

## ADAPTIVE CONTEXT PROCESSING IN ON-LINE HANDWRITTEN CHARACTER RECOGNITION

NAOMI IWAYAMA, KAZUSHI ISHIGAKI

*Fujitsu Laboratories Ltd., 64 Nishiwaki, Okubo-cho, Akashi, Hyogo 674-8555, Japan  
E-mail: naomi@flab.fujitsu.co.jp, ishigaki@flab.fujitsu.co.jp*

We propose a new approach to context processing in on-line handwritten character recognition (OLCR). Based on the observation that writers often repeat the strings that they input, we take the approach of adaptive context processing (ACP). In ACP, the strings input by a writer are automatically added to a dictionary designated for ACP. This dictionary thereby can provide good coverage of the strings a writer inputs. Furthermore, the dictionary is compact enough to be loaded on a small terminal. In our experiments, the first-hit rate of OLCR with ACP was 95.44% after all the strings to be input had been added to the ACP dictionary while that without ACP was 86.09%.

### 1 Introduction

Our work is on on-line handwritten character recognition (OLCR) consisting of classification and context processing (CP). Considerable efforts have been made toward increasing recognition accuracy. We developed a high-performance classifier that outputs the best candidates for an input pattern [1]. On average for the database (HANDS\_kuchibue\_d-97-06) [2], the classifier has marked 88.8% first-hit rate and the 99.0% rate that a correct character is included among 20 candidates (nominating rate). As post-processing, CP is useful for improving the first-hit rate [3], so in this paper, we focus on CP as post-processing.

We expect that the first-hit rate of our OLCR with effective CP may approximate to the 99.0% nominating rate since CP permutes candidates for each pattern position in an input pattern sequence. CP based on a character bigram model [4] raised the first-hit rate of our OLCR to 93.2%. We consider that this CP is not sufficiently effective because of a large difference between 93.2% and the 99.0% nominating rate. Therefore, the goal of this work is to develop more effective CP.

A dictionary for CP is the key to effective CP since CP works by referring to the dictionary that is called the context dictionary (CD). In this work, CD must satisfy two requirements. One requirement is that CD provides good coverage of strings input by a writer. This is because CP failures are often caused by a lack of strings in CD [5]. Good coverage means that CD includes not a large variety of the strings input by a writer but a large number of the strings input by a writer. In short, if CD includes the strings that a writer inputs many times, CD provides good coverage. The other requirement is that CD is compact enough to be loaded on a small mobile terminal.

The development of CD that satisfies the two above requirements is confronted by two difficulties. One difficulty is a trade-off between coverage of strings and dictionary size. The other difficulty is that a conventional CD cannot always provide good coverage even if the dictionary is very large. Since the dictionary is compiled beforehand from a training corpus irrespective of the writer [6], a coinage invented by the writer is never included in the dictionary. If the writer inputs the coinage many times, the dictionary cannot provide good coverage.

To overcome these difficulties, we take a new approach for our CD: this CD can adapt to the writer. This CD is called an adaptive context dictionary (ACD). In this paper, we propose effective CP with ACD, which we call adaptive context processing (ACP). Section two describes how ACD satisfies the two requirements, and section three explains how ACP is implemented to maximize its effectiveness. In section four, we show by experiments that OLCR with ACP improves the first-hit rate remarkably more than OLCR without ACP. Lastly, our conclusions are in section five.

## **2 Adaptive Context Dictionary**

This section describes how ACD satisfies the two requirements.

First, to provide good coverage, ACD must include the specific strings that a writer inputs. To find those strings, we gave careful consideration to two points about writers' behavior. One is that OLCR is often exclusively used by individual writers. Indeed, OLCR is often used to input strings into small mobile terminals, each of which has only one user. The other point is the high probability that a writer inputs the same string that the writer has input before. In a small-scale investigation, we observed that writers often repeat strings in their input. Based on these two observations, we reached the conclusion that if ACD includes repeated strings that a writer inputs, ACD can provide good coverage. Accordingly, we designed ACD to be dynamically compiled from strings input by a writer. Therefore, ACD can also include any repeated coinages that the writer inputs.

Next, to be compact, ACD must not increase monotonically. Moreover, ACD must not narrow its coverage. This implies that ACD must delete strings that the writer never inputs again. Therefore, we created the concept of least recently used strings, which are strings that have not been input again for a certain fixed period. These strings are designed to be automatically deleted from ACD. To find least recently used strings, a time stamp along with an input string is recorded in ACD when a writer inputs the string. In this way, ACD includes only the strings that a writer repeats in his/her input since the least recently used strings are deleted from ACD. Thus, ACD is compact in spite of providing good coverage.

Note that a writer requires no extra operation to construct ACD. If the output string from OLCR is not the desired input string, a writer can try to input the string

by rewriting it or selecting the string from a set of candidates. If the writer successfully inputs the string, the writer does not modify it, of course. To summarize, by the writer's reaction to the output from OLCR, OLCR can recognize whether the writer has confirmed the results of recognition as an input string. The strings confirmed by the writer are automatically added to ACD.

### 3 Adaptive Context Processing

This section explains how ACP is implemented to maximize its effectiveness.

First, the role of CP in OLCR is examined. OLCR is performed by a classifier and a context processor. For an input pattern sequence, the classifier outputs candidate strings that are ranked in order of confidence value. For each candidate string, the context processor tests its validity by referring to CD and assigns a new confidence value. According to the new confidence value, the context processor permutes candidate strings. Finally, OLCR outputs the string that ranks first. Basically, the role of CP is to raise the correct string to the first rank.

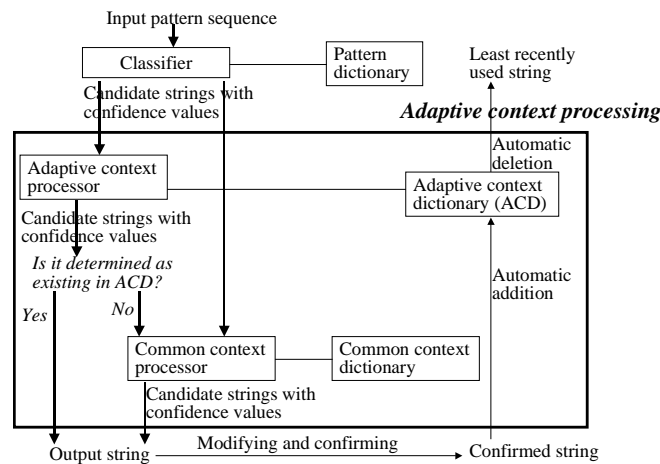


Figure 1: ACP

Next, ACP is described. Figure 1 shows an overview of ACP. To maximize the effectiveness of ACP, ACP must work well (a) when a writer inputs a string that exists in ACD, and (b) when a writer inputs a string that does not exist in ACD.

For ACP to work well in both situations, our ACP design combines two context processors referring to different CDs. One is a context processor referring to ACD, and we call this an adaptive context processor (AC processor). The other is a context processor based on a character bigram model, and we call this a common context processor (CC processor).

The algorithm for ACP is as follows. Throughout this section, suppose that a candidate string from a classifier is  $S$ . Let  $\{S\}$  be a set of  $S$ -es, let  $S_{C1}$  be the first  $S$ , let  $L_C(S)$  and  $L_{AC}(S)$  and  $L_{CC}(S)$  be the confidence values assigned to  $S$  by the classifier and the AC processor and the CC processor respectively, and let  $S_{AC}$  be the  $S$  that has the highest  $L_{AC}(S)$  among the  $S$ -es that exist in ACD.

- (i)  $S_{AC} = \text{null}$  and  $L_{AC}(S_{AC}) = 0$
- (ii) foreach  $S_i$  (all strings that exist in ACD)
- (iii) if  $S_i$  exists in  $\{S\}$  then
- (iv) if  $L_{AC}(S_i) > L_{AC}(S_{AC})$  then  $S_{AC} = S_i$
- (v) if  $S_{AC}$  is not null and  $L_{AC}(S_{AC}) > L_C(S_{C1})$  then
- (vi) OLCR outputs  $S_{AC}$
- (vii) else
- (viii)  $S_{CC1} = S_j$  such as  $L_{CC}(S_{CC1}) = \max\{L_{CC}(S_j) | S_j \text{ exists in } \{S\}\}$
- (ix) OLCR outputs  $S_{CC1}$

The algorithm is described below in greater detail.

First,  $L_{AC}(S)$  is described. We defined  $L_{AC}(S)$  as the formula (1) if  $S$  does not exist in ACD and as the formula (2) if  $S$  exists in ACD. To determine the validity of  $S$  that exists in ACD, we chose the following two criteria.

- How recently did the writer input  $S$ ?
- How frequently does the writer input  $S$ ?

The definition of the formula (2) is based on the two criteria. For two candidates  $S$  and  $S'$ , if both of them exist in ACD and  $S$  is more valid than  $S'$  on the basis of the two criteria,  $L_{AC}(S)$  is higher than  $L_{AC}(S')$ .

$$L_{AC}(S) = L_C(S) \quad (1)$$

$$L_{AC}(S) = L_C(S) + w_1 * f(\text{elapse}(S)) + w_2 * g(\text{freq}(S)) \quad (2)$$

where  $\text{elapse}(S)$  is the elapsed time from the time of the most recent use of  $S$  to the current time,  $\text{freq}(S)$  is the usage frequency of  $S$ ,  $f$  and  $g$  are the functions of  $\text{elapse}(S)$  and  $\text{freq}(S)$  respectively, and  $w_1$  and  $w_2$  are the weights. We designed  $f$  as a monotonic decreasing function and  $g$  as a monotonic increasing function.

By our definition, if  $S$  exists in ACD,  $L_{AC}(S)$  is higher than  $L_C(S)$ . This means that the AC processor raises the rank of  $S$  that exists in ACD since our classifier outputs the best candidates in descending order of  $L_C(S)$ .

We determined  $w_1$  and  $w_2$  statistically from a pair of confidence values that the classifier assigns to the first  $S$  and to the correct  $S$ . This means that the AC processor was adjusted to the classifier. Consequently, if  $S$  exists in ACD and  $L_C(S)$  is high enough to be reliable,  $L_{AC}(S)$  is higher than  $L_C(S_{C1})$ . Otherwise, if  $S$  exists in ACD and  $L_C(S)$  is too low to be reliable,  $L_{AC}(S)$  is not higher than  $L_C(S_{C1})$ .

Next, step (v) and (vi) are described. If the condition in step (v) evaluates to "true",  $S_{AC}$  ranks first in descending order of  $L_{AC}(S)$ . In this case, OLCR outputs  $S_{AC}$  as the results of recognition.

Finally, step (viii) and (ix) are described. If the condition in step (v) evaluates to "false", the first candidate from the AC processor is  $S_{C1}$ , which does not exist in

ACD. In this case, the results from the AC processor are abandoned, and output from the classifier is sent to the CC processor. Lastly, OLCR outputs the first candidate from the CC processor,  $S_{CC1}$ , as the results of recognition.

In the algorithm, an input string is determined as either existing in ACD or not by step (v). If the condition in step (v) evaluates to “true”, this is regarded as case (a) (see paragraph 3 in this section). In case (a), our classifier mostly outputs the input string as a candidate since our classifier has the 99.0% nominating rate. Moreover, since the AC processor is adjusted to the classifier, the AC processor raises the candidate to the first rank. Therefore,  $S_{AC}$  is regarded as the input string. This means that ACP works well for case (a).

Otherwise, if the condition in step (v) evaluates to “false”, this is regarded as case (b). In case (b), ACP that the AC processor performs fails because the AC processor refers to ACD that does not include the input string. Consequently,  $S_{C1}$  is not regarded as the input string. Since the CC processor refers to CD that has been compiled from a training corpus, the CC processor can test the validity of strings that any writer may have input. This means that ACP also works well for case (b).

#### 4 Experiments

This section describes experiments showing that OLCR with ACP improves the first-hit rate more than OLCR without ACP.

The experiments were conducted as follows. The data used in the experiments is data from a person’s schedule over a three month period, and it contains 400 characters. Fourteen subjects participated in the experiments. Using OLCR, they inputted the data in three cycles. In the first cycle, they used OLCR without ACP. In the second and third cycles, they used OLCR with ACP. In the third cycle, each subject inputted the data under the condition that all the strings to be input have been added to the individual writer’s ACD. ACD size was about 3 KB on average.

To prove that OLCR with ACP improves the first-hit rate more than OLCR without ACP, we have to confirm the following two properties. First, the classifier has to have no difference in its first-hit rates among the three cycles. Second, the first-hit rates after CP have to be significantly different among the three cycles.

Table 1: First-hit rate and nominating rate of classifier and first-hit rate after CP

Cycle of experiment	First	Second	Third
First-hit rate of classifier (%)	<b>81.71</b>	<b>81.84</b>	<b>80.51</b>
Nominating rate of classifier (%)	<b>97.19</b>	<b>97.54</b>	<b>97.73</b>
First-hit rate after CP (%)	<b>86.09</b>	<b>89.10</b>	<b>95.44</b>

Table 1 lists the first-hit rates and nominating rates of the classifier and the first-hit rates after CP in the three cycles. The results of analysis of variance (ANOVA) for the first-hit rates of the classifier show no significant difference

( $F(2,13)=0.78$ ,  $p<0.05$ ). Therefore, we conclude that there is no difference in the first-hit rate of the classifier among the three cycles.

The results of ANOVA show no significant difference in the first-hit rate between the first cycle and second cycle ( $F(1, 13)=2.49$ ,  $p<0.05$ ), but they show a significant difference between the first cycle and the third cycle ( $F(1,13)=24.8$ ,  $p<0.05$ ). Based on these results, we conclude that ACP is effective for data that has many repeated strings.

## 5 Conclusions

We have proposed adaptive context processing (ACP) with ACD in OLCR. ACD is dynamically updated as a writer inputs strings into OLCR. In our experiments, ACD size was about 3 KB after all the strings to be input had been added to ACD. The first-hit rate of OLCR with ACP was 95.44% while that without ACP was 86.09%.

We analyzed cases where OLCR with ACP made misrecognitions even though input strings were included in ACD. We found that a misrecognition might occur if the classifier does not output an input string as a candidate. Adapting the classifier to individual writers enables an input string to be output as a candidate. Therefore, the adaptive classifier integrated with ACP achieves even higher first-hit rate.

## Acknowledgments

We would like to thank Prof. Masaki Nakagawa for valuable advice. We also wish to thank Noboru Iwayama for critical review and all participants in our experiments.

## References

1. H. Tanaka et al., Hybrid Pen-Input Character Recognition System Based on Integration of Online-Offline Recognition, Proc. 5th ICDAR (1999) pp.209-212.
2. M. Nakagawa et al., Collection and utilization of on-line handwritten character patterns sampled in a sequence of sentences without any writing instructions, IEICE Technical Report, PRU95-110, vol.95, No.278 (1995) pp.43-48.
3. M. Nagata, Context-Based Spelling Correction for Japanese OCR, Proc. of 16th COLING (1996) pp. 806-811.
4. M. Nakagawa et al., Robust and Highly Customizable Recognition of On-line Handwritten Japanese Characters, Proc. 13th ICPR (1996) pp.269-273.
5. M. Okamoto et al., Theme on On-line Handwriting Character Recognition Method and String Separation Method Using Language Processing, Progress in Handwriting Recognition (1996) pp.387-392.
6. R.K. Srihari, Use of lexical and syntactic techniques in recognizing handwritten text, Proc. of the ARPA workshop on Human Language Technology (1994) pp.403-407.