_{\boldsymbol{y}\in\mathcal{Y}(\boldsymbol{x})} p(\boldsymbol{y}) = 1$, the normalization factor Z is given by:

$$Z = \sum_{\boldsymbol{y}'\in\mathcal{Y}(\boldsymbol{x})} \exp\left\{ \sum_{t=1}^{\#\boldsymbol{y}'} \sum_{k=1}^{K} \lambda_k f_k(y'_{t-1}, y'_t) \right\} , \tag{3}$$

which is a sum over all paths $\boldsymbol{y}'$ in the lattice.

In learning, given a set of training data $\mathrm{D} = \{\boldsymbol{y}^{(i)}\}_{i=1}^{N}$ where N is the number of all training samples, we need to find a set of feature weights $\boldsymbol{\lambda}$. We use a penalized log-likelihood maximization, in which the objective function defines:

$$\log p(\boldsymbol{\lambda} | \mathrm{D}) = \sum_{i=1}^{N} \log p_{\boldsymbol{\lambda}}(\boldsymbol{y}^{(i)}) - \sum_{k=1}^{K} \frac{\lambda_k^2}{2\sigma^2} . \tag{4}$$

One can use a numerical method for unconstrained optimization to solve Equation 4. In our implementation, we apply the L-BFGS algorithm [8], which is known to be a very efficient optimization method used in large NLP tasks.

To summarize, the system learns the model parameters by representing whole training samples with lattices, and extracting the global feature space. It then iterates on each lattice to estimate feature weights using the Viterbi-style algorithm (explained in the next section). The system tries to discriminate the correct path from incorrect paths using the current estimation of feature weights, and computes the expectation of feature weights based on resulting errors. The system proceeds until it converges to a point where the change of the objective function is less than a given threshold.

2.3 Decoding

In decoding, we need to find the most likely path such that $\boldsymbol{y}^* = \text{argmax}_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})} p(\boldsymbol{y})$ To avoid an exponential-time search over all candidate paths in the lattice $\mathrm{Y}(\boldsymbol{x})$, we thus adapt the Viterbi algorithm to the lattice structure. Let $\mathrm{L}(y)$ be a set of nodes that connect to node y from the left. The Viterbi algorithm stores a partial probability δ of the most likely path reaching node y, which can be expressed in a recursive form:

$$\delta_y = \max_{y' \in \mathcal{L}(y)} \left[\delta_{y'} \exp \left(\sum_{k=1}^{K} \lambda_k f_k(y', y) \right) \right] . \tag{5}$$

The partial best path is one that achieves the maximal probability. The recursion terminates at the end of string (eos):

$$\phi^*_{y_{\text{eos}}} = \text{argmax}_{y' \in \mathcal{L}(y_{\text{eos}})} \delta_{y'} . \tag{6}$$

Finally, we can backtrack through the lattice with the back pointer ϕ to recover the global best path $\boldsymbol{y}^*$.

3 Two-Pass Search Algorithm

As mentioned earlier, in practice, the search space is not complete like in learning where all words are known. Some word hypotheses are missing due to the unknown word problem. There are two possible approaches to handle this issue: increasing the coverage of the system dictionary, or using a model that can process unknown words. In this paper, we are interested in the latter approach. Our goal is to recover missing word hypotheses based on the learned model. Here we design the two-pass search algorithm that finds the most likely path in the expanded search space.

The two-pass search algorithm computes the marginal probability of each node in the lattice, which reflects the uncertainty of that node. Nodes that have their probabilities less than a given a threshold, denoted by ϵ, are considered as suspicious nodes. Thus, we can identify uncertainty ranges in the input string that might contain parts of known or unknown words depended on their probabilities. We generate all possible substrings (sequences of character clusters) within uncertainty ranges, and expand nodes in the lattice according to these substrings. We then search the most likely path in the new lattice that contains much better coverage of word hypotheses. Algorithm 1 provides an outline of the two-pass search algorithm.

Algorithm 1: TWO-PASS SEARCH ALGORITHM

```
input  : An unsegmented string $\boldsymbol{x}$.
output : The most likely path $\boldsymbol{y}^* = y_1, \ldots, y_{\#\boldsymbol{y}}$.
  • (1-pass) Build an initial lattice $\mathcal{Y}(\boldsymbol{x})$, and estimate the probability $p_y$ of each
    node $y$ in the lattice using Equations 7, 8 and 9.
  • Find nodes that $p_y < \epsilon$ or node-state = UNK, and mark their surface lengths.
  • Generate all possible substrings within uncertainty ranges, and expand new nodes
    in $\mathcal{Y}(\boldsymbol{x})$ according to these substrings.
  • (2-pass) Perform the Viterbi search to get $\boldsymbol{y}^*$ using Equations 5 and 6.
```

Let α_y be a forward probability and β_y be a backward probability reaching node y from the left and right, respectively. The marginal probability of a particular node y can be calculated by:

$$p_y = \frac{\alpha_y \cdot \beta_y}{Z} , \tag{7}$$

where

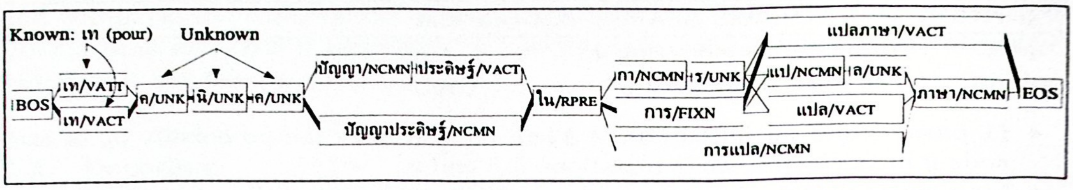
$$\alpha_y = \sum_{y' \in \mathcal{L}(y)} \left[\alpha_{y'} \exp \left(\sum_{k=1}^{K} \lambda_k f_k(y', y) \right) \right] , \tag{8}$$

$$\beta_y = \sum_{y' \in \mathcal{R}(y)} \left[\beta_{y'} \exp \left(\sum_{k=1}^{K} \lambda_k f_k(y, y') \right) \right] . \tag{9}$$

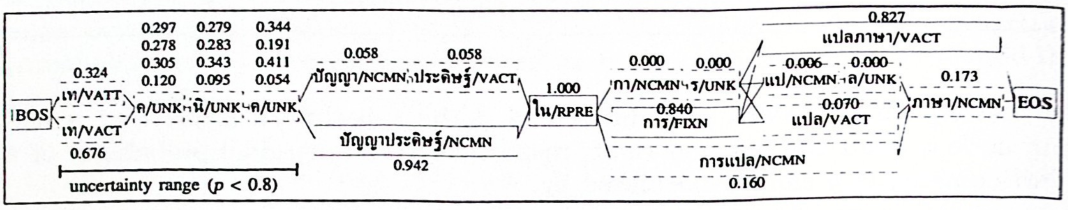
Note that the forward and backward probabilities are calculated with recursion similar to the Viterbi algorithm, except using sum function instead of max function. We can apply the Forward-Backward algorithm to obtain p_y for a given node y.

Let us describe the two-pass search algorithm more clearly with a motivating example. Recall the lattice in Figure 1, and suppose that the word 'technique' does not appear in the system dictionary. The system tries to build a lattice where the predicted path now contains both known and unknown nodes (Figure 2(a)). This situation often occurs in practice, when a word hypothesis is missing and the system searches the most likely path based on available information. If we stop at this point, we obtain an incorrect segmentation.

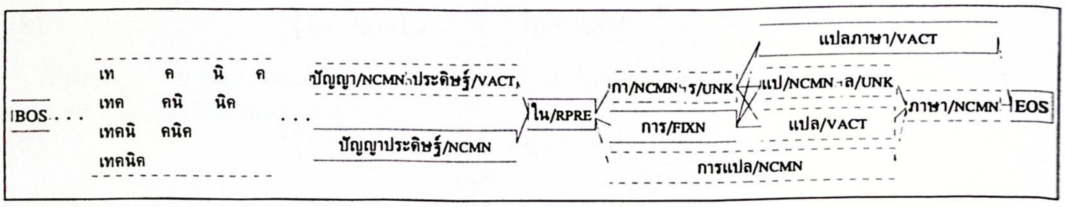
When the two-pass search algorithm is performed, the probability of each node in the lattice is calculated (Figure 2(b)). Note that nodes tagged with UNK show a list of probabilities, since they represent a number of candidate word hypotheses actually tagged with open-class POS tags. In this example, four tags are assigned, including NCMN, NPRP, VACT and VATT. The algorithm identifies the suspicious nodes that their probabilities are less than a threshold $\epsilon = 0.8$, and then all possible substrings within the uncertainty range are generated (Figure 2(c)). Finally, the algorithm can recover the correct path in the new search space (Figure 2(d)).



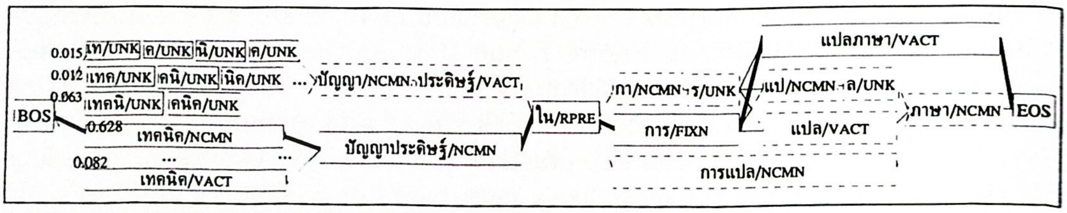
(a) Lattice when a word hypothesis is missing



(b) The suspicious node indicated by the dashed line box are identified, and the uncertainty range which has no potential nodes in the paths is located



(c) Possible substrings are generated within the uncertainty range



(d) Lattice with new candidate word hypotheses. The correct path can be recoverd in the expanded search space

Fig. 2. Examples show how the two-pass search algorithm proceeds

4 Experiments

4.1 Experimental Settings

We conducted our experiments on ORCHID corpus, which is a Thai part-of-speech tagged corpus [9]. The original texts were taken from technical papers published in the proceedings of the National Electronics and Computer Technology Center (NECTEC) annual conference. ORCHID has been known to be a hard dataset for evaluation, since it consists of a large number of unknown words. In experiments, we divided the corpus into the training and test sets according to Table 1. We converted all spacial tags for punctuation to original forms, e.g. , <colon> ‘:’. We also merged space tokens to their next tokens, and considered them as leading spaces of words. The system dictionary was generated from unique tokens occurred in the training set and TCL's computational lexicon [5].

Table 1. Details of dataset used in our experiments

corpus	ORCHID
POS tags	15 categories and 48 subcategories
# of training samples	9,234 (papers published in '89-'90)
# of training tokens	116,102
# of test samples	5,808 (papers published in '91)
# of test tokens	71,097
# of known test tokens	57,381
# of unknown test tokens	9,403 (not include punctuation)
# of entries in the system dictionary	55,791
# of features	2,203,932

Since our problem formulation is based on a log-linear model, a variety of features can be incorporated in the model. Table 2 provides the details of features used in our experiments. We broadly classified character types into 6 categories, including symbol, alphabet, number, ordinal number, Thai and other character types. Note that a word w_i is considered to be a rare word, if it occurs less than 2 times in the training set.

For the propose of comparison, we used the maximal matching algorithm for word segmentation and the unigram baseline method for POS tagging. The idea of the unigram baseline method is to assign each token with the most likely POS tag that can be estimated from the frequency count in the training set [4]. For word segmentation, precision is defined as the percentage of tokens recovered by the system that also occurred in the test set, while recall is defined as the percentage of tokens in the test set recovered by the system. For full morphological analysis (word segmentation and POS tagging), a token is considered to be a correct one only when both the word boundary and its POS tags are correctly identified. The following metrics are used to evaluate the performance.

Table 2. Features are based on three-level, first-order Markov assumption. Each node y contains morphological information of word surface w, POS p and sub-POS p'. $f_k(\cdot)$ is operated on $y_{t-1} \rightarrow y_t$

Unigram Feature	
condition	feature template
$\forall y_t$	$\langle p_t \rangle$
	$\langle p_t, p'_t \rangle$
	character type of $w_t \times \{\phi, \langle p_t \rangle, \langle p_t, p'_t \rangle\}$
w_t is not rare	$w_t \times \{\phi, \langle p_t \rangle, \langle p_t, p'_t \rangle\}$
w_t is rare	up to 2 prefixes of $w_t \times \{\phi, \langle p_t \rangle, \langle p_t, p'_t \rangle\}$
	up to 2 suffixes of $w_t \times \{\phi, \langle p_t \rangle, \langle p_t, p'_t \rangle\}$

Bigram Feature	
condition	feature template
$\forall y_{t-1} \rightarrow y_t$	$\langle w_{t-1}, w_t \rangle$
(if w_t is rare, replace	$\langle p_{t-1}, p_t \rangle$
its surface with a	$\langle w_{t-1}, w_t, p_t \rangle$
symbol '$*$')	$\langle w_{t-1}, p_{t-1}, w_t \rangle$
	$\langle p_{t-1}, p_t, p'_t \rangle$
	$\langle p_{t-1}, p'_{t-1}, p_t \rangle$
	$\langle w_{t-1}, p_{t-1}, p_t \rangle$
	$\langle p_{t-1}, w_t, p_t \rangle$
	$\langle w_{t-1}, p_{t-1}, w_t, p_t \rangle$
	$\langle w_{t-1}, p_{t-1}, p_t, p'_t \rangle$
	$\langle p_{t-1}, p'_{t-1}, w_t, p_t \rangle$
	$\langle p_{t-1}, p'_{t-1}, p_t, p'_t \rangle$
	$\langle w_{t-1}, w_t, p_t, p'_t \rangle$
	$\langle w_{t-1}, p_{t-1}, p'_{t-1}, w_t \rangle$
	$\langle w_{t-1}, p_{t-1}, p'_{t-1}, w_t, p_t, p'_t \rangle$

$$\text{Recall} = \frac{\#\text{ of correct tokens}}{\#\text{ of tokens in the test set}}$$

$$\text{Precision} = \frac{\#\text{ of correct tokens}}{\#\text{ of tokens recovered by the system}}$$

$$F_1 = \frac{2 \cdot \text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}}$$

$$\text{Recall}_{known} = \frac{\#\text{ of correct known tokens}}{\#\text{ of known tokens in the test set}}$$

$$\text{Recall}_{unknown} = \frac{\#\text{ of correct unknown tokens}}{\#\text{ of unknown tokens in the test set}}$$

4.2 Results

Tables 3 and 4 show the performance of the two-pass search algorithm by varying the probability threshold ϵ in the range of [0.5, 0.8]. The probability threshold

Table 3. Results of word segmentation

Algorithm	Recall	Precision	F_1	Recall$_{known}$	Recall$_{unknown}$
Baseline	$65.80(\frac{46780}{71097})$	$66.56(\frac{46780}{70284})$	66.18	$68.19(\frac{39127}{57381})$	$50.32(\frac{4732}{9403})$
Standard Search	$83.47(\frac{59345}{71097})$	$78.24(\frac{59345}{75851})$	80.77	$88.29(\frac{50663}{57381})$	$58.76(\frac{5525}{9403})$
Two-Pass$_{\epsilon=0.5}$	$\mathbf{85.15}(\frac{60541}{71097})$	$\mathbf{82.87}(\frac{60541}{73051})$	**84.00**	$\mathbf{89.35}(\frac{51272}{57381})$	$58.78(\frac{5527}{9403})$
Two-Pass$_{\epsilon=0.6}$	$84.69(\frac{60213}{71097})$	$82.72(\frac{60213}{72795})$	83.69	$88.79(\frac{50949}{57381})$	$58.83(\frac{5532}{9403})$
Two-Pass$_{\epsilon=0.7}$	$84.08(\frac{59777}{71097})$	$82.54(\frac{59777}{72418})$	83.30	$88.00(\frac{50493}{57381})$	$59.13(\frac{5560}{9403})$
Two-Pass$_{\epsilon=0.8}$	$82.84(\frac{58894}{71097})$	$82.03(\frac{58894}{71799})$	82.43	$86.47(\frac{49616}{57381})$	$\mathbf{59.16}(\frac{5563}{9403})$

Table 4. Results of word segmentation and POS tagging

Algorithm	Recall	Precision	F_1	Recall$_{known}$	Recall$_{unknown}$
Baseline	$57.58(\frac{40940}{71097})$	$58.25(\frac{40940}{70284})$	57.91	$61.19(\frac{35109}{57381})$	$31.28(\frac{2941}{9403})$
Standard Search	$77.57(\frac{55149}{71097})$	$72.71(\frac{55149}{75851})$	75.06	$82.46(\frac{47314}{57381})$	$50.14(\frac{4715}{9403})$
Two-Pass$_{\epsilon=0.5}$	$\mathbf{79.05}(\frac{56202}{71097})$	$\mathbf{76.94}(\frac{56202}{73051})$	**77.98**	$\mathbf{83.20}(\frac{47741}{57381})$	$50.51(\frac{4749}{9403})$
Two-Pass$_{\epsilon=0.6}$	$78.63(\frac{55901}{71097})$	$76.79(\frac{55901}{72795})$	77.70	$82.66(\frac{47434}{57381})$	$50.66(\frac{4764}{9403})$
Two-Pass$_{\epsilon=0.7}$	$78.08(\frac{55515}{71097})$	$76.66(\frac{55515}{72418})$	77.36	$81.97(\frac{47033}{57381})$	$50.90(\frac{4786}{9403})$
Two-Pass$_{\epsilon=0.8}$	$76.98(\frac{54729}{71097})$	$76.23(\frac{54729}{71799})$	76.60	$80.60(\frac{46251}{57381})$	$\mathbf{50.95}(\frac{4791}{9403})$

ϵ is used for identifying suspicious nodes. The baseline method cannot produce acceptable results. These results indicate how difficult the morphological analysis in ORCHID corpus is. The standard search which performs a single-pass Viterbi search can boost the performance with a reasonable gap. The best performance (except Recall$_{unknown}$) can be obtained by using the two-pass search algorithm with $\epsilon = 0.5$. The two-pass search algorithm improves the performance of the standard search by 3.23 F_1 in word segmentation and 2.92 F_1 in the combination of word segmentation and POS tagging. The two-pass search algorithm yields the best Recall$_{unknown}$ with $\epsilon = 0.8$. The more the value of ϵ is set, the more the suspicious nodes are identified. The new word hypotheses are subsequently generated, which in turn increase noises. In practice, the optimal value of ϵ can be selected by cross-validation.

In Table 5, we consider the effect of 'prefix & suffix' and 'character type' features in morphological analysis. We use the two-pass search algorithm with $\epsilon = 0.5$ and perform experiments without each feature. By removing 'prefix & suffix' feature, the performance decreases obviously. However, the contribution of character type feature is not significant as we expected. Without it, the performance improves slightly.

4.3 Error Analysis

From overall results, we observe that there are two major cases, which the two-pass search algorithm could not analyze the input strings correctly. The first

Table 5. Results without each feature produced by Two-Pass$_{\epsilon=0.5}$

Feature $\mathbb{F}$	Level	Recall	Precision	F_1	Recall$_{known}$	Recall$_{unknown}$
$\mathbb{F}\backslash\{$prefix & suffix$\}$	word	83.49 (-1.66)	81.77 (-1.10)	82.62 (-1.38)	88.44 (-0.91)	58.32 (-0.46)
	word & POS	77.55 (-1.50)	75.95 (-0.99)	76.74 (-1.24)	82.48 (-0.72)	50.10 (-0.41)
$\mathbb{F}\backslash\{$character type$\}$	word	85.41 (+0.26)	82.82 (-0.05)	84.10 (+0.10)	89.73 (+0.38)	58.69 (-0.09)
	word & POS	79.40 (+0.35)	76.99 (+0.05)	78.18 (+0.2)	83.66 (+0.46)	50.60 (+0.09)

case is when unknown words are mixed with various character types. Since ORCHID corpus was derived from technical papers, the number of technical terms and jargons is relatively high. For example, the surface 'Gao.65AI035AS' is segmented into smaller tokens, 'Gao', '.', '65', 'AI', '035' and 'AS', according to their character types. In this case, it is not difficult to resolve, because we can use some regular expressions to detect domain-specific word surfaces. The second case which makes Thai morphological analysis much more difficult is when unknown words are formed from known words, and each of them is otherwise found independently in the corpus. This case can be thought of as the problem of compound words. To deal with the problem, we need to find a set of features that can capture the compounding process.

5 Related Work

A wide variety of statistical corpus-based approaches has been applied to Thai morphological analysis. Jaruskulchai [3] used a model selection technique called minimum description length to estimate how likely a sequence of syllables can form a word. Aroonmanakun [1] exploited statistics of collocation to merge syllables to a word. These approaches are only focused on word segmentation, and hard to integrate POS tagging into their frameworks due to the limitation of problem formulation. Charoenpornsawat et at. [2] applied the Winnow algorithm to learn contextual features for handling the unknown word problem. Recently, Kruengkrai et al. [6] have proposed a unified framework for Thai morphological analysis based on conditional random fields (CRFs). Their approach uses a linear-chain formulation and produces a lattice using n-best word/tag sequences, while our approach uses a general graphical model (i.e., an instance of Markov random fields) and directly generates a lattice from a string input.

Discriminative models have been applied to morphological analysis in other non-segmenting languages. Uchimoto et al. [10] proposed a Japanese morpheme model that estimates probabilities of every substring for a given sentence based on maximum entropy. Kudo et al. [7] proposed a CRF-based training method that can avoid the label and length bias problems in Japanese morphological analysis. Our approach can also avoid those problems due to performing whole-sequence normalization.

6 Conclusion

We have presented a discriminative learning approach for Thai morphological analysis. We consider Thai morphological analysis as a search problem. We propose the two-pass search algorithm that finds the most likely path in the expanded search space. The objective of our algorithm is to increase the coverage of word hypotheses based on probability estimation in the lattice. The experimental results on ORCHID corpus show that the two-pass search algorithm can improve the performance over the standard search approach.

References

1. Aroonmanakun, W.: Collocation and Thai Word Segmentation. Proc. of the 5th SNLP & 5th Oriental COCOSDA Workshop. (2002) 68–75
2. Charoenpornsawat, P.: Feature-based Thai Word Segmentation. Master's Thesis, Computer Engineering, Chulalongkorn University. (1999)
3. Jaruskulchai, C.: An Automatic Thai Lexical Acquisition from Text. Proc. of PRICAI. (1998) 289–296
4. Jurafsky, D., Martin, J. H.: Speech and Language Processing: An Introduction to Natural Language Processing, Computational Prentice-Hall, Inc. (2000)
5. Kruengkrai, C., Charoenporn, T., Sornlertlamvanich, V., Isahara, H.: Acquiring Selectional Preferences in a Thai Lexical Database. Proc. of IJCNLP. (2004)
6. Kruengkrai, C., Sornlertlamvanich, V., Isahara, H.: A Conditional Random Field Framework for Thai Morphological Analysis. Proceedings of the Fifth International Conference on Language Resources and Evaluation. (2006)
7. Kudo, T., Yamamoto, K., and Matsumoto, Y.: Applying Conditional Random Fields to Japanese Morphological Analysis. Proc. of EMNLP. (2004)
8. Liu, D. C., Nocedal, J.: On the limited memory BFGS method for large scale optimization. Math. Programming. (1989) 45:503–528
9. Sornlertlamvanich, V., Charoenporn, T., Isahara, H.: ORCHID: Thai Part-Of-Speech Tagged Corpus. Technical Report TR-NECTEC-1997-001, NECTEC. (1997)
10. Uchimoto, K., Nobata, C., Yamada, A., Sekine, S., Isahara, H.: Morphological Analysis of a Large Spontaneous Speech Corpus in Japanese. Proc. of ACL. (2003)