A semi-incremental recognition method for on-line handwritten Japanese text

Cuong Tuan Nguyen, Bilan Zhu and Masaki Nakagawa
Department of Computer and Information Sciences
Tokyo University of Agriculture and Technology

***Abstract*—This paper presents a semi-incremental recognition method for online Japanese handwritten text recognition, which is used for busy recognition interface (recognition while writing) and lazy recognition interface (recognition after writing) without large waiting time. We employ local processing strategy and focus on a recent sequence of strokes defined as "scope". For the latest scope, we build and update a segmentation and recognition candidate lattice and advance the best-path search incrementally. We utilize the result of the best-path search in the previous scope to exclude unnecessary segmentation candidates. This reduces the number of candidate character recognition with the result of reduced processing time. We also reuse the segmentation and recognition candidate lattice in the previous scope for the latest scope. Moreover, triggering recognition processes every few strokes save CPU time. Experiment made on TUAT-Kondate database shows the effectiveness of the proposed method not only in reduced processing time and waiting time, but also in recognition accuracy.**

***Keywords*—Online recognition, handwriting recognition, incremental recognition**

I. Introduction

In recent years, due to the development of pen input devices such as Tablet PCs, electronic whiteboards, PDAs, digital pens (like the Anoto pen) and touch-based smart phones, Pad PCs, and so on, online handwritten text recognition as an input method has been given a considerable attention after a long period of research [1, 2, 3]. Compared to isolated character recognition, handwritten text recognition faces the difficulty of character segmentation and recognition. Moreover, in continuous handwriting, characters tend to be written more cursively.

To obtain high recognition rate, it is best to recognize online handwritten text after the whole text is completed since the full context information is available. We call this method as batch recognition [4]. Batch recognition is appropriate for the user interface that users are writing while thinking. In this case, users do not need recognition result when writing and they only need recognized text when they break writing. We call this user interface as lazy recognition interface [5] while we call on-the fly recognition after each character is written as busy recognition interface. However, waiting time of recognizing whole text by the batch recognition takes time as the amount of characters increases.

For the busy recognition interface and the lazy recognition interface as well, incremental recognition is essential. Tanaka shows an incremental recognition system for online Japanese handwriting recognition in his patent application [6]. Wang, Liu and Zhou have presented an approach to real-time (incremental) recognition of Chinese handwritten text [7, 8]. In Wang's method, the candidate characters are generated and recognized to assign candidate classes whenever a new stroke is produced, and sentence recognition result is produced whenever pen up time exceed a specific value. The incremental recognition is also useful for the lazy recognition interface. We can apply it in background while a user is writing so that the whole recognition result is obtained without any noticeable waiting time.

Here in this paper, we focus on when incremental recognition processes are triggered. If a system triggers them whenever every new stroke is given, we classify it as pure incremental recognition. So far, all the published incremental recognition systems are classified in this group. However, we may trigger the processes by a little larger unit, i.e., several strokes so that we can exploit a little larger context. We classify such a system as semi-incremental recognition. This paper presents a semi-incremental recognition of online handwritten Japanese text, which is useful for both the busy and the lazy recognition interfaces. Whenever the number of newly written strokes reaches to the fixed number *Ns* named window size, the new strokes are added to the previous strokes, character patterns are segmented, candidate character patterns are recognized, a lattice representing segmentation and recognition candidates is updated, and search is processed, while writing continues. This process is repeated on recent strokes rather than on full text, so that text recognition result is shown immediately after writing is finished without noticeable waiting time while keeping high recognition rate.

Although batch recognition achieves high recognition rate with low total CPU time, it costs large waiting time as the amount of characters increases. On the contrary, pure incremental recognition incurs little waiting time but the recognition rate may drop due to local processing of every stroke and the total CPU time is increased due to repeated processing after receiving every stroke. Semi-incremental recognition with appropriate value of the window size may maintain high recognition rate as batch recognition, incur little waiting time and decrease the total CPU time compared to pure incremental recognition. Human recognition is neither too global covering full text nor too local focusing to each stroke, so that the semi-incremental recognition may realize the processing similar to human way of employing context.

Here we define several terms. A stroke is a sequence of pen-tip coordinates from pen-down to pen-up. An off-stroke is a vector from pen-up to pen-down. Digital ink is a sequence of strokes and off-strokes.

In the rest of this paper, Section 2 gives an overview of the baseline batch recognition method. Section 3 describes the semi-incremental recognition method. Section 4 presents experimental result of the semi-incremental recognition method. Section 5 draws our conclusion.

II. Overview of the batch recognition method

This section describes the overview of the batch recognition method. It processes all on-line handwritten text patterns at a time, i.e., after all strokes are added, it estimates the average character size and the center line, applies segmentation based on them, recognizes each segment of strokes and finally employs context information to find the best recognition of handwritten text.

A. Segmentation

Using the technique presented in [9], we first separate multiple text lines into each text line. Then, we segment each text line into candidate character patterns as shown in Figure 1. Here, we employ the strategy of over-segmentation, using SVM to classify each off-stroke into three classes, segmentation point (*SP*), non-segmentation point (*NSP*) and undecided point (*UP*) according to some geometric features4]. A segmentation point *SP* separates two characters at the off-stroke while a non-segmentation point *NSP* indicates the off-stroke is within a character. Off-strokes with low confidence are classified as *UP*. An off-stroke between two text lines is treated as *SP*. A sub-sequence of strokes delimited by *SP* or *UP* off-strokes is called a primitive segment. A primitive segment and consecutive primitive segments beside *UP* form candidate character patterns. Concatenation of consequent primitive segments is limited by their total lengths.

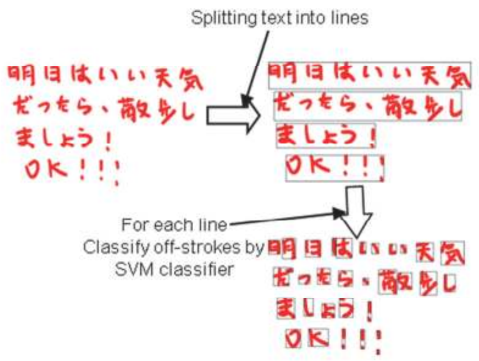

Splitting text into lines
For each line
Classify off-strokes by SVM classifier

Figure 1. Segmentation process.

B. Candidate lattice construction

Employing character recognition, each candidate character pattern is associated with a number of candidate classes with confidence scores. All the possible segmentations and recognition candidate classes are represented by a segmentation-recognition candidate lattice (src-lattice in short) as shown in Figure 2, where each node denotes a candidate segmentation point and each arc denotes a character class assigned to a candidate character pattern.

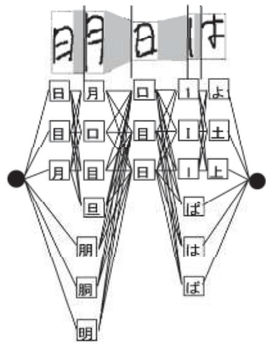


Figure 2. Segmentation-recognition candidate lattice

For implementation, we employ candidate character blocks and each of them represents a set of all the candidate character patterns separated by two adjacent *SP* off-strokes. Figure 3 shows them for the src-lattice with two *SP* off-strokes and three candidate character blocks.

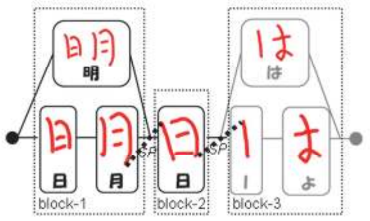


Figure 3. Candidate character blocks.

C. Best-path search and recognition

From a src-lattice, paths are evaluated by combining the scores of character recognition, geometric features and linguistic contexts as proposed in [4]. By applying the Viterbi algorithm, the optimal path which has the highest evaluation score is found. Text recognition result is obtained from this path.

III. Semi-incremental recognition method

The main objective to develop the semi-incremental recognition method is to perform possible computation as much as possible while a user is writing. Moreover, it should keep the recognition rate as high as possible compared with the batch recognition method. In the batch recognition, the majority of computing time is spent for the recognition of candidate character patterns. If those can be processed in the background of user's handwriting, text recognition result will be displayed without any noticeable waiting time.

A. Strategy of local processing

Semi-incremental recognition performs recognition process after receiving some newly written strokes. Ideally, we only have to process the newly received strokes and update the src-lattice. In fact, the newly added strokes affect recognition of a small number of strokes previously received. Thus, the section we must process includes these strokes and the newly received strokes. We call it "scope".

As for best-path search, it is made from the first stroke to the last stroke in the batch recognition while it can be made incrementally using scope. Therefore, if the scope is well defined, the semi-incremental recognition should produce almost the same recognition result without incurring much waiting time.

We also introduce a pointer named segmentation resuming pointer, or ***Seg_rp*** in short. This is a *SP* up to which the result of segmentation and character recognition is considered stable and fixed. Segmentation and recognition is resumed from the latest ***Seg_rp*** according the evaluation on the latest scope.

B. Processing flow

From the previously described strategy, Figure 4 shows the processing flow of the semi-incremental recognition method.

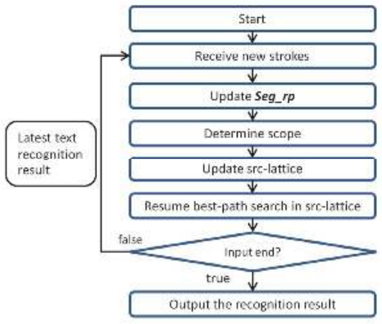

Start
Receive new strokes
Update Seg_rp
Determine scope
Update src-lattice
Resume best-path search in src-lattice
Input end?
false
true
Output the recognition result
Latest text recognition result

Figure 4. Flow of semi-incremental recognition

First, we receive new strokes. Secondly, we update ***Seg_rp***. Thirdly, we determine the scope. Since this processing step includes segmentation, we do not need to apply it in the successive processing steps. Fourthly, we update the src-lattice. Finally, we resume the best-path search from the beginning in this scope to get text recognition result. The result is used for next processing cycle.

C. Determination of scope

To determine the scope, we use the result from the segmentation process. The segmentations of the strokes before and after the system has received new strokes are compared with each other. If classification-changed off-strokes are detected, we consider the strokes before the earliest classification-changed off-strokes are stably classified while the strokes after that are not classified stably. Otherwise, the off-stroke before the newly added strokes is considered as the earliest classification-changed off-stroke. This earliest classification-changed off-stroke may occur within some candidate character block or between two candidate character blocks. We define the scope as the sequence of strokes starting from the first stroke of the candidate character block containing or just preceding the earliest classification-changed off-stroke to the last stroke.

If we had to compare the results of segmentation before and after new strokes are added over a long range strokes, however, it slows down the process and contradicts the local processing strategy. Therefore, we employ ***Seg_rp*** from which we compare segmentation.

D. Seg_rp and segmentation point determination

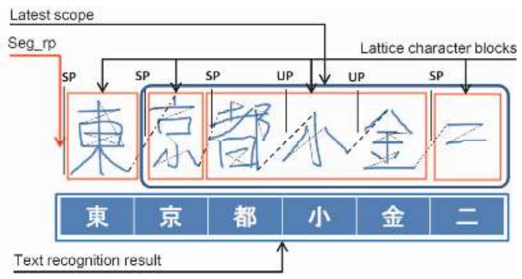
The pointer: ***Seg_rp*** is determined from *SP* off-strokes. From the result of text recognition up to the latest scope, i.e., the best-path up to the latest scope in the src-lattice, an off-stroke between two recognized characters can be considered as *SP* confidently. Among those off-strokes, we choose ***Seg_rp*** based on the number of characters from each off-stroke to the last character in the recognition result. If this number equals to N_CHAR, that off-stroke will be determined as a new ***Seg_rp***. N_CHAR is defined as a fixed number of characters required to determine a new ***Seg_rp***. The idea behind this is that *SPs* away from new strokes are stable.

Determination of *SP* off-strokes has large effect to the recognition rate and performance of the system. Although *SP* off-strokes can be detected based on the result of segmentation process, the performance of segmentation by SVM for detecting *SP* off-strokes is still limited. Due to the uncertainty of segmentation, a large number of outputs from SVM are marked as *UP*. Each *UP* roughly doubles the number of candidate character patterns for which character recognition is applied. To overcome this problem, we also use the result of text recognition up to the latest scope to determine *UP* to *SP* off-strokes. *UP* off-strokes between recognized characters, before the latest N_CHAR_MIN characters in the recognition result are determined as *SP* off-strokes. Here, N_CHAR_MIN denotes a predefined constant for the minimum number of characters that follow an *UP* off-stroke to make it a stable *SP* off-stroke. Generally, N_CHAR_MIN is smaller than or equal to N_CHAR.

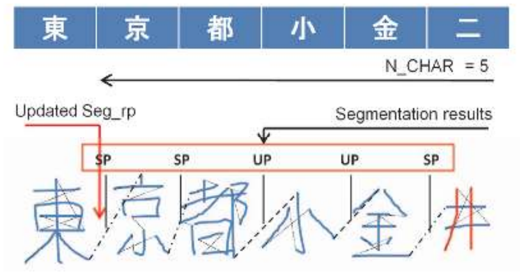
By setting ***Seg_rp***, the maximum number of characters in the last block is bounded by N_CHAR plus the number of new characters in newly added strokes. This is the main factor to reduce the maximum waiting time in each processing. Moreover, changing more *UP* off-strokes to *SP* in lattice blocks also reduces the time cost to rebuild the src-lattice due to shortened block size.

E. An example of the processes

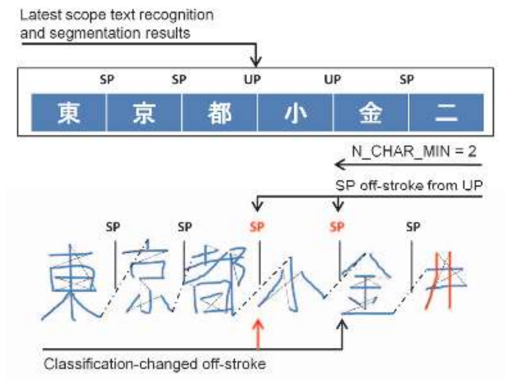
Figure 5 shows an example to determine the scope**.** Assume the latest scope with segmentation and text recognition results in Figure 5(a). Then, the new strokes marked red are added. We update ***Seg_rp*** and apply segmentation from the updated ***Seg_rp*** (Figure 5(b)). Next we change *UPs* to *SPs* if they satisfy the above-mentioned condition and find the earliest classification-changed off-stroke (Figure 5(c)). Finally, we locate the character block including or just preceding this off-stroke and update the scope (Figure 5(d)).



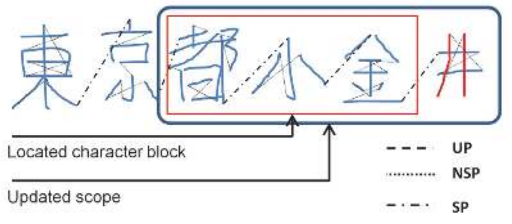
(a). Latest scope with segmentation and text recognition results.



(b). Receiving new strokes, updating ***Seg_rp*** and applying segmenation.


Latest scope text recognition and segmentation results
SP SP UP UP SP
東 京 都 小 金 二
N_CHAR_MIN = 2
SP off-stroke from UP
SP SP SP SP SP
Classification-changed off-stroke

(c). Determining *UPs* to *SPs* and finding classification-changed offstrokes.



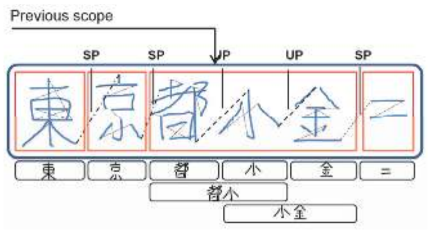
(d). Locating the character block and updating the scope

Figure 5. An example of determining scope.

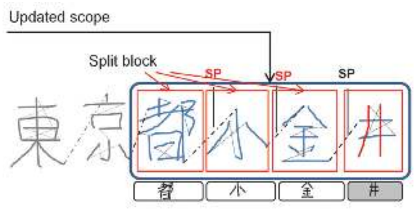
F. Update of src-lattice

For updating the src-lattice in the latest scope, to maximize the reuse of the src-lattice in the previous scope, we use the following method. It takes advantage of previously built lattice candidates in the previous scope. From the beginning of the scope, the method finds *SP* off-strokes and splits candidate character blocks by these *SP* off-strokes. Each *SP* off-stroke divides a candidate character block into two parts: the preceding part and the succeeding part beside this *SP* off-stroke. The src-lattice in these lattice blocks will be checked if a candidate character pattern already exists in the previous scope. When exists, we get it from the previous scope, otherwise we rebuild it.

Figure 6(a) represents the lattice blocks of the previous scope, when new strokes are added as shown in Figure 6(b), classification of the off-stroke between the two first characters in the updated scope is changed to *SP*. From this *SP* off-stroke, the previously built candidate character block is divided into three candidate character blocks and the candidate character patterns of the previous scope is reused for the updated scope. Then, only one candidate character pattern (shown in gray) are rebuilt due to the new strokes.



(a) Previous scope.



(b) Updated scope.

Figure 6. Reuse of candidate character patterns.

G. Resuming best-path search and recognition

From the first character lattice block in current scope, we resume best-path search and get text recognition result.

IV. EXPERIMENTS

For evaluating our proposed semi-incremental recognition method, we use horizontally written character string patterns extracted from the TUAT Kondate database collected from 100 people. We employ 10,174 strings for training and 3,511 strings for testing. The experiments are implemented on an Intel(R) Core™2 Quad Q9400 CPU 2.66GHz CPU with 3.25GB memory.

The first experiment is to evaluate recognition rate. We test the recognition rate with changing the N_CHAR parameter and the number of strokes received in each recognition (N_STROKE). Table I shows the test result. While the batch recognition rate is 93.03%, the semi-incremental recognition method outperform the batch recognition with the maximum rate being 93.16% at N_CHAR = 7 and N_STROKE = 4. This reflects the effect of recognition in a local scope.

To evaluate processing time, average CPU time per stroke is shown in Table II. Compared to the pure incremental method, i.e. when N_STROKE = 1, the semi-incremental method with N_STROKE = 4 saves about 50% in the CPU time.

TABLE I. RECOGNITION RESULT (%).

N_STROKE	**N_CHAR**					
	3	***4***	***5***	***6***	***7***	***8***
1	92.64	92.75	92.89	93.01	92.93	92.93
2	92.71	92.94	92.96	92.97	92.97	92.97
3	92.82	92.95	92.97	92.99	92.99	93.03
4	93.00	93.10	93.12	93.15	**93.16**	93.11
5	93.02	93.08	93.07	93.08	93.12	93.07

TABLE II. PROCESSING TIME PER STROKE (MS).

Semi-Incremental							Batch
N_STROKE	N_CHAR						3.04
	3	*4*	*5*	*6*	*7*	*8*	
1	8.47	9.60	10.81	11.46	11.57	11.95	
4	4.60	5.16	5.60	5.82	5.75	6.04	

To evaluate the segmentation, we also extract segmentation results after the recognition process and compare them with those by the batch recognition method. Table III shows the evaluation result on three measures: precision, recall and f-measure. From f-measure, the performance of the semi-incremental recognition method is slightly better than the batch recognition method when N_ CHAR > 5.

TABLE III. SEGMENTATION RESULT (%).

Eval. measure	Recognition method							
	Semi-incremental N_CHAR							Batch
	3	*4*	*5*	*6*	*7*	*8*	*9*	
Recall	98.95	98.91	98.90	98.88	98.85	98.84	98.82	98.75
Precision	99.05	99.20	99.26	99.30	99.31	99.33	99.33	99.37
F-measure	99.00	99.06	99.08	99.09	99.08	99.09	99.08	99.06

The next experiment is to evaluate waiting time of the method. We also make reflection of the method without reuse of the src-lattice. This evaluation is done on 5 different pages of handwritten text captured from touch screen devices with the number of strokes for each page being 347, 398, 590, 262, or 554, respectively.

Figure 7 shows average waiting time when increasing N_CHAR from 3 to 9 and N_STROKE is fixed at 1. When N_CHAR increases, the average waiting time slightly increases. Figure 8 shows it when increasing N_STROKE while fixing N_CHAR at 3.

Average waiting time without reuse of the src-lattice is shown in those figures where we can see the effect is significant.

When N_STROKE is less than 5, the maximum waiting time is about 55ms, which is too small for a user to notice the delay.

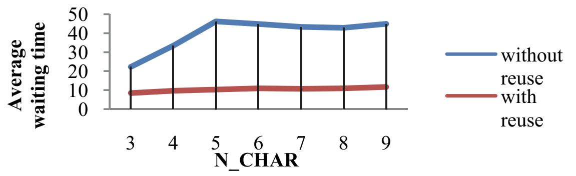


Figure 7. Waiting time with N_CHAR.

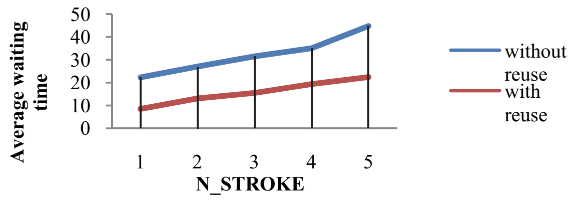


Figure 8. Waiting time with N_STROKE.

V. CONCLUSION

In this paper, we presented a semi-incremental recognition method for on-line handwritten Japanese text. By employing local processing, average waiting time has been reduced. Moreover, determining *SP* off-strokes based on recognition result shortens block lengths, bounds waiting time and even increases the recognition rate slightly. The reuse of the src-lattice is also shown effective.

The semi-incremental recognition method is superior to the batch recognition method clearly in waiting time and even in recognition rate. It also excels the pure incremental recognition method in recognition rate and total CPU time.

The semi-incremental recognition method should also work for other languages by changing the parameters.

The disadvantage of this method arises also from local processing. Delayed strokes occurring before the latest ***Seg_rp*** will not be processed and it could cause misrecognition result. This can be solved by keeping all the lattice blocks. When new strokes are added, the system determines the lattice block, updates the candidate lattice in the lattice block and produces recognition result.

ACKNOWLEDGMENT

This work is being supported by the Grant-in-Aid for Scientific Research (B)-224300095.

REFERENCES

[1] C. L. Liu, S. Jaeger, and M. Nakagawa, "Online Recognition of Chinese Characters: The State-of-the-Art," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 26, no. 2, pp. 198-213, February 2004.

[2] R. Plamondon and S. N. Srihari, "On-Line and Off-Line Handwriting Recognition: A Comprehensive Survey," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 63-85, January 2000.

[3] A. Graves, S. Fernandez, M. Liwicki, H. Bunke, and J. Schmidhuber, "Unconstrained Online Handwriting Recognition with Recurrent Neural Networks," *Advances in Neural Information Processing Systems*, vol. 20, pp. 577-584, 2008.

[4] B. Zhu, X.D. Zhou, C.L. Liu, and M. Nakagawa, "A robust model for on-line handwritten japanese text recognition," *International Journal on Document Analysis and Recognition*, vol. 13, no. 2, pp. 121-131, June 2010.

[5] M. Nakagawa, K. Machii, N. Kato, and T. Souya, "Lazy Recognition as a Principle of Pen Interfaces," in *ACM INTERCHI*, 1993, pp. 89-90.

[6] H. Tanaka, "Implementation of real-time box-free online Japanese handwriting recognition system," *Japanese Patent* 3925247, issued March 13, 2002 (in Japanese).

[7] D.H. Wang and C.L. Liu, "An Approach to Real-Time Recognition of Chinese Handwritten Sentences," in *CJKPR*, Fukuoka, Japan, 2010.

[8] D.H. Wang, C.L. Liu, and X.D. Zhou, "An approach for real-time recognition of online Chinese handwritten sentences," *Pattern Recognition*, no. 45, pp. 3661-3675, 2012.

[9] X.D. Zhou, D.H. Wang, and C.L. Liu, "A robust approach to text line grouping in online handwritten Japanese documents," *Pattern Recognition*, vol. 42, no. 9, pp. 2077-2088, 2009.