Semi-Incremental Recognition of On-Line Handwritten Japanese Text

Cuong-Tuan NGUYEN{*}, Student Member, Bilan ZHU¹, Member, and Masaki NAKAGAWA¹, Fellow

SUMMARY This paper presents a semi-incremental recognition method for on-line handwritten Japanese text and its evaluation. As text becomes longer, recognition time and waiting time become large if it is recognized after it is written (batch recognition). Thus, incremental methods have been proposed with recognition triggered by every stroke but the recognition rates are damaged and more CPU time is incurred. We propose semi-incremental recognition and employ a local processing strategy by focusing on a recent sequence of strokes defined as "scope" rather than every new stroke. 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 several strokes saves CPU time. Experiments made on TUAT-Kondate database show the effectiveness of the proposed semi-incremental recognition method not only in reduced processing time and waiting time, but also in recognition accuracy.

key words: online recognition, handwriting recognition, incremental recognition

1. Introduction

In recent years, due to the development of pen-based and touch-based devices such as tablets, digital pens (like the Anoto pen) and touch-based smart phones, on-line handwritten text recognition as an input method has been given considerable attention after a long period of research [1]–[3]. Since hand-held devices have relatively smaller CPU performance for less power consumption compared with desktop PCs and they are interactive devices, however, handwriting recognition on these devices must respond to user’s input with high recognition rates but without incurring much CPU time.

For on-line handwritten text recognition, utilizing full context of the whole input sequence is important. Nakagawa et al. [4] shows the effect of integrating linguistic context, character structure context along with recognition score to improve on-line handwritten Japanese text recognition. Graves et al. [3] shows the state-of-the-art on-line English handwriting recognition with the bi-directional recurrent neural networks which integrate context from both forward and backward directions.

To implement these methods, it is straightforward to recognize on-line handwritten text after the whole text is completed. We call this strategy as batch recognition [5]. Batch recognition is appropriate for the user interfaces where users are writing while thinking. In this case, users do not need recognition result when writing and they only need recognized text when they suspend writing. We call this user interface as lazy recognition interface [6] while we call on-the-fly recognition user interfaces after every character or stroke is written as busy recognition interfaces. Here, a stroke is a sequence of finger-tip or pen-tip coordinates from finger/pen-down to finger/pen-up.

Although the batch recognition strategy can easily use the full context to achieve high recognition rates, waiting time of recognizing whole text by the batch recognition takes time as the amount of characters increases. To solve this problem, Tanaka et al. [7] and Wang et al. [8] proposed incremental recognition methods which recognize handwriting as users write Japanese and Chinese, respectively. Although incremental recognition does not incur long waiting time after the user has finished writing, it may degrade the recognition rate due to local processing of every stroke. In fact, the recognition rate by incremental recognition decreases by about 0.3 point as compared with batch recognition [7]. Incremental recognition also extend the total CPU time due to repeated processing after receiving every stroke.

In this paper, we focus on when incremental recognition processes are triggered. If a system triggers them whenever a 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 this strategy as semi-incremental recognition. This paper presents a semi-incremental recognition method of on-line handwritten Japanese text, which is useful for both the busy and the lazy recognition interfaces. Whenever the number of newly written strokes reaches the fixed number named the 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 a high recognition rate.

Copyright © 2016 The Institute of Electronics, Information and Communication Engineers
Although batch recognition achieves a 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 extended 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 with the pure incremental recognition.

This paper is based on the conference paper [9]. After it was published, recognition of unnecessary incomplete characters was reduced, the process to handle delayed strokes was included and all the components were tuned. This paper presents all the components and comprehensive evaluations. The method is employed in commercial on-line handwriting recognizers.

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

2. Batch Recognition Method

This section describes the batch recognition method for on-line handwritten Japanese text. It presents the strategy and all the components. It processes whole on-line handwritten text 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.

2.1 Strategy of Soft-Decision for Segmentation and Recognition

In recognition of on-line handwritten text with the explicit segmentation approach, there are two main tasks. First, an input sequence of strokes is segmented into smaller units as lines, words and characters. Second, the segmented units are recognized and the best path is searched to maximize the total score of segmentation and recognition.

Character segmentation is done based on geometric layout features. Due to the instability and ambiguity of these features in actual handwriting, however, it is difficult to determine segmentation without using recognition cues and linguistic context. Therefore, soft-decision is employed for segmentation and recognition. Then, the best path search is applied to perform segmentation and recognition. Namely, the following process is applied. Handwritten text is segmented into text lines and each text line is over-segmented into primitive segments such that each segment is composed of a single character or a part of a character. A segment or a sequence of a few consecutive segments is assumed as a candidate character pattern, which is recognized by a character recognizer with a list of candidate categories and scores. Multiple ways of segmentation into candidate character patterns and multiple ways of recognition into character classes are represented in a segmentation-recognition candidate lattice (src-lattice in short) [5]. Text recognition result is produced by searching into the lattice for the path with the highest total score of geometric context, linguistic context and character recognition scores. Figure 1 shows the flow of the batch recognition method.

2.2 Segmentation

Using the technique presented in [10], we first separate an input sequence of strokes into text lines. Then, we segment each text line into candidate character patterns as shown in Fig. 2. For over-segmentation, we apply the support vector machine (SVM) to classify each off-stroke (vector from pen-up to pen-down) into three classes, segmentation point (SP), non-segmentation point (NSP) and undecided point (UP) according to geometric features [5]. 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.

Table 1 and Table 2 show terms to derive geometric features and geometric features derived, respectively. Since the...
entire input sequence of strokes is available, batch recognition can use both the contexts in forward and backward directions for segmentation. The features for determining segmentation at the current off-stroke are not only extracted from its immediate preceding stroke and succeeding stroke but also from all the preceding strokes and succeeding strokes, which are highlighted in Table 2. A segmentation feature analysis in [11] shows that the feature extracted from both the forward and backward contexts are useful for character segmentation.

2.3 Candidate Lattice Construction

Employing the combination of on-line and off-line recognition methods for character recognition [12], 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 src-lattice as shown in Fig. 3, where each node denotes a candidate segmentation point and each arc denotes a character class assigned to a candidate character pattern.

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 4 shows them for the src-lattice with three SP off-strokes and four candidate character blocks.

2.4 Best-Path Search and Recognition

From a src-lattice, paths are evaluated by combining the scores of character recognition, geometric features, and linguistic context. The evaluation function for a sequence of m candidate character patterns $X = x_1, x_2, \ldots, x_m$ to be assigned to a sequence of character classes $C = c_1, c_2, \ldots, c_m$ is expressed as:

$$f(X, C) = \sum_{j=1}^{m} \left\{ \sum_{i=1}^{6} [\lambda_{i1} + \lambda_{i2}(k_i - 1)] \log P_h(x_i|S_{bi}) + \lambda_{i3} \log P(g_j|S_{bi}) \right\} + m \lambda$$

where $P_h(h = 1, 2, \ldots, 6)$ denote the probabilities of $P(c_i|x_{i-2}c_{i-1})$, $P(b_i|x_i)$, $P(q|x_i)$, $P(p^{p_i}|c_i)$, $P(p^{b_i}|c_{i-1}c_i)$, and $P(x|x_i)$, respectively; $k_i$ denotes the number of primitive segments contained in a candidate character pattern $x_i$. $P(x|x_i)$ is estimated by the score of the on-line and off-line combined recognizers. $P(c_i|x_{i-2}c_{i-1})$ is the tri-gram for linguistic

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Terms of features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbol</td>
<td>Definition of the symbol</td>
</tr>
<tr>
<td>$S_p$</td>
<td>Immediate preceding stroke</td>
</tr>
<tr>
<td>$S_t$</td>
<td>Immediate succeeding stroke</td>
</tr>
<tr>
<td>$B_p$</td>
<td>Bounding box of $S_p$</td>
</tr>
<tr>
<td>$B_t$</td>
<td>Bounding box of $S_t$</td>
</tr>
<tr>
<td>$B_{p,t}$</td>
<td>Bounding box of all preceding strokes</td>
</tr>
<tr>
<td>$B_{t,t}$</td>
<td>Bounding box of all succeeding strokes</td>
</tr>
<tr>
<td>$D_{p,t}$</td>
<td>Distance between $B_p$ and $B_t$ in x-axis</td>
</tr>
<tr>
<td>$D_{t,t}$</td>
<td>Distance between $B_t$ and $B_t$ in x-axis</td>
</tr>
<tr>
<td>$O_{p,t}$</td>
<td>Overlap area between $B_p$ and $B_t$</td>
</tr>
<tr>
<td>$O_{p,t}$</td>
<td>Overlap area between $B_{p,t}$ and $B_{t,t}$</td>
</tr>
<tr>
<td>$D_{p,t}$</td>
<td>Absolute distance of centers of $B_p$ and $B_t$ in x-axis</td>
</tr>
<tr>
<td>$D_{p,t}$</td>
<td>Average character size of text line</td>
</tr>
<tr>
<td>$D_{p,t}$</td>
<td>Distance between top of $B_{p,t}$ and top of $B_t$ in y-axis</td>
</tr>
<tr>
<td>$P$</td>
<td>Pattern of all strokes</td>
</tr>
<tr>
<td>$P_{sb}$</td>
<td>Sub-pattern of $S_p$ and $S_t$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2</th>
<th>Geometric features for character segmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature</td>
<td>Definition of the feature</td>
</tr>
<tr>
<td>F1</td>
<td>Passing time for off-stroke</td>
</tr>
<tr>
<td>F2</td>
<td>$D_{p,t} / acs$</td>
</tr>
<tr>
<td>F3</td>
<td>$D_{p,t} / width of B_p$</td>
</tr>
<tr>
<td>F4</td>
<td>$D_{p,t} / width of B_t$</td>
</tr>
<tr>
<td>F5</td>
<td>$D_{p,t} / acs$</td>
</tr>
<tr>
<td>F6</td>
<td>$D_{p,t} / height of B_p$</td>
</tr>
<tr>
<td>F7</td>
<td>$D_{p,t} / height of B_t$</td>
</tr>
<tr>
<td>F8</td>
<td>$D_{p,t} / acs$</td>
</tr>
<tr>
<td>F9</td>
<td>$O_{p,t} / (acs)^2$</td>
</tr>
<tr>
<td>F10</td>
<td>$O_{p,t} / (width of B_t * height of B_t)$</td>
</tr>
<tr>
<td>F11</td>
<td>$O_{p,t} / (acs)^2$</td>
</tr>
<tr>
<td>F12</td>
<td>$D_{p,t} / acs$</td>
</tr>
<tr>
<td>F13</td>
<td>$D_{p,t} / acs$</td>
</tr>
<tr>
<td>F14</td>
<td>$D_{p,t} / acs$</td>
</tr>
<tr>
<td>F15</td>
<td>Length of off-stroke / acs</td>
</tr>
<tr>
<td>F16</td>
<td>Sine value of off-stroke</td>
</tr>
<tr>
<td>F17</td>
<td>Cosine value of off-stroke</td>
</tr>
<tr>
<td>F18</td>
<td>$F_2 / maximum F_2$ in text</td>
</tr>
</tbody>
</table>
context. \( P(b_i | c_i) \), \( P(q_j | c_i) \), \( P(p_b^j | c_i) \) and \( P(p_h^j | c_i) \) are geometric contexts, where \( b_i \), \( q_j \), \( p_b^j \) and \( p_h^j \) are character pattern size, inner gap, single-character position and pair-character position, respectively. \( P(g_j | Sb) \) is the probability that spacing between character patterns \( (Sb) \) appears as a spacing feature vector \( g_j \). \( P(g_j | Sw) \) is the probability that spacing within a character pattern \( (Sw) \) appears as a spacing feature vector \( g_j \). The details are shown in Gao et al. [16]. Each of the geometric contexts is assumed to be a normal distribution and its logarithm is modeled by a quadratic discriminant function, which can be trained using training patterns. The parameters \( \lambda_{1b} \), \( \lambda_{2b}(b = 1, \ldots, 7) \) and \( \lambda_{1w} \) are trained by the minimum classification error (MCE) algorithm [13].

By applying the Viterbi algorithm, the optimal path which has the highest evaluation score is found. Text recognition result is obtained from this path.

3. 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.

Both of the methods in [7] and [8] proposed incremental segmentation and recognition of handwritten text. For incremental segmentation, they determine the segmentation of the latest stroke based on the previously segmented sequence of strokes. This is unrecoverable, however, if wrong segmentation is made without knowing strokes coming hereafter. To deal with the problem, we present the resuming strategy, which allows the change of segmentation and recognition of previously written strokes due to strokes coming in future.

3.1 Resuming Strategy

The context at the current time step in incremental recognition presented so far is limited to forward direction, since the future input is not available. As batch recognition could employ the context in both forward and backward direction for segmentation and recognition, we need to develop a method which allows backward context in incremental recognition.

Semi-incremental recognition performs recognition process after receiving \( N_t \) newly written strokes, where \( N_t \) denotes the number of strokes to trigger incremental recognition. Ideally, we only have to process the newly received strokes and update the src-lattice. In fact, the backward context of the newly added strokes affects the segmentation and recognition of a small number of strokes previously received. Therefore, to allow the effect of the backward context, we extend the incremental processing to the section that includes these strokes and the newly received strokes. We call it "scope".

We advance segmentation and recognition as new strokes are input by resuming segmentation from a range of recent input strokes since their segmentation may change due to the newly added strokes. We proceed with the pointer being slightly behind the latest input stroke up to which the result of segmentation and character recognition is considered stable and fixed.

Newly added strokes may affect the segmentation and recognition in the current scope. Change of segmentation leads to change of character recognition. As each change in classification of an off-stroke induces change in recognition of primitive character segments connecting to that off-stroke, we must update the scope to include those primitive character segments.

As for the 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.

3.2 Processing Flow

From the previously described strategy, Fig. 5 shows the processing flow of the semi-incremental recognition method.

First, we receive new strokes. Secondly, we resume segmentation. Thirdly, we determine the scope. 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.

3.3 Resumption of Segmentation

First, we determine the pointer to resume segmentation named \( \text{Seg}_r \). It must be determined so that the segmentation and recognition of its preceding strokes is stable. In 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 an \( \text{SP} \) with confirmation of the best path search. Since the effect of backward context of newly added strokes to the segmen-
tation and recognition of preceding strokes reduces as they are far away from the latest stroke in backward direction, the segmentation at an off-stroke between two recognized characters far enough from the latest stroke could be fixed. We determine Seg \_rp among those off-strokes based on the number of characters from each off-stroke to the last character in the recognition result. If this number equals to a predefined parameter named \( N_{\text{seg}} \), that off-stroke will be determined as a new Seg \_rp. \( N_{\text{seg}} \) is defined as a fixed number of characters required to determine a new Seg \_rp.

3.4 Fixation of SP Off-Strokes from UP Off-Strokes

Determination of off-strokes to SP off-strokes has large effect to the recognition rate and performance of the system. Although SP off-strokes are detected by SVM in the segmentation process, the performance of 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 off-strokes (UPs in short) to SP off-strokes (SPs in short). We call this process UP fixation. UPs between recognized characters, before the latest \( N_{\text{seg, det}} \) characters in the recognition result are determined as SPs. Here, \( N_{\text{seg, det}} \) denotes a pre-defined constant for the minimum number of characters that follow an UP off-stroke to make it a stable SP off-stroke. Generally, \( N_{\text{seg, det}} \) is smaller than or equal to \( N_{\text{seg}} \).

3.5 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.

3.6 Bounded Waiting Time

Since we resume segmentation from Seg \_rp, segmentation of the off-strokes before Seg \_rp remains unchanged. Thus, we only need to compare the results of segmentation before and after new strokes are added from Seg \_rp to the latest stroke.

3.7 An Example of the Processes

Figure 6 shows an example to determine the scope. Assume the latest scope with segmentation and text recognition.
tion result in Fig. 6(a). Then, the new strokes marked red are added. We update Seg_{rp} and apply segmentation from the updated Seg_{rp} (Fig. 6(b)). Next we change UPs to SPs by UP fixation and find the earliest classification-changed off-stroke (Fig. 6(c)). Finally, we locate the character block including or just preceding this off-stroke and update the scope (Fig. 6(d)).

3.8 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 SPs and splits candidate character blocks by these SPs. 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 7(a) represents the lattice blocks of the previous scope, when new strokes are added as shown in Fig. 7(b), classification of the two off-strokes shown in red in the updated scope is changed to SP. From these SP off-strokes, 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 two candidate character patterns (shown in gray) are rebuilt due to the new strokes.

3.9 Skipping Partial Patterns

Recognition of partial character patterns can be postponed until the complete character patterns are received. Therefore, we skip recognizing them to reduce CPU time. We treat candidate character patterns containing the last primitive segment as partial candidate character patterns (PPs) until a new primitive segment is detected or the recognition is requested. We call this process PP skip.

3.10 Handling of Delayed Strokes

The batch recognition system is not designed to handle delayed strokes [5], since the segmentation of strokes is based on writing order. To make the segmentation of a text line including them correctly, however, we first detect delay strokes and ignore them in the segmentation process. Then, we determine a segmented block for each delayed stroke to merge the delayed stroke into it. Finally, we rebuild the src-lattice.

Delayed strokes are detected using the previous recognition result. Firstly, we retrieve the bounding box for each recognized character from the segmentation-recognition result of the previous scope. Then, we determine each newly added stroke as a delayed stroke if it is close to the previous bounding boxes rather than the latest bounding box. When delayed strokes occur, we rebuild the src-lattice in two steps: first we build the src-lattice without delayed strokes, second we put delayed strokes into appropriate primitive segments and rebuild the candidate character patterns containing the delayed strokes.

3.11 Resuming Best-Path Search and Recognition

From the first character lattice block in current scope, we resume the best-path search and get text recognition result. Resuming the best-path search at each incremental processing sets bounds to the processing time and waiting time. This solves the drawback of the method [7], in which the processing time for the best-path search is prolonged as the number of characters increases.

4. Experiments

4.1 Measures for Evaluation

First, over-segmentation is applied then segmentation is determined along with character recognition and best-path search. The over-segmentation process classifies each off-stroke as an SP, NSP, or UP off-stroke. An UP off-stroke can then be further classified as an SP or NSP in the text recognition process.

Let #SP, #NSP, #UP are the numbers of returned SPs, NSPs and UPs, respectively. #SP_r is the number of correctly classified SPs among the returned SPs. #SP_t is the number of true SPs defined in the ground truth. #UP_t is the number
of UPS being true SPs.

The performance of over-segmentation is evaluated with the following measures.

**Precision** ($p$):

\[
p = \frac{\#SP_{c}}{\#SP_{t}} \tag{2}
\]

**Recall** ($r$):

\[
r = \frac{\#SP_{c} + \#UP_{c}}{\#SP_{t}} \tag{3}
\]

Inclusion of $UP_{c}$ in the dividend is typical for over-segmentation since UPS maintain the possibility that they will be classified correctly.

The F-measure ($f$) is calculated as follows:

\[
f = \frac{2 \times p \times r}{(p + r)} \tag{4}
\]

Although UPS maintain the possibility that they will be classified correctly, thus increase recall, leaving many UPS instead of SPs or NSPs, however, incurs more waiting time as analyzed in Sect. 3.4. Therefore, we evaluate the detection rate ($d$) of over-segmentation as ability of determining more SPs instead of UPS by the following formula:

\[
d = \frac{\#SP}{(\#SP + \#UP)} \tag{5}
\]

As final segmentation is determined from the result of the best-path search, we get SPs as off-strokes between two recognized characters and the remaining are NSPs. Let $\#SP_{f}$, $\#SP_{fc}$, and $\#SP_{f1}$ are the number of returned SPs in final segmentation, the number of correctly classified SPs among those returned SPs and the number of true SPs in the ground truth, respectively. The F-measure of final segmentation ($F$) denoted as the segmentation measure is evaluated as follows:

\[
F = \frac{2 \times P \times R}{(P + R)} \tag{6}
\]

where $P$ and $R$ are the precision and recall of final segmentation, respectively. They are defined as follows:

\[
P = \frac{\#SP_{fc}}{\#SP_{f}} \tag{7}
\]

\[
R = \frac{\#SP_{fc}}{\#SP_{f1}} \tag{8}
\]

### 4.2 Setup for Experiments

We trained the character recognizer and geometric scoring functions using Japanese on-line handwriting database Nakayosi[14]. We employed a trigram table extracted from the year 1993 volume of the Asahi newspaper and the year 2002 volume of the Nikkei newspaper to model linguistic context. For training the weight parameters of the evaluation function (1) and evaluating the performance of text recognition, we used horizontally written text line patterns extracted from the TUAT-Kondate database collected from 100 people [15]. We separated the text lines into 4 sets by writers and then used 3 sets (10,174 text lines written by 75 people) for training and 1 set (3,511 text lines written by 25 people) for testing. We changed the role four times and took the average. We used this separation to assure writer independence and conducted cross validation to evaluate the effect unbiased to data sets.

The parameters of the evaluation function in Eq. (1) have been trained using each training set, but $N_{e}$ and $N_{seg}$ are not trained since $N_{e}$ and $N_{seg}$ are control variables rather than parameters.

We implemented a handwritten Japanese text recognition system using our semi-incremental recognition method. We used the batch recognition system for Japanese [5] without modification. We ran all the systems on an Intel(R) Core(TM) i7 CPU 870@ 2.6Ghz with 4-GB memory.

### 4.3 Character Recognition Rate

Figure 8 shows the character recognition rate (i.e., the number of correctly recognized characters over that of all the characters in handwritten text) by the semi-incremental recognition method, including the case of pure-incremental recognition ($N_{e}=1$) in comparison with the batch recognition method. To evaluate the effect of resuming segmentation to the character recognition rate, we made experiments with respect to $N_{e}$ (from 1 to 10) and $N_{seg}$ (from 4 to 25). For each $N_{seg}$, the character recognition rate increases as $N_{e}$ increases from 1 to 10. With larger $N_{seg}$ the average recognition rate increases, since the segmentation and recognition are resumed from a more stable Seg fp. As $N_{seg}$ approaches 20 and 25, the recognition rate reaches the performance of batch recognition without large dependence on $N_{e}$ (rates with $N_{e}=10$ and those with $N_{e}=1$ are nearly the same). The maximum recognition rate is 93.26% with $N_{seg}=20$, which is nearly the same as that of the batch recognition method, i.e., 93.27%. The case of pure-incremental recognition ($N_{e}=1$) degrades the recognition rate seriously with

![Fig. 8 Recognition rate with respect to $N_{seg}$.](image)
small values for $N_{seg}$ (0.8 point when $N_{seg} = 4$), but decreases the degradation as $N_{seg}$ is set larger (0.01 point with $N_{seg} \geq 20$). For $N_{seg} > 20$, the recognition rate does not increases but the waiting time increases as discussed in Sect. 4.4.

4.4 Waiting Time

The evaluation was done on five 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. We evaluate the waiting time by the semi-incremental recognition method from two points, dependency on $N_{seg}$ and that on $N_s$.

As for the first point, Fig. 9 shows the average waiting time of semi-incremental recognition with respect to $N_{seg}$ when $N_s = 1$ and $N_s = 10$. The waiting time increases as $N_{seg}$ increases regardless of $N_s$. As mentioned above, the recognition rate saturates for $N_{seg} > 20$. Therefore, hereafter we consider $N_{seg}$ up to 20.

As for the second point, we measured the average waiting time with respect to $N_s$ while $N_{seg}$ was fixed to 20 including the case of pure-incremental recognition ($N_s = 1$). We evaluated the effectiveness of reusing the src-lattice as well as applying UP fixation and PP skip.

Figure 10 shows the average waiting time of recognizing the five pages of Japanese text in comparison with the batch recognition method which takes 1,479ms in average for the waiting time. By reusing the src-lattice, the average waiting time is significantly reduced from roughly 150ms to 30ms. Applying UP fixation and PP skip further reduces the waiting time. The waiting time by the semi-incremental method with applying all three methods is less than 50 ms, which is small enough to be unnoticeable by users. Pure-incremental recognition ($N_s = 1$) incurs the smallest waiting time since there is only one new stroke at each incremental processing.

Although the waiting time by the batch recognition method increases as text becomes longer, that by the semi-incremental recognition method is bounded, regardless of the length of the text. Figure 11 shows the waiting time of recognizing the first sample page (with 347 strokes) by the semi-incremental recognition method ($N_s = 1, 5, 10$ while $N_{seg}$ is fixed to 20) and the batch recognition method, while the number of strokes increases from 1 to 100. The case of $N_s = 1$ denotes pure-incremental recognition. As the number of strokes increases, the waiting time of the batch recognition method gradually increases to 310ms, while those of semi-incremental and pure-incremental recognition are bounded to less than 150ms.

4.5 CPU Time

To evaluate processing time, average CPU time per stroke is shown in Table 3. Compared with pure incremental recognition ($N_s = 1$), the semi-incremental recognition method with $N_s > 2$ save up to 53.16% of CPU time. Although the semi-incremental recognition method incurs more CPU time than the batch recognition method, it requires less CPU time as $N_s$ increases. As for $N_s \geq 7$, CPU time of the semi-incremental method approaches to CPU time of the batch recognition method.

4.6 Effect of Resuming Segmentation and UP Fixation

While the batch recognition method applies segmentation for the entire handwritten text, the semi-incremental recognition method resumes segmentation from the Seg_rp. The first experiment was to confirm that the resumption of segmentation does not degrade the segmentation performance if we set appropriate values for the parameters. Moreover, the performance of the semi-incremental method depends on whether we apply UP fixation or not.

Table 4 lists the F-measures of over-segmentation. The semi-incremental method without applying UP fixation performs the same as the batch recognition method. Applying...
Table 3  Processing time per stroke.

<table>
<thead>
<tr>
<th>Semi-incremental - (N_s)</th>
<th>Batch</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (pure incremental)</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>6.23</td>
</tr>
<tr>
<td>3</td>
<td>5.01</td>
</tr>
<tr>
<td>4</td>
<td>4.91</td>
</tr>
<tr>
<td>5</td>
<td>4.28</td>
</tr>
<tr>
<td>6</td>
<td>4.12</td>
</tr>
<tr>
<td>7</td>
<td>3.89</td>
</tr>
<tr>
<td>8</td>
<td>4.00</td>
</tr>
<tr>
<td>9</td>
<td>3.94</td>
</tr>
<tr>
<td>10</td>
<td>3.78</td>
</tr>
</tbody>
</table>

Table 4  F-measures of over-segmentation (%)

<table>
<thead>
<tr>
<th>Semi-incremental recognition</th>
<th>Batch recognition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without UP fixation</td>
<td>UP fixation</td>
</tr>
<tr>
<td>99.79 ± 0.02</td>
<td>99.41 ± 0.05</td>
</tr>
</tbody>
</table>

Fig. 12  Detection rate of over-segmentation.

Fig. 13  Final segmentation measure.

5. Conclusion

In this paper, we presented a semi-incremental recognition method for on-line handwritten Japanese text. By resuming the segmentation and recognition in a local scope, the method reduces the waiting time to be small enough to be unnoticeable by users. Moreover, determining SP off-strokes based on recognition result shortens block lengths and bounds the waiting time. Skipping the recognition of partial patterns and reusing recognized character patterns in the src-lattice are also shown to be effective in reducing the waiting time.

The control variables \(N_s\) and \(N_{seg}\) should be set according to the environments. As far as they are set as shown in the experiments, the semi-incremental recognition method is clearly superior to the batch recognition method in the waiting time while maintaining the recognition rate. It also excels pure incremental recognition in the character recognition rate and the total CPU time.

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

References

Cuong-Tuan Nguyen was born on 21 March 1988 in Vietnam. He received his B.Eng. degree in Computer Engineering and M.Sc. degree in Computer Science from Ho Chi Minh City University of Technology and Tokyo University of Agriculture and Technology (TUAT), respectively. Since April 2014 he is a Ph.D. student in Department of Electronic and Information Engineering at TUAT. He is currently working on handwritten text recognition of Japanese and English text.

Bilan Zhu received M.Sc. degree from Tokyo University of Agriculture and Technology (TUAT), Japan in 2004. In 2007, she received Ph.D. degree in Engineering from TUAT. She is now an Assistant Professor at TUAT. She is working on on-line, off-line handwriting recognition, documents processing and context analysis of Japanese, Chinese and English text.

Masaki Nakagawa was born on 31 October 1954 in Japan. He received his B.Sc. and M.Sc. degrees from the University of Tokyo in 1977 and 1979, respectively. During the academic year 1977/78, he enrolled in the Computer Science course at Essex University in England and received the M.Sc. with distinction in Computer Studies in July 1979. He received his Ph.D. in Information Science from University of Tokyo in December 1988. He has been working at Tokyo University of Agriculture and Technology since April 1979. He is currently a Professor of Media Interaction in the Department of Computer and Information Sciences.