International Journal on Document Analysis and Recognition (IJDAR) (2020) 23:53–72
https://doi.org/10.1007/s10032-019-00343-y

ORIGINAL PAPER

A unified method for augmented incremental recognition of online handwritten Japanese and English text

Cuong Tuan Nguyen[1] · **Bipin Indurkhya**[2] · **Masaki Nakagawa**[1]

Received: 20 January 2018 / Revised: 9 July 2019 / Accepted: 28 August 2019 / Published online: 5 September 2019

Abstract
We present a unified method to augmented incremental recognition for online handwritten Japanese and English text, which is used for busy or on-the-fly recognition while writing, and lazy or delayed recognition after writing, without incurring long waiting times. It extends the local context for segmentation and recognition to a range of recent strokes called "segmentation scope" and "recognition scope," respectively. The recognition scope is inside of the segmentation scope. The augmented incremental recognition triggers recognition at every several recent strokes, updates the segmentation and recognition candidate lattice, and searches over the lattice for the best result incrementally. It also incorporates three techniques. The first is to reuse the segmentation and recognition candidate lattice in the previous recognition scope for the current recognition scope. The second is to fix undecided segmentation points if they are stable between character/word patterns. The third is to skip recognition of partial candidate character/word patterns. The augmented incremental method includes the case of triggering recognition at every new stroke with the above-mentioned techniques. Experiments conducted on TUAT-Kondate and IAM online database show its superiority to batch recognition (recognizing text at one time) and pure incremental recognition (recognizing text at every input stroke) in processing time, waiting time, and recognition accuracy.

Keywords Online recognition · Handwriting recognition · Batch recognition · Incremental recognition

1 Introduction

Due to the development of pen-based and touch-based devices, such as tablets, smart-phones and digital pens, there has been a renewed interest in online handwriting recognition, which provides a practical input method for devices without a keyboard [7, 22]. Since hand-held devices have relatively smaller CPU performance for less power consumption compared with desktop PCs, and they are interactive devices, handwriting recognition on these devices must respond to the user input with a high recognition rate but without incurring much CPU time.

Compared to isolated character or word recognition, online handwritten text recognition faces the problem of word segmentation or character segmentation. There are two approaches to segmentation. One is implicit segmentation, which has been extensively studied in recent years, and the other is explicit segmentation. High performance with implicit segmentation is reported for English [2, 9] but not yet for Japanese or Chinese online handwriting text recognition. Owing to the progress in deep neural network technology, one can consider deploying it for practical systems, but there are some obstacles such as speed and memory space for the large category size to be used in stand-alone systems, especially for hand-held mobile phones and tablets. On the other hand, the explicit segmentation technique also provides reliable performance in recognition of online handwritten Japanese text [31], and online handwritten Chinese text [27]. This approach is also applied for online handwritten English text recognition [19, 23]. It first applies segmentation to separate the whole text line into characters or words, then recognizes each separated patterns, and finally

✉ Cuong Tuan Nguyen
ntcuong2103@gmail.com

Bipin Indurkhya
bipin.indurkhya@uj.edu.pl

Masaki Nakagawa
nakagawa@cc.tuat.ac.jp

[1] Department of Computer and Information Sciences, Tokyo University of Agriculture and Technology, 2-24-16 Naka-cho, Koganei-shi, Tokyo 184-8588, Japan

[2] Institute of Philosophy, Jagiellonian University, Kraków, Poland

concatenates the results to get the text line recognition result. Segmentation is based on geometric layout features (e.g., gap between strokes, stroke histogram and inter-relationship, where a stroke is a sequence of finger-tip or pen-tip coordinates from finger/pen-down to finger/pen-up). In order to solve ambiguity in segmentation, soft decision is often employed for segmentation and recognition, and all the candidates of segmentation and recognition are represented in the segmentation and recognition candidate lattice. This approach is also called "segmentation by recognition," or "over-segmentation," since it nominates true segmentation points exhaustively, thereby excessively over segmenting a character or a word pattern. Each segment in the lattice is recognized, and then text recognition result is produced by searching the lattice for the highest score path, taking into account the geometric context, the linguistic context and the recognition scores. Excessively segmented patterns are combined in the above best-path search in the lattice.

The context of the input sequence (global context) is important for handwritten text recognition. Zhu et al. [31] showed the effectiveness of the geometric features extracted from all the preceding or succeeding strokes for segmentation of handwritten Japanese text. Nakagawa et al. [16] improved segmentation and recognition by applying geometric and linguistic contexts. Graves et al. [2] used bi-directional recurrent neural networks to integrate the context from both forward and backward directions of an input sequence for recognizing English handwritten text.

There are basically two methods to trigger recognition. The batch recognition method, which recognizes handwritten text after the user has finished writing, can easily use the full context to achieve a high recognition rate. For Japanese, Zhu et al. [31] reported on a batch recognition method that integrates segmentation and recognition, resulting in a high recognition rate. However, if all the processes for segmentation and recognition are executed after the entire text is written, a long waiting time is incurred: the more the written text, the longer the waiting time. The other method is the incremental recognition method [24, 25], which recognizes the handwritten characters incrementally as the user is writing. Tanaka et al. [24] proposed an incremental recognition method for online Japanese handwriting recognition. Wang et al. [25] presented a method for real-time (incremental) recognition of Chinese handwritten text. With these methods, the candidate characters are generated and recognized to assign candidate classes whenever a new stroke is produced. The problem of waiting time is solved by the incremental recognition method, which, however, may degrade the recognition rate due to a lack of global context in its local processing of input sequence. Tanaka et al. [24] reported that incremental recognition method degrades 0.3 points of the recognition rate as compared with batch recognition method. Due to repeated processing after receiving every stroke, it also increases the total CPU time required for recognition, as reported by Wang et al. [25]. Not only the recognition processes are triggered repeatedly, but also attempts are made to recognize incomplete patterns after every stroke. Therefore, it takes a substantial amount of CPU time for recognizing a long input stroke sequence. Moreover, these two methods apply the best-path search from the beginning to the end of the input sequence, whenever the user requests the recognition result. This extends the waiting time when there are many strokes in the input sequence.

There are also two alternatives for the user interface of handwritten text recognition: busy or on-the-fly recognition and lazy or delayed recognition [14]. A busy recognition interface shows the recognition result while the user is writing. It gives immediate feedback to the user, but the user might be bothered by having to confirm or correct the recognition. A predictive input interface [11], which predicts a character or word from a few beginning strokes, may be categorized as a busy recognition interface. On the other hand, a lazy recognition interface delays the output of the recognition result until needed. It is suitable for a user who is writing while thinking. The user does not need a recognition result when writing and only needs the recognized text after he/she stops writing.

A lazy recognition interface can be implemented straightforwardly with the batch recognition method. Due to the problem of waiting time, however, the incremental recognition method should be run in the background when a user is writing even for a lazy recognition interface when the problem of waiting time is serious.

As stated above, it is effective to use the full context, both in the forward and the backward directions, for text recognition. This can be easily achieved with the batch recognition method but not with the incremental recognition method, since succeeding strokes are not available. The method by Zhu et al. [31] involves bi-directional geometric context for segmentation, where each off-stroke (a vector from finger/pen-up to finger/pen-down) is classified using the features extracted from both its preceding and succeeding strokes. On the other hand, the incremental recognition methods by Tanaka [24] for Japanese text and by Wang et al. [25] for Chinese text use only the features extracted from the current off-stroke and its preceding strokes for segmentation. This limits recognition performance since the backward context is not used. To use backward context, we should provide a way in which succeeding strokes affect the recognition of previous strokes.

In this work, we aim to overcome these drawbacks and combine the advantages of both the batch and the incremental recognition methods. We focus on maintaining global context in incremental recognition and triggering recognition after every several strokes. We refer to this solution as the semi-incremental recognition method, while calling

the method of triggering recognition at every stroke a pure incremental method. So far, all current incremental recognition systems are classified as pure.

Since we proposed a semi-incremental recognition method for online handwritten Japanese text recognition [18] and for English text [19], we revised and introduced three techniques to improve the performance: reusing the segmentation and recognition candidate lattice in the previous incremental stage for the current stage; fixing undecided segmentation points if they are stable between character patterns; and skipping recognition of partial candidate character patterns for Japanese [20]. These three techniques are also effective for pure incremental recognition.

This paper combines the incremental recognition methods for Japanese and English into a unified method for both languages by incorporating the three techniques mentioned above. We refer to this method as "augmented incremental recognition" because it incorporates the three techniques and it triggers recognition for every several recent strokes including the case of pure incremental recognition, i.e., triggering recognition at every new stroke. We present experimental evidence here to show that augmented incremental recognition with an appropriate size of global context maintains as high a recognition rate as batch recognition, incurs little waiting time and decreases the total CPU time even for the case of pure incremental recognition.

The rest of this paper is organized as follows. The baseline batch recognition method is summarized in Sect. 2. The augmented incremental recognition method is presented in Sect. 3. Experiments on the augmented incremental recognition method are described in Sect. 4, and the conclusions are presented in Sect. 5.

2 Overview of batch recognition method

This section introduces the batch recognition method following the explicit segmentation approach for handwritten text recognition. After all the strokes are input, the method employs soft decision for segmentation to create and build the segmentation and recognition candidate lattice and then determines the correct segmentation and recognition using the best-path search. This method has been applied for Japanese text [31] and English text [19].

2.1 Processing flow

Figure 1 shows the flow of the batch recognition method. First, the segmentation process separates handwritten text into text lines and then segments each text line into primitive segments, which are characters or parts of a character for Japanese text or words or parts of a word for English text. Second, a lattice is built by recognizing primitive segments.

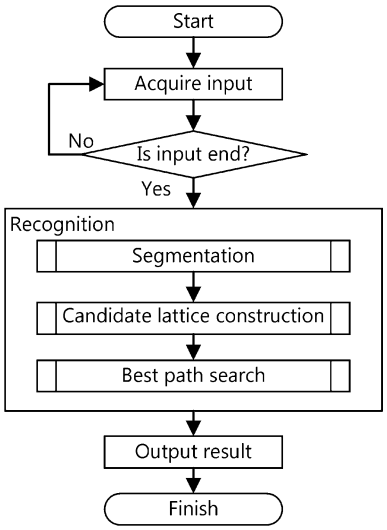


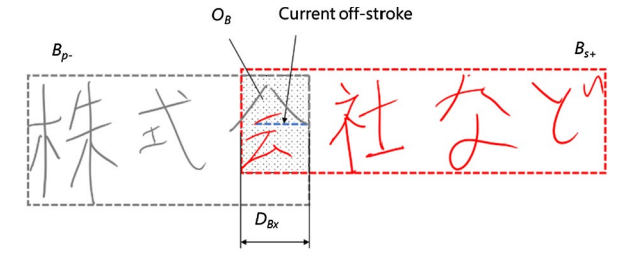
Fig. 1 Flow of batch recognition

Finally, the lattice is searched for the best path to obtain the recognition result.

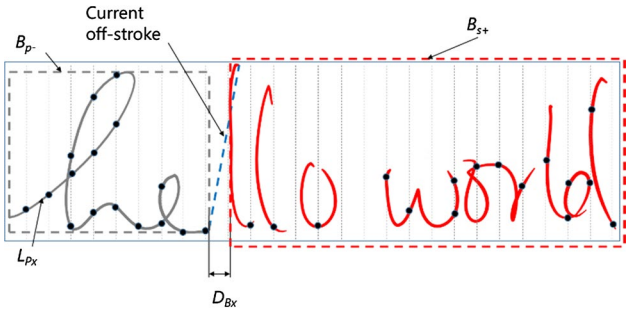
2.2 Segmentation

The segmentation process includes two stages: line segmentation and character or word over-segmentation. In line segmentation, the whole text is segmented into text lines by the method of Zhou et al. [28] for Japanese or by linear regression for English [19]. In the second stage, each segmented line is over-segmented into characters or parts of a character for Japanese [31], or words or parts of a word for English [17].

For character or word over-segmentation, we use a classifier to classify each off-stroke into three classes: segmentation point (SP), non-segmentation point (NSP) and undecided point (UP) according to geometric features. The features for segmentation include those extracted from the current off-stroke and both of its preceding and succeeding strokes, which are global features. Examples of the global features for Japanese text and English text are shown in Fig. 2. The supervised labels for training are determined as follows: an SP separates two characters or two words at the off-stroke, while an NSP indicates that the off-stroke is within a character or within a word. An off-stroke between two text lines is treated as an SP. The classifier is trained to predict an off-stroke being SP or NSP. When classifying an off-stroke, if the confidence level is low, it is treated as an UP, indicating that it could be an SP or an NSP. The final classification of UPs is determined in the later processes. For the off-stroke classification, we apply a support vector machine (SVM) classifier for Japanese text [1] and a bi-directional long short-term memory (BLSTM) [3] for



(a) Japanese text.



(b) English text.

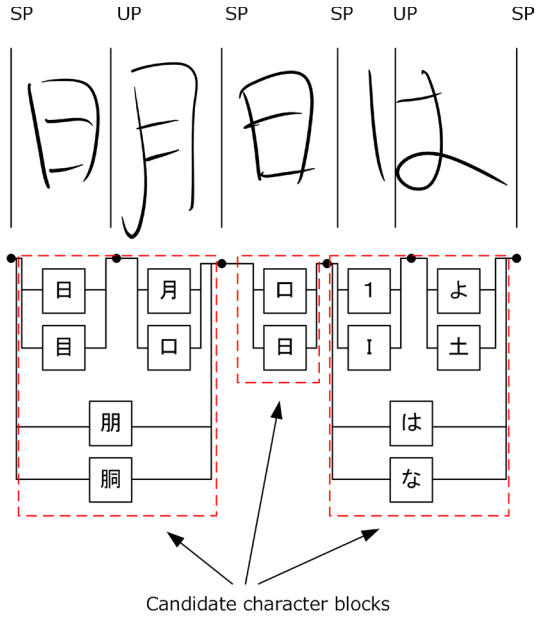
Fig. 2 Segmentation features for **a** Japanese text and **b** English text. B_{p-}, bounding box of all preceding strokes; B_{s+}, bounding box of all succeeding strokes. OB, overlap of B_{p-} and B_{s+}, D_{Bx}, distance in x axis, L_{Px}, average stroke length over x axis

English text [19]. While SVM classifies each off-stroke based on the features at the off-stroke alone, BLSTM, as a type of recurrent neural networks, integrates the features from both the preceding and the succeeding off-strokes for classification.

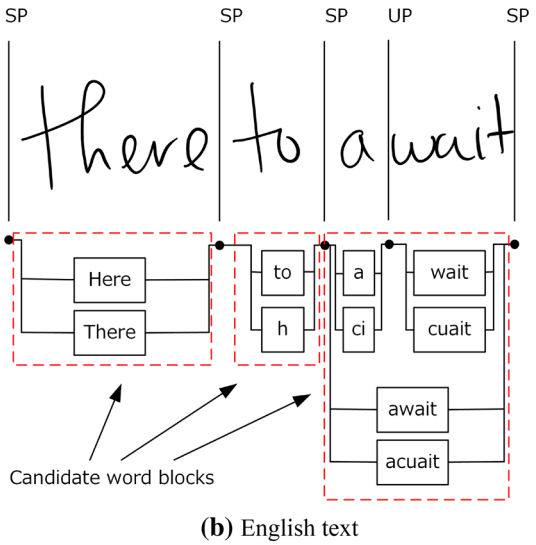
2.3 Construction of segmentation and recognition candidate lattice

We call a subsequence of strokes delimited by SP or UP off-strokes a primitive segment, which could be a character/word or a part of a character/word. Therefore, a primitive segment and consecutive primitive segments beside a UP form candidate character patterns or candidate word patterns. All the candidate character/word patterns are represented in a segmentation candidate lattice.

Each candidate character/word pattern in a segmentation candidate lattice is recognized, and a number of candidate classes with confidence scores are associated with each candidate pattern in the lattice. Then, all the possible segmentations and recognition candidate classes are represented in the lattice. We call this lattice segmentation and recognition candidate lattice or src-lattice in short. In src-lattice, we define candidate character/word blocks, each of which represents a sub-lattice of all the candidate character/word patterns separated by two adjacent SP off-strokes. Figure 3a, b shows, respectively, an example of src-lattice for Japanese



(a) Japanese text



(b) English text

Fig. 3 Segmentation-recognition candidate lattices for **a** Japanese text and for **b** English text

text and another for English text, where each node denotes a candidate segmentation point and each arc denotes a character class for Japanese text (a) or a word class for English text (b) assigned to a candidate character/word pattern. Note that a single candidate character/word block may result in two or more characters/words.

2.4 Best-path search and recognition

From an src-lattice, paths are evaluated by combining the scores of character/word recognition, geometric features and linguistic context [26, 31]. We apply the Viterbi algorithm

to search for the optimal path that has the highest evaluation score and obtain the text recognition result.

For evaluating a path through a sequence S of m primitive segments $S = s_1, s_2, \ldots, s_m$ of an input sequence X, forming a sequence of n candidate character/word patterns $Z = z_1, z_2, \ldots, z_n$ which is assigned as $C = c_1, c_2, \ldots, c_n$, we have the posterior probability as follows:

$$P(C|X,S,Z) = \frac{P(X,S,Z|C)P(C)}{P(X,S,Z)} = \frac{P(S|X,Z,C)P(X,Z|C)P(C)}{P(X,S,Z)} \tag{1}$$

We omit the class-independent denominator to obtain the following formula:

$$P(C|X,S,Z) \propto P(S|X,Z,C)P(X,Z|C)P(C) \tag{2}$$

From the posterior probability, we obtain the evaluation function as:

$$f(X,S,Z,C) = \log P(S|X,Z,C) + \log P(X,Z|C) + \log P(C) \tag{3}$$

The term $P(X,Z|C)$ is the probability of having the input sequence X to form the n candidate character/word pattern sequence Z when C is intended. It is approximated from geometric features and recognition scores of single characters or words [1, 19].

The linguistic context probability $P(C)$ is estimated using a trigram language model with back-off weight:

$$P(C) = \prod_{i=1}^{n} P(c_i|c_{i-2}c_{i-1}) \tag{4}$$

We assume that the segmentation probability $P(S|X,Z,C)$ does not depend on character/word classes C, and it is approximated by the score from a segmentation classifier at each candidate segmentation point d_j (SP or UP) between two primitive segments s_j and s_{j+1}:

$$P(S|X,Z,C) = \prod_{j=1}^{m-1} P(d_j|X,Z) \tag{5}$$

Each candidate segmentation point d_j could be an off-stroke between character/word patterns or an off-stroke within a character/word pattern.

$$P(S|X,Z,C) = \prod_{j=\overline{1,m-1};T(d_j)=B} P_{\mathrm{sp}}(d_j) \times \prod_{j=\overline{1,m-1};T(d_j)=W} P_{\mathrm{nsp}}(d_j) \tag{6}$$

where T denotes the labeling function outputting the off-stroke type (B: between, W: within) for a candidate segmentation point. $P_{\mathrm{sp}}(d_j)$ and $P_{\mathrm{nsp}}(d_j)$ are the classification probabilities of an off-stroke being classified as SP and NSP, respectively.

The evaluation function is expressed as:

$$f(X,S,G,C) = \sum_{i=1}^{n} \left\{ \sum_{h=1}^{6} \left[\lambda_{h1} + \lambda_{h2}\left(k_i - 1\right)\right] \log P_h \right\} + \lambda_{71} \sum_{j=\overline{1,m-1};T(d_j)=B} \log P_{\mathrm{sp}}(d_j) + \lambda_{72} \sum_{j=\overline{1,m-1};T(d_j)=W} \log P_{\mathrm{nsp}}(d_j) + n\lambda \tag{7}$$

where $P_h (h = 1, \ldots, 6)$ denote the probabilities of language model $P\left(c_i|c_{i-2}c_{i-1}\right)$, geometric $P\left(b_i|c_i\right)$, $P\left(q_i|c_i\right)$, $P\left(p_i^u|c_i\right)$, $P\left(p_i^b|c_{i-1}c_i\right)$, and recognition $P_r\left(r_i|c_i\right)$, respectively, k_i denotes the number of primitive segments contained in the candidate character pattern z_i. For Japanese text, the weighting parameters $\lambda_{h1}, \lambda_{h2} (h = \overline{1,7})$ and λ are selected using a genetic algorithm to optimize the text recognition performance on a training dataset. For English text, we use a simpler form of the formula by setting $\lambda_{h1} = 0$ for $h = \overline{1,6}$, using the same parameter for λ_{71}, λ_{72} and setting $\lambda = 0$.The parameters are optimized by the minimum classification error (MCE) algorithm [13] on a training dataset.

Let *Node*(i, j) represent recognition data of the character/word candidate pattern spanning primitive segments from s_i to s_j, *SubNode*(i, j, k) represent the kth-recognized candidate of *Node*(i, j). Each *Node*(i, j) has its own candidate character/word pattern z. Each *SubNode*(i, j, k) has its own character/word recognition result c and holds records of the best segmentation path Z and recognition path C. Algorithm 1 shows the pseudocode for searching the best path through the lattice by Viterbi algorithm. For each time step j of the primitive segment s_j, we build all the *Node*(i, j) start from $i =$ *GetFirstSegment*(j) as the first segment of the character/word candidate block containing s_j. The best path to each *SubNode*(i, j, k) is collected at each time step j by *NodeCollect*(j).

Algorithm 1. Lattice best path search by Viterbi algorithm.

Input:

Input sequence X

Primitive segments S

Initialization:

for k=1 **to** Node(1,1).candidates **do**

$SubNode(1, 1, k).C = SubNode(1, 1, k).c$

$SubNode(1, 1, k).Z = Node(1, 1).z$

add $SubNode(1, 1, k)$ to $NodeCollect(1)$

Algorithm:

for $j = 2$ **to** m **do**

for $i = GetFirstSegment(j)$ **to** j

for $k = 1$ **to** $Node(i, j)$.candidates

$best\_score = 0$

foreach $PrevSubNode$ **in** $NodeCollect(i)$ **do**

Z = Concatenate($PrevSubNode.Z$, $Node(i, j).z$)

C = Concatenate($PrevSubNode.C$, $SubNode(i, j, k).c$)

$score = f(X, S, Z, C)$

if $score > best\_score$ **then**

$best\_score = score$

$SubNode(i, j, k).Z = Z$

$SubNode(i, j, k).C = C$

$SubNode(i, j, k).score = score$

add $SubNode(i, j, k)$ to $NodeCollect(j)$

Termination:

$BestSubNode$ = max $SubNode.score$ **for** all $SubNode$ **in** $NodeCollect(m)$

Output $BestSubNode.C$

2.5 Hybrid recognizer

There are two main approaches for recognizing an isolated character or word pattern. Online methods treat each pattern as a temporal sequence of pen movements, while off-line methods process each pattern as a two-dimensional image. Online methods are robust against stroke connection and deformation but sensitive to stroke order variations or stroke duplications, while off-line methods are insensitive to the latter but weak with respect to the former. A combination of the online and off-line recognition methods improves the recognition accuracy because they mutually compensate each other's disadvantages [10, 29].

These two approaches are also combined at the level of features. Online recognition methods incorporating off-line features, and off-line methods including online features solve the problem of using online or off-line features alone, as shown in previous studies [4, 21].

Although a combination of recognition methods or features improves the recognition rate, it requires more computation and incurs a longer waiting time when used for batch recognition, especially for Japanese and Chinese, which have a large set of character categories.

In this study, we use a combination of online and offline recognition methods rather than features because this approach allows more freedom for selecting recognition methods. We employ a Japanese character recognizer that combines online and offline recognition methods [31] for online recognition of Japanese text, and an English word recognizer that also combines online and offline recognition methods [30] for online recognition of English text. We omit a description of the recognizers because our augmented incremental recognition method does not depend on the specifics of a particular recognizer, but is applicable to different recognizers.

3 Augmented incremental recognition method

The main idea behind our augmented incremental recognition method is to perform as much computation as possible while the user is writing. It should also keep the recognition rate as high as possible compared with the batch recognition method (in which most of the computing time is spent for the recognition of candidate character/word patterns). If candidate patterns can be processed while the user is writing, the text recognition result will be displayed without any noticeable waiting time. With the pure incremental recognition, the recognition of last character/word is made in the local or a very limited global context. If more global context can be utilized, the recognition rate will be improved. Moreover, by avoiding repeated processing after every stroke, the total CPU time can be reduced. Augmented incremental recognition incorporates all these ideas by introducing segmentation scope and recognition scope as well as three recognition techniques.

Although line segmentation, character/word segmentation, character/word recognition are different for English and Japanese, we present a unified framework that applies augmented incremental recognition and incorporates the three techniques mentioned above. This section describes our framework and the three techniques in detail.

3.1 Resuming strategy for segmentation and recognition scopes

The augmented incremental recognition method performs the recognition process after receiving newly written strokes. As new strokes change the global context of their preceding strokes, the method should provide a way to maintain and update this global context. However, it may not be necessary to keep the entire text in the global context, but only a certain window of text may suffice for effective recognition. It is desirable that this window of strokes be adjustable.

Global context can be decomposed into forward context and backward context. In this work, we consider the forward and backward contexts in terms of temporal order relation. The forward context reflects the past to evaluate the present, while the backward context reflects the future to evaluate the present. More specifically, the forward context is the context provided by the preceding strokes and the backward context is the context supplied by the succeeding strokes.

In conventional incremental recognition, because future strokes are unavailable, the backward context for the newly written strokes is missing. Therefore, the segmentation and recognition results of newly written strokes are not reliable. In augmented incremental recognition, however, because a number of strokes are accumulated before applying segmentation and recognition, later strokes can provide the context for the previously entered strokes. Thus, not only the forward context but also the backward context can be exploited to increase the recognition rate.

To determine the resuming range for each incremental recognition, Tanaka et al. [24] use a threshold calculated from average character height. This causes the problem of estimating average character height from a few strokes when the user starts writing. Wang et al. [25] use the whole text line for the range of segmentation and then determine the range of recognition based on the changed segmentation. Finally, the best-path search is made from the beginning when the user requests the recognition result. For each incremental recognition, however, it is unnecessary to apply the segmentation on the whole text line.

For augmented incremental recognition, we consider a range of strokes for resuming segmentation as "segmentation scope (S-scope)" and another range inside this for resuming recognition as "recognition scope (R-scope)."

S-scope should be determined so that the newly written strokes do not affect the segmentation before it. As the backward context by the newly written strokes affects

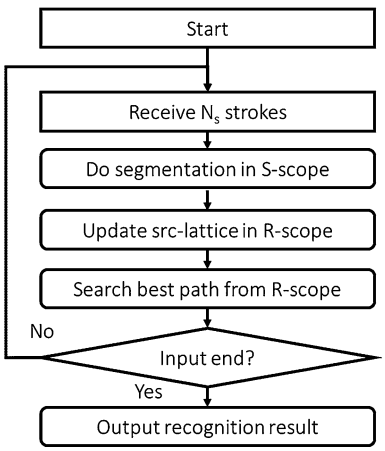


Fig. 4 Flow of augmented recognition method

a range of recent strokes, S-scope must cover this range. Moreover, it should provide a consistent forward context for segmentation.

As the segmentation before S-scope is considered stable, R-scope should be set within S-scope, but it should be designed so that the newly written strokes may only affect the recognition within the R-scope.

By appropriately setting these scopes, augmented incremental recognition incorporates the forward and backward contexts to recognize online handwritten text. Moreover, limiting the segmentation and recognition within these scopes incurs little processing cost for each incremental recognition. The best-path search can also be done incrementally inside the R-scope to reduce waiting time.

3.2 Triggering incremental recognition

Augmented incremental recognition triggers the recognition process whenever the number of newly written strokes reaches the window size N_s. In the specific case of $N_s = 1$, the method triggers the recognition in the same way as the pure incremental recognition method.

Since the context to recognize recent strokes changes in each incremental recognition, segmentation and recognition of previous incremental recognition also need to be reevaluated and updated. Triggering recognition with a large window (large number of strokes) reduces the change of the context. Therefore, it reduces the processing required to update the segmentation and recognition in each incremental recognition. This leads to reduction in total CPU time. Increasing the window size, however, incurs more waiting time for processing. We determine the window size through experiments and discuss its effectiveness.

3.3 Processing flow

Figure 4 shows the processing flow of our augmented incremental recognition method.

Augmented incremental recognition proceeds as follows. When some newly written strokes are added to the previous strokes, character/word segmentation is resumed for the current S-scope. Then, character/word recognition is resumed and the src-lattice is updated for the current R-scope. Finally, the best-path search is resumed in the R-scope, while writing continues. The process is repeated for processing new strokes in the next incremental recognition. The segmentation and recognition results obtained from the best-path search are used for the next processing cycle.

As writing proceeds, i.e., new strokes are added, the S-scope and the R-scope are updated. We call the scope before the last update the previous scope and the scope after the update the current scope, regardless of whether it is S-scope or R-scope.

3.4 Determination of S-scope

Following the above resuming strategy, we consider a certain range of global context to resume segmentation and eventually recognition. Here, we introduce a pointer called the segmentation-resumption pointer (*Seg_rp*) as a starting point for the S-scope. Thus, S-scope is from *Seg_rp* to the latest stroke. We determine *Seg_rp* based on the segmentation and recognition result of the previous cycle of incremental recognition, which is obtained from the best-path search in the src-lattice and is highly reliable. Off-strokes between two recognized characters/words in the previous R-scope are candidates for *Seg_rp* (*Seg_rp* candidates). We simply employ the number N_{seg} of characters/words from the end of the previous scope to the off-strokes in the text recognition result. In other words, we select a candidate as *Seg_rp* such that the distance from the end of the previous scope to the candidate equals to N_{seg} as illustrated in Fig. 5. The larger N_{seg} is, the wider the S-scope is. The ideas behind this are as follows:

(1) *Seg_rp* can be determined so that segmentation before *Seg_rp* is stable but that after *Seg_rp* is unstable and need to be reconsidered with the succeeding strokes.
(2) *Seg_rp* candidates are more stable as they are far away back from the end of the previous scope.

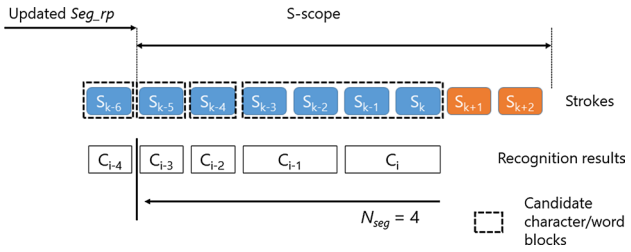


Fig. 5 Determination of S-scope

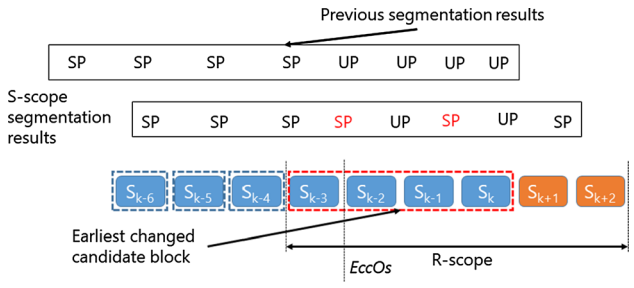
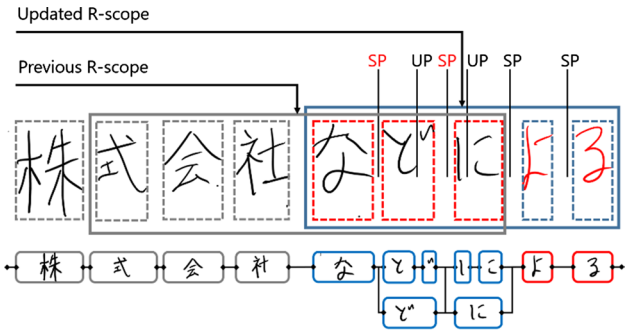
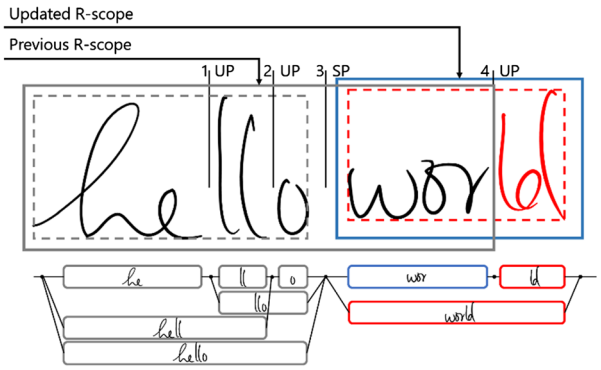


Fig. 6 Determination of recognition scope



(a) Japanese text



(b) English text

Fig. 7 Reuse of candidate word patterns for **a** Japanese text and **b** English text

3.5 Determination of R-scope

To determine the R-scope, we use the result from the segmentation process. The segmentations of the strokes before and after receiving new strokes are compared. If classifications of some off-strokes are changed, we consider that the candidate character/word blocks before the earliest classification-changed off-stroke (denoted *EccOs*) are stably classified, while the candidate character/word blocks after that are not stably classified. Otherwise, the off-stroke before the newly added strokes is considered as *EccOs*. *EccOs* may occur within some candidate character/word blocks or between two candidate character/word blocks. We define the R-scope as the sequence of strokes starting from the first stroke of the candidate character/word block containing or just preceding *EccOs* to the latest stroke. Figure 6 illustrates this method.

3.6 Update of src-lattice and resuming best path search

After determining the R-scope, we update the src-lattice inside the R-scope. Newly added strokes may change the segmentation and recognition of previous strokes in the R-scope but may leave some parts unchanged. Therefore, we can reuse them to reduce the processing time. To maximize the reuse of the src-lattice in the previous R-scope, we use the following method for updating the src-lattice in the current R-scope. It takes advantage of previously built lattice candidates in the previous R-scope. From the beginning of the current R-scope, our augmented incremental recognition method finds *SP* off-strokes and splits candidate character/word blocks by these off-strokes. Each SP off-stroke divides a candidate character/word block into two parts: preceding and succeeding this SP off-stroke. The src-lattice in these lattice blocks will be checked if a candidate character/word pattern already exists in the previous R-scope. When a candidate exists, we obtain it from the previous R-scope; otherwise, we rebuild it.

Figure 7a, b show an example, for Japanese and English text, respectively, for updating the src-lattice when N_s is set to two. When new N_s strokes are added, shown in red, we update the src-lattice from the beginning of the current R-scope, which triggers the building of nine candidate character patterns (Fig. 7a) and seven candidate word patterns (Fig. 7b). Among them, only two candidate character patterns and three candidate word patterns, bounded by red solid rectangles, have to be newly built. The remaining candidate character patterns and candidate word patterns bounded by blue solid rectangles are reused from the previous R-scope.

When the src-lattice is updated, we resume the search by the Viterbi algorithm from the first character/word lattice block in the current R-scope instead of searching from the beginning as in [24, 25]. This method limits the processing time for the best-path search regardless the length of input sequence.

Let i_r be the index of the character/word lattice block to be resumed. The evaluation function in Eq. (7) is decomposed into two parts consisting of the evaluation up to $i_r - 1$ and the evaluation from i_r to the last character/word lattice block as in Eq. (8).

$$f(X,S,Z,C) = f(X,S,Z,C)\left[1,i_r-1\right] + f(X,S,Z,C)\left[i_r,m\right] \tag{8}$$

Since the recognition candidates do not change before the R-scope, the incremental best path search by Viterbi remains unchanged up to $i_r - 1$. Therefore, by calculating the second term of the right side in Eq. (8), we can maintain the context to be the same as the batch recognition method.

3.7 Fixation of SPs from UPs

When all the candidate segmentation points are classified as UP, each UP doubles the number of possible paths passing through it. The method by Wang et al. [25] does not consider SP off-strokes, where all the candidate segmentation points are classified as UP. For recognizing an input sequence with

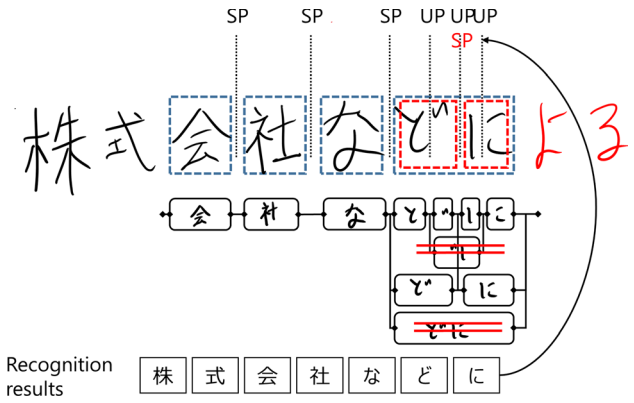


Fig. 8 Path reduction by UP fixation

n_s UPs, 2^ n_s recognition paths must be evaluated. Therefore, recognition time grows exponentially as the length of input sequence increases. To reduce recognition time for handwritten Chinese and Japanese text, candidate character patterns formed by multiple primitive segments have been restricted in length [27, 31]. The length restriction, however, is not applicable for handwritten English text due to a large variance in the lengths of candidate word patterns.

Our method sets SP for the strokes that are highly likely to be separated. As more SPs are determined, the recognition time becomes shorter. On the other hand, the recognition rate may degrade due to misclassifications of off-strokes to SPs. We refer to methods without SPs and NSPs as full soft decision, those with SPs or NSPs as partial soft decision, and those with both SPs and NSPs as minimum soft decision. We employ the minimum soft decision in our approach

Determination of SP off-strokes greatly affects the recognition rate and the performance of our augmented incremental recognition method. Although SP off-strokes can be detected based on the result of the segmentation process, the performance of segmentation using an SVM for detecting SP off-strokes is still limited. Due to the uncertainty of segmentation, a large number of outputs from the SVM are marked as UPs. To overcome this problem, we also use the result of text recognition up to the latest R-scope to fix more UPs to SP off-strokes in the S-scope. We call this process UP fixation. The UP off-strokes between recognized characters/words, before the latest N_{seg_det} characters/words in the recognition result, are fixed as SP off-strokes. Here, N_{seg_det} denotes a predefined constant for the minimum number of characters/words that follow an UP off-stroke to make it a stable SP off-stroke. Generally, N_{seg_det} is set smaller than or equal to N_{seg}. Figure 8 shows an example of UP fixation removing two candidate character patterns (red double strike-through lined box). Although the candidate character patterns are already built at the current UP fixation, so that the cost for recognizing these candidate character patterns has already occurred, and we expect a reduction in cost for the future candidate character patterns that incorporate these current candidates character patterns as future strokes are inputted.

3.8 Skipping partial patterns

For incremental recognition, incomplete character/word patterns occur at the end of the text while writing. Unless predictive input is used, the recognition of these incomplete character/word patterns has no meaning. If we can skip recognizing them, it would save processing time. Recognition of partial character patterns for Japanese or partial word patterns for English can be postponed until the complete character/word patterns are received. Therefore, we skip recognizing them to reduce CPU time. We treat candidate character/word patterns containing the last primitive segment as partial candidate character/word patterns (PPs) until a new primitive segment is detected or the recognition is requested. We call this process PP skip.

3.9 Handling delayed strokes

To correctly segment a text line that includes delayed strokes, we first detect the delayed strokes and ignore them in the segmentation process. We then determine a segmented block for each delayed stroke into which that stroke is merged. Finally, we rebuild the src-lattice.

Delayed strokes are detected using the previous recognition result. First, we retrieve the bounding box for each recognized character/word from the segmentation-recognition result up to the previous R-scope. We then deem 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/word patterns containing the delayed strokes. We extend the R-scope back to the point where the delayed stroke occurs. Then, we resume the best-path search from the R-scope. By extending the R-scope, isolated character/word recognition results (candidates) inside the R-scope may change. When the best-path search is resumed from the beginning of the R-scope, different paths may be chosen due to different candidates in the R-scope. This may change the previously selected recognition results (although candidates outside the R-scope are not changed).

It is possible to provide real-time feedback even if a delayed stroke occurs, as it does not take long to rebuild the candidate lattice containing the delayed stroke and search for the best path from the R-scope. Algorithm 2 shows the pseudocode of augmented incremental recognition with handling delayed strokes.

Algorithm 2: Augmented incremental recognition with handling delayed strokes.

Input: N_s new strokes *NewStrokes*
LastBestLatticePath

1. Algorithm
2. *Strokes* = *Strokes*.add(*NewStrokes*)
3. *Segmentations*, *EccOs*= incrementalSegmentation(*Strokes, LastBestLatticePath*)
4. *DelayedStrokes* = checkDelayedStrokes(*Strokes, LastBestLatticePath*)
5. *Strokes* = *Strokes*.remove(*DelayedStrokes*)
6. *Lattice, RScope* = buildLatice(*Strokes, Segmentations, EccOs*)
7. *Lattice*, *RScope* = addStrokeToLattice(*Lattice*, *DelayedStrokes*)
8. *LatticeCandidates* = recognize(*Lattice*, *RScope*)
9. *BestPath* = bestPathSearch(*LatticeCandidates*, *RScope*)

Output: *BestPath*

4 Evaluation experiments and discussion

4.1 Metrics of segmentation evaluation

First, over-segmentation is applied and 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 be the numbers of returned SPs, NSPs and UPs, respectively. $\#SP_c$ is the number of correctly classified SPs among the returned SP_s. $\#SP_t$ 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):

$$p = \frac{\#SP_c}{\#SP} \tag{9}$$

Recall (r):

$$r = \frac{\#SP_c + \#UP_t}{\#SP_t} \tag{10}$$

Inclusion of $\#UP_t$ in the dividend is typical for over-segmentation since UPs maintain the possibility that they will be classified correctly.

F-measure (f) is calculated as follows:

$$f = \frac{2 \times p \times r}{(p + r)} \tag{11}$$

Although UPs maintain the possibility that they will be classified correctly, thus improving 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 its ability to determine more SPs instead of UPs by the following formula:

$$d = \frac{\#SP}{(\#SP + \#UP)} \tag{12}$$

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 ones are NSPs. Let $\#SP_f$, $\#SP_{fc}$ and $\#SP_{ft}$ be 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 denoted as the segmentation measure is evaluated as follows:

$$F = \frac{2 \times P \times R}{(P + R)} \tag{13}$$

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

$$P = \frac{\#SP_{fc}}{\#SP_f} \tag{14}$$

$$R = \frac{\#SP_{fc}}{\#SP_{ft}} \tag{15}$$

4.2 Average waiting time in recognition interfaces

For busy recognition interface, waiting occurs at each incremental recognition step. The waiting time is the processing time of the incremental recognition step. Therefore, the average waiting time t_{busy} is calculated as follows:

$$t_{\text{busy}} = \frac{1}{n}\sum_{i=1}^{n} t^{i}_{\text{incre}} \tag{16}$$

where t^{i}_{incre} is the processing time for incremental recognition step i, and n is the number of incremental recognition steps.

For lazy recognition interface, assuming that the user requests the result of the entire text recognition immediately after the last stroke, the waiting time is measured from receiving the final stroke until the time when the recognition result is returned. This waiting time is incurred by the processing time for the last incremental recognition step and the processing time for previous incremental recognition steps if it is longer than the user's writing duration. Therefore, we use the following formula to calculate the waiting time t_{lazy} for lazy recognition interface:

$$t_{\text{lazy}} = t^{n}_{\text{incre}} + t_{\text{previous}} \tag{17}$$

where t^{n}_{incre} and t_{previous} are the processing time of the last incremental recognition and the waiting time caused by the previous incremental recognition steps, respectively. The second term t_{previous} is calculated as follows:

$$t_{\text{previous}} = \begin{cases} \sum_{i=1}^{n-1} t^{i}_{\text{incre}} - t_{\text{writing}}, f \sum_{i=1}^{n-1} t^{i}_{\text{incre}} > t_{\text{writing}} \\ 0 \qquad \text{else} \end{cases} \tag{18}$$

where $\sum_{i=1}^{n-1} t^{i}_{\text{incre}}$ and t_{writing} are the total processing times of the previous incremental recognition steps and the user's writing duration, respectively.

Table 1 Statistics of online handwritten text databases

	Kondate	IAM-OnDB
Number of strokes/text line	38.52	25.09
Max number of strokes/text line	213	62
Number of strokes/character	3.77	–
Number of strokes/word	–	4.14

4.3 Experimental setup

To evaluate our augmented incremental recognition method, we conducted experiments on both online handwritten Japanese text and English text.

For Japanese text, we trained the character recognizer and geometric scoring functions using Japanese online handwriting database Nakayosi [15]. We used 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. From the TUAT-Kondate database collected from 100 people [12], 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 the weighting parameters and 1 set (3511 text lines written by 25 people) for testing as in [27, 31]. 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 unbiased effect with respect to data sets.

For English text, we used the IAM online database (IAM-OnDB) [8], which consists of pen trajectories collected from 221 different writers using an electronic whiteboard. We followed the handwritten text recognition task IAM-OnDB-t2, in which the database was divided into a training set, two validation sets, and a test set containing 5364, 1438, 1518 and 3859 written lines, respectively. We used a trigram table extracted from the LOB text corpus [5] for language modeling.

To evaluate the average waiting time of the system, especially while dealing with long input sequences, we selected 20 longest stroke sequences in each database. Since the average number of strokes per text line in both Kondate and IAM-OnDB is small, as shown in Table 1, we only evaluated the average waiting time for the selected sequences. We chose the top 20 longest sequences: from 154 to 214 strokes per text line (avg. 165.9 strokes) from Kondate, and from 50 to 62 strokes per text line (avg. 54.55 strokes) from IAM-OnDB, respectively.

The parameters of the evaluation function in Eq. (7) have been trained with each training set, but N_s and N_{seg} are not

Table 2 Overall performance of augmented incremental recognition

	Japanese text			English text		
	Aug. incr.	*B*	P. Incr.	Aug. incr.	*B*	P. incr.
Recognition rate (%)	93.19 ± 1.16	93.28 ± 1.2	92.50 ± 1.3	74.64	74.79	71.50
t_{busy} (s)	0.0223 (0.085)	–	0.0175	0.258 (4.37)	–	0.365
t_{lazy} (s)	0.0299 (0.124)	0.118 (1.523)	–	1.04 (5.71)	3.85 (20.5)	–
CPU time (ms)	4.20	3.04	7.03	29.5	18.2	57.2

Aug. incr., augmented incremental recognition mode; *B*, batch recognition mode; P. Incr., pure incremental recognition mode

N_s, N_{seg} and $N_{\text{seg\_det}}$ were set to 5, 8 and 5 for Japanese text, and to 1, 8 and 3 for English text, respectively. Waiting time shown in brackets is the maximum waiting time

trained since N_s and N_{seg} are control variables rather than parameters.

We implemented augmented incremental recognition systems for online handwritten Japanese text and English text. We ran all the systems on an Intel(R) Xeon(R) CPU E5-2630v2 2.6 Ghz with 32-GB memory.

It is not easy to compare our method with previous incremental or batch methods since the implementations and the datasets are different. Therefore, we implemented a unified approach so that the previous methods are realized by setting parameters, and then we show that the proposed method achieves a better performance in the waiting time and CPU time without degrading the recognition rate even compared with the batch recognition. In this sense, the extremal baselines are the pure incremental and the batch recognition methods. Augmented incremental recognition with $N_s = 1$ is almost pure incremental recognition but enhanced from the employment of the two scopes and the three techniques mentioned in this paper. Shrinking the S-scope and R-scope to the minimum (1) and disabling the three techniques convert this approach to the pure incremental recognition.

4.4 Overall performance

We conducted an experiment to measure the overall performance of the augmented incremental recognition method. We employed all the three techniques with N_{seg} and N_{seg_det} which produced the best recognition rates for training patterns, i.e., N_s, N_{seg} and N_{seg_det} set as 5, 8 and 5 for Japanese text, and as 1, 8 and 3 for English text, respectively. Table 2 shows the recognition rate (i.e., character/word recognition accuracy), waiting time and CPU time in comparison with the batch recognition method when online handwritten text is recognized line by line. For Japanese text, averages and standard deviations are shown from fourfold of the cross-validation. For English text, just average is shown for the test set.

The recognition rate is maintained almost as high as the batch recognition method. The augmented incremental recognition method produces a recognition rate of 93.19% as compared with 93.25% of the batch recognition method for handwritten Japanese text and 74.64% as compared with 74.79% of the batch recognition method for handwritten English text. The augmented incremental method also adds little CPU time in comparison with the batch recognition method.

On the other hand, we aimed to improve the recognition accuracy in comparison with the pure incremental method while keeping the waiting time small and decreasing CPU time. As a result, the proposed method improved the recognition accuracy from 92.5 to 93.2% for Japanese and from 71.5 to 74.46% for English in comparison with the pure

Table 3 Comparison with the state-of-the-art methods

Target	Accuracies for data set		
	Data set	Accuracies (%) by the state of the art	Accuracies (%) by Aug. Incr.
Online handwritten Japanese text	Kondate	93.98 [27] 93.24 [31] (batch recognition method)	93.19
Online handwritten English text	IAMonDB	89.6 [6][a] 81.1 [10]	74.64

Aug. incr., augmented incremental recognition

[a]Trained by a private dataset

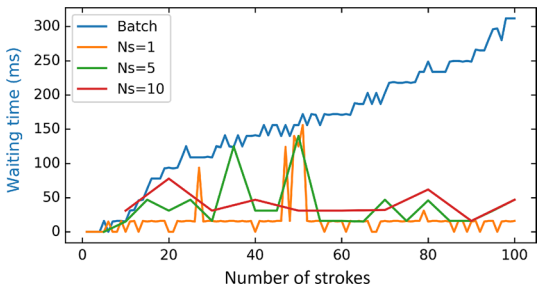


Fig. 9 Waiting time for Japanese text as the number of strokes increases

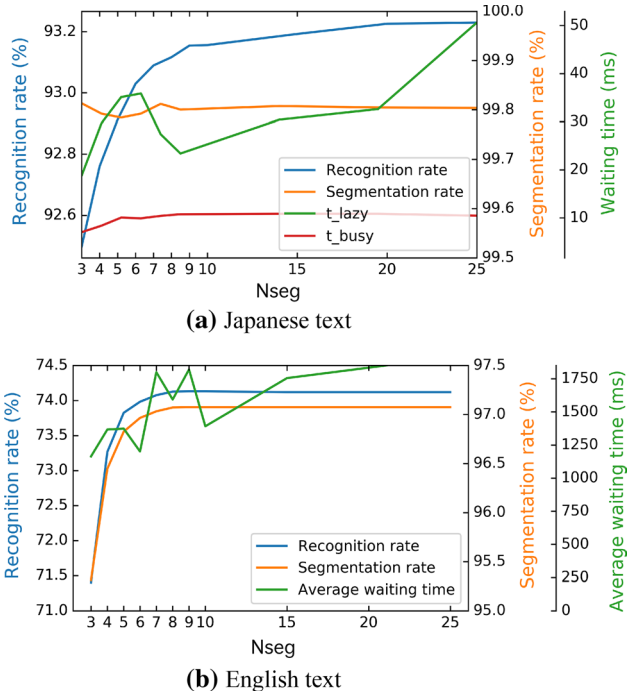


Fig. 10 Effect of changing N_{seg} for **a** Japanese text and **b** English text

incremental method. The improvements are significant, and they are validated by the paired t test with $p < 0.0005$ and $p < 0.0005$ for Japanese text and English text, respectively. We claim that the improvements are due to a more effective use of the context.

The most notable effect is in the waiting time. The augmented incremental method reduces the waiting time from 0.118 to 0.0299 s for Japanese (74.6% reduction) and from 3.85 to 1.04 s for English (72.8% reduction). The waiting time for recognizing English text is much larger than that for Japanese text because the English word recognizer requires more processing time compared with the Japanese character recognizer. For both the busy and lazy recognition interfaces by augmented incremental recognition, the waiting times t_{busy} and t_{lazy} are small enough for practical use.

This effect is further enhanced as the number of strokes increases. Although the average number and the largest number of strokes per text line are 38.5 and 213 for Kondate and 25.1 and 62 for IAM-OnDB, respectively, the waiting time by the batch recognition method becomes longer as the number of strokes per text line increases and multiple text lines are recognized. However, the waiting time by the augmented recognition method stays constant. Figure 9 shows our experimental results as the number of strokes increases for both the methods.

Last but not the least, before going into details is the comparison of recognition rates with the state-of the-art recognizers. Table 3 shows the comparison using the same test set and evaluation method. Our augmented incremental recognition is comparable with the system by Zhou et al. [27] for Japanese. For English, our method is somewhat inferior to the Google's system by Keysers et al. [6], but they employ a different training set. The performance of our system is slightly poorer compared to the best academic system by Liwicki et al. [10], which uses the same training and testing sets as well as the same corpus for the language model. The BLSTM recognizers [6, 10], which recognize character sequences without segmentation perform better than the combined recognizer for recognizing words as our segmentation-based method. However, we employ a segmentation-based method so that incremental recognition can be employed, while Liwicki et al. apply BLSTM which makes it hard to incorporate incremental recognition.

4.5 Effect of resuming segmentation and recognition

To evaluate the effect of resuming segmentation and recognition, we conducted experiments with varying N_{seg} and measured the segmentation rate, the recognition rate and the waiting time. N_{seg} was varied from 3 to 25 for Japanese and English. N_s is set to 1 in both the experiments to show the effect of changing N_{seg} clearly. Figure 10 shows the three measures for Japanese text (a) and English text (b). As N_{seg} is expanded up to 8, the segmentation rate and especially the recognition rate are generally improved, which confirms

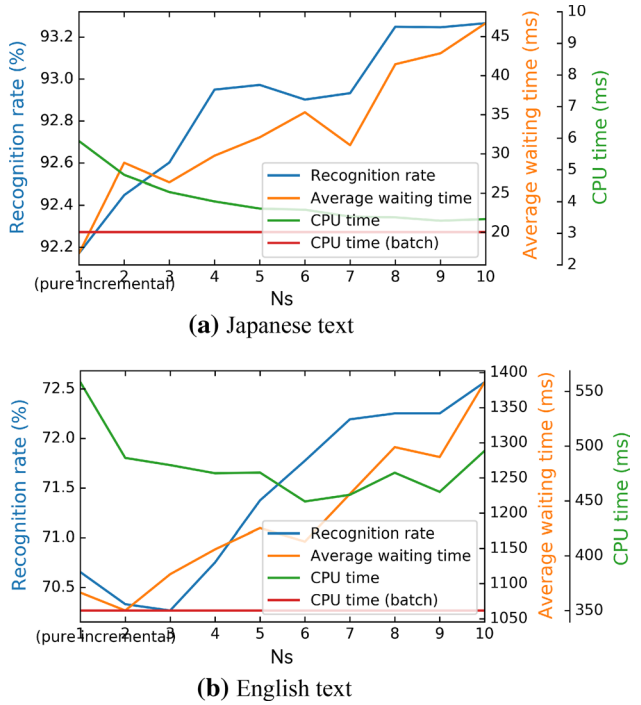


Fig. 11 Recognition rate and waiting time with N_s for **a** Japanese text and **b** English text

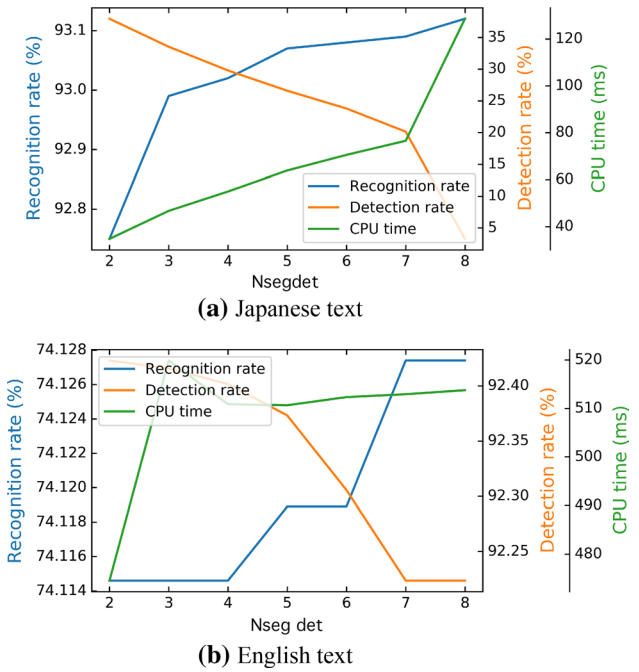


Fig. 12 Effect of UP fixation for **a** Japanese text and **b** English text

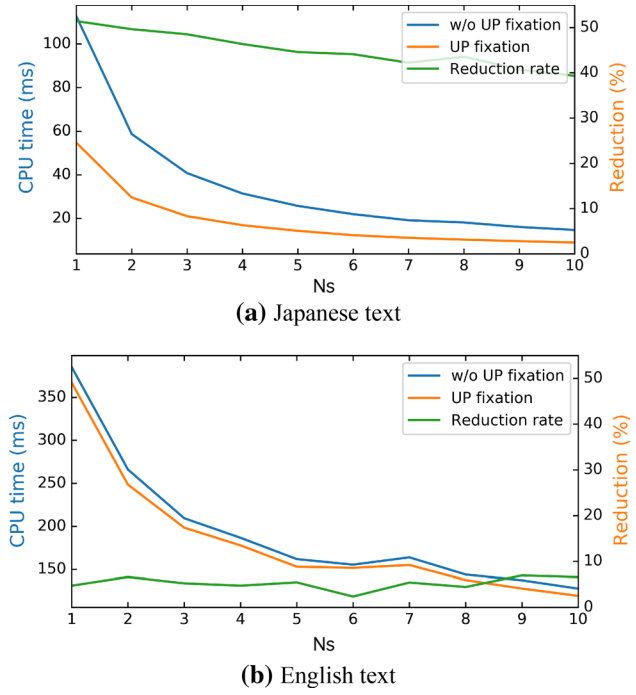


Fig. 13 CPU time with applying UP fixation for **a** Japanese text and **b** English text

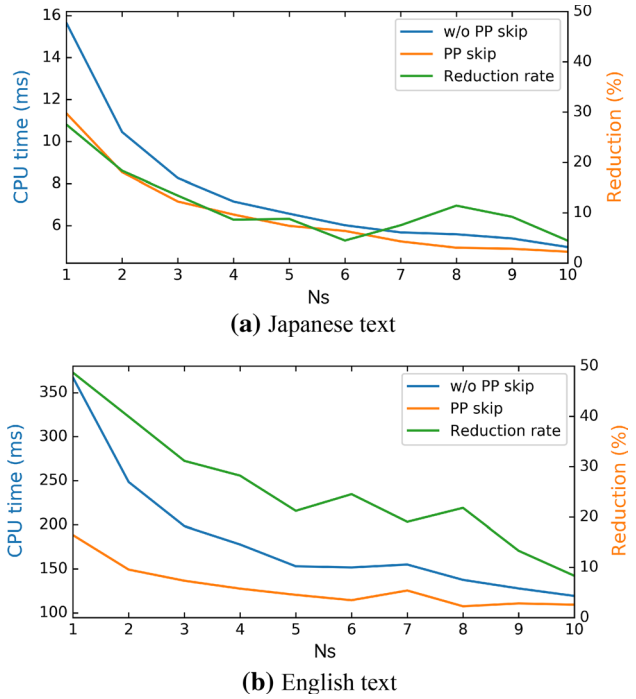


Fig. 14 CPU time with applying PP skip for **a** Japanese text and **b** English text

the effect of resuming segmentation and recognition using the scopes. For N_{seg} larger than 8, the improvement tends to be milder or even saturated. On the other hand, the average waiting time increases gradually.

As the scope is expanded, more unstable segmentation points are covered and determined correctly, thereby yielding better segmentation and recognition rates, although the average waiting time is extended due to an increase in changes in segmentation and recognition.

Note that augmented incremental recognition with $N_s = 1$ is enhanced from the pure incremental recognition by the employment of the two scopes and the three techniques. In fact, the pure incremental recognition produced a recognition rate of 92.50% and 71.50% while augmented incremental recognition with $N_s = 1$ produced better results with the best 93.10% and 74.64% for Japanese text and English text, respectively.

4.6 Effect of the recognition trigger

We conducted experiments to evaluate the effect of window size N_s to trigger recognition for both Japanese and English. To make the effect clear, we fixed N_{seg} at 3 while increasing N_s from 1 to 10. Figure 11 shows the result. As N_s is expanded, the CPU time is gradually reduced and approaches the batch recognition method due to a reduction in the number of partial patterns. As N_s is expanded, the recognition rate is also improved due to a longer local context available for recognition. Triggering recognition with a higher number N_s, however, takes longer waiting time due to a larger amount of processing needed for each incremental recognition.

4.7 Effect of UP fixation

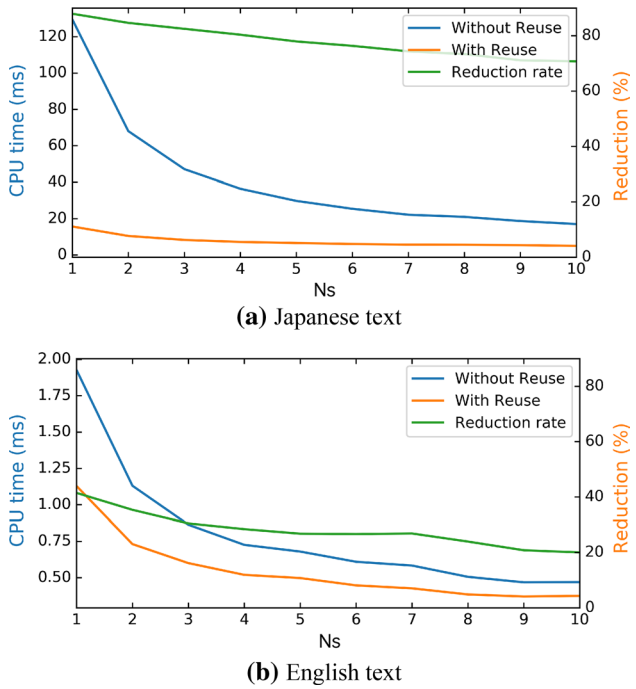
We evaluated the effect of applying UP fixation on the performance of augmented incremental recognition method. In this experiment, we fixed $N_{seg} = 8$ and executed the method by varying N_{seg_det} from 2 to 8 (since N_{seg_det} is smaller than or equal to N_{seg}). When N_{seg_det} is 8, UP fixation is not applied as explained in Sect. 3.7. Figure 12 shows the result. Applying UP fixation improves the detection rate to 37.95% when $N_{seg\_det} = 2$ as compared with the method without UP fixation of 3.28%. Larger N_{seg_det} yields a more stable fixation and brings about a higher recognition rate but lowers the detection rate. The method with UP fixation slightly reduces the recognition rate: from 93.11 to 93.08%.

On the other hand, UP fixation reduces the number of search paths and candidate lattice patterns as expected in Sect. 3.7 with the total effect of reducing the CPU time as shown in Fig. 13.

For Japanese, the effect is largest when $N_s = 1$ with 51.42% reduction in the CPU time and it decreases slightly as N_s is set larger. For smaller N_s, incremental recognitions are triggered more often, and thus each fixed UP remains

Table 4 Orders of using systems and modes

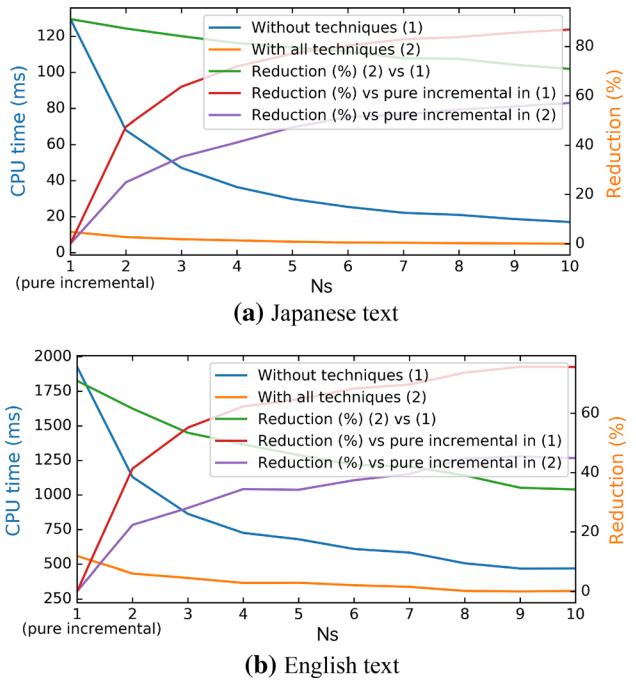
Group	System and mode					
	English			Japanese		
	B	Aug. incr.	Q	*B*	Aug. incr.	Q
G1.1	1	2	3	5	4	6
G1.2	2	1	3	4	5	6
G2.1	5	4	6	1	2	3
G2.2	4	5	6	2	1	3



(a) Japanese text

(b) English text

Fig. 15 CPU time with applying reuse for **a** Japanese text and **b** English text



(a) Japanese text

(b) English text

Fig. 16 CPU time with applying all the techniques for **a** Japanese text and **b** English text

effective for succeeding incremental recognitions. For English, however, the effect of UP fixation is rather small and there is no clear difference with changing N_s. This is due to the high detection rate of English text segmentation using BLSTM [17].

4.8 Effect of PP skip

We conducted experiments to evaluate the effect of PP skip on reducing CPU time. In the experiments, N_s was varied from 1 to 10. Figure 14 shows the CPU time with and without applying PP skip. PP skip reduces the CPU time up to 27.51% for Japanese text and up to 48.69% for English text. The highest reduction rate is when $N_s = 1$, since the number of partial patterns is largest in this case. The effect is larger for English than for Japanese, since English word recognition takes longer than Japanese character recognition, and the number of strokes per English word is larger than that per Japanese character as shown in Table 4.

4.9 Effect of reuse

We evaluated the CPU time performance of the system with and without applying reuse. Figure 15 shows the CPU time performance for both Japanese text and English text with varying N_s from 1 to 10. Applying reuse reduces the CPU time up to 89.72% for Japanese text and 41.44% for English text by the recognition system. The effect of reuse is largest for $N_s = 1$, and it decreases for larger N_s. For smaller N_s, due to the larger number of triggered incremental recognitions, a candidate character/word pattern could be reused more often, thereby increasing its effectiveness.

4.10 Effect of all three techniques

We evaluated the effect of applying all the three techniques to the recognition methods. In this experiment, we set the best parameters as in Sect. 4.4 and ran the experiments with N_s from 1 to 10. Figure 16 shows the results. Applying all of the techniques for the augmented incremental recognition method reduces the CPU time up to 91.11% for Japanese text and 71.00% for English. The effect of the three techniques is largest when $N_s = 1$, which is pure incremental recognition. This shows that the three techniques are also effective for pure incremental recognition, though their effectiveness gradually decreases for larger N_s. Without the three techniques, semi-incremental recognition reduces up to 86.88% of the CPU time for Japanese and up to 75.66% for English when $N_s = 10$ from the pure incremental recognition of $N_s = 1$. The red line in the figure shows the reduction rate of semi-incremental recognition from pure incremental recognition. With all the techniques, semi-incremental recognition, called total augmented incremental recognition, reduces up to 57.02% of the CPU time for Japanese and up to 44.87% for English when $N_s = 10$ from the pure incremental recognition of $N_s = 1$ as shown in the purple line. Whether the three techniques are combined or not, the augmented incremental recognition incurs less CPU time than the pure incremental recognition method. This shows the speed up is not only by the three techniques but also by the semi-incremental processing.

Table 5 Questionnaire on the waiting time and intermediate feedback using five-level Likert scale

#	Question
Q1	How do you feel about the waiting time of the augmented incremental recognition mode? (1: late, 2: rather late, 3: even, 4: rather quick, 5: quick)
Q2	How do you feel about the waiting time of the batch recognition mode? (1: late, 2: rather late, 3: even, 4: rather quick, 5: quick)
Q3	Which of the two modes do you think gives a smaller waiting time; the augmented incremental mode or the batch mode? (1: clearly batch, 2: batch, 3: even, 4: augmented, 5: clearly augmented)
Q4	Is the intermediate feedback helpful while writing? (1: unhelpful, 2: rather unhelpful, 3: even, 4: rather helpful, 5: helpful)
Q5	Give your opinion on the intermediate feedback

B, batch recognition mode; Aug. incr., augmented incremental recognition mode; Q, answering questionnaire

Table 6 Average user evaluation scores for waiting time

Group		System and mode									
		Japanese					English				
		Scores of Q1 and Q2		Score of Q3	Real waiting time (s)		Scores of Q1 and Q2		Score of Q3	Real waiting time (s)	
		B	Aug. incr.		*B*	Aug. incr.	*B*	Aug. incr.		*B*	Aug. incr.
G1	G 1.1	3.80	4.60	3.80	0.27 ± 0.08	0.03 ± 0.02	2.60	4.40	4.80	2.26 ± 0.43	0.60 ± 0.13
	G 1.2	3.80	5.00	4.80	0.33 ± 0.03	0.04 ± 0.02	2.40	3.40	4.20	1.84 ± 0.45	0.98 ± 0.88
G2	G 2.1	3.40	4.00	4.00	0.26 ± 0.04	0.04 ± 0.02	1.60	2.80	4.80	2.31 ± 0.48	0.78 ± 0.36
	G 2.2	4.20	4.20	3.60	0.28 ± 0.04	0.03 ± 0.01	2.40	4.00	4.80	1.96 ± 0.45	0.56 ± 0.11
	Ave.	3.80	4.45	4.05	0.28 ± 0.05	0.04 ± 0.02	2.25	3.65	4.65	2.09 ± 0.45	0.73 ± 0.37

B, batch recognition mode; Aug. incr., augmented incremental recognition mode

Table 7 User evaluation of intermediate feedback

Score	# People	Typical comments	# Comments/# total
4 or 5 (agree)	18	"I can perceive misrecognitions to fix them" "It would be better if there was"	12/18
2 (disagree)	1	No comment	0/1
3 (neutral)	1	"Since I concentrate on writing, I rarely confirm intermediate feedback"	1/1

5 User experience evaluation

5.1 Setup for experiment

We prepared one online recognition system for Japanese and one for English. In each system, we provided modes for augmented incremental recognition and batch recognition. We asked 20 participants to use both the modes in both the systems. The participants were divided into four groups G1.1, G1.2, G2.1 and G2.2, and they were asked to use the two modes in each system according to the order as shown in Table 4. The grouping and ordering in the experiment were designed to cancel the effect of different people and the order to use the two modes. Each participant was asked to write 10 English sentences (4–7 words each) for the English recognizer with each of the two modes, and 10 Japanese sentences (16–18 characters each) for the Japanese recognizer again with each mode. After they wrote 10 sentences in both the modes for each system, we asked them to answer a questionnaire about the waiting time in the two modes. We also asked them to evaluate whether the intermediate feedback was helpful. All questions used a 5-level Likert scale. The questionnaire is shown in Table 5.

5.2 Result of experiment

Table 6 shows the average scores for each question from Q1 to Q3. For Japanese text input, both the batch and augmented incremental recognition modes received a positive feedback (score > 3) as they incurred little waiting time, because the Japanese recognizer is faster than the English recognizer. The augmented incremental recognition mode received a higher average evaluation score than the batch recognition mode because it incurs less waiting time. The answers to Q3 show the participants' preference for the augmented incremental recognition. For English text input, the batch recognition mode received a negative feedback (score < 3) as they incur a larger waiting time (about 2 s.) on average. On the other hand, the augmented incremental recognition mode received a positive feedback. The answers to Q3 show the participants' clear preference for the augmented incremental recognition.

We validated the hypothesis that the users accept batch recognition and augmented incremental recognition equally by a paired t test, and it was rejected for Japanese text input and also for English text input with $p < 0.001$ and $p < 0.0005$, respectively.

In the experiment, we also provided the participants with two conditions: (1) automatic intermediate feedback by augmented incremental recognition, and (2) no feedback until requested. For the evaluation, we asked their opinions in Q4 and free comments in Q5 on the intermediate feedback. Table 7 shows the result. Most participants prefer intermediate feedback during writing with average score of 4.4 ± 0.3. Among them, the most common opinion was that the user can perceive misrecognitions to fix them. The intermediate result, however, may not be correct until the last few strokes, and so a few participants did not find intermediate recognition so useful.

6 Conclusion

We presented a unified approach to augmented incremental recognition for both online handwritten Japanese and English text. Augmented incremental recognition is parametrized to cover pure incremental recognition, which triggers recognition at every input stroke, and semi-incremental recognition triggering recognition after several input strokes. Resuming the segmentation and recognition in local scopes reduces the waiting time to be very small for users. Augmented incremental recognition incorporates three techniques: reusing the segmentation and recognition candidate lattice in the previous R-scope, fixing undecided segmentation points and skipping recognition of partial candidate character/word patterns.

Effectiveness of the overall method and all the three techniques were evaluated on the common large databases of online handwritten Japanese and English text patterns with notable effects. The proposed method reduces 74.6% of the waiting time for Japanese and 72.8% of the waiting time for English as compared with the batch recognition method without scarifying the recognition rate.

The three techniques show their effectiveness. Reusing the segmentation and recognition candidate lattice reduces the CPU time up to 87.92%. Fixing undecided segmentation points shortens the block size and reduces the CPU time up to 51.42%. Skipping recognition of partial candidate character/word patterns reduces it up to 48.69% independently. Overall, the three techniques reduce the CPU time up to 91.11% by the recognition system without degrading the recognition rate.

The augmented incremental recognition method is clearly superior to the batch recognition method in the waiting time without degrading the recognition rate. It also excels pure incremental recognition in the character recognition rate and the total CPU time. Our user experience study also confirms the superiority of augmented incremental recognition. We demonstrated augmented incremental recognition for Japanese and English; it can be applied for other languages as well. Still, there remain some research issues. Although our user study showed that intermediate feedback owing to augmented incremental recognition is appreciated, this may change after using the system several times or after a while. To understand this effect, a long-term user study with the

system needs to be conducted. Another research issue for the future is to realize augmented incremental recognition for segmentation-free recognition methods and apply techniques based on deep neural networks.

Acknowledgements This research has been partially supported by NEDO under the contract number 27J1103, JSPS KAKENHI Grant Number JP 18K18068.

References

1. Gao, J., Zhu, B., Nakagawa, M.: Building compact recognizer with recognition rate maintained for on-line handwritten Japanese text recognition. Pattern Recognit. Lett. **35**, 169–177 (2014). https://doi.org/10.1016/j.patrec.2013.08.014
2. Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., Schmidhuber, J.: A novel connectionist system for unconstrained handwriting recognition. IEEE Trans. Pattern Anal. Mach. Intell. **31**, 855–868 (2009). https://doi.org/10.1109/TPAMI.2008.137
3. Graves, A., Schmidhuber, J.: Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Netw. **18**, 602–610 (2005). https://doi.org/10.1016/j.neunet.2005.06.042
4. Jaeger, S., Manke, S., Reichert, J., Waibel, A.: Online handwriting recognition: the NPen++ recognizer. Int. J. Doc. Anal. Recognit. **3**, 169–180 (2001). https://doi.org/10.1007/PL00013559
5. Johansson, S., Leech, G.N., Goodluck, H.: Manual of Information to accompany the Lancaster-Oslo/Bergen Corpus of British English, for use with digital Computers (1978)
6. Keysers, D., Deselaers, T., Rowley, H.A., Wang, L.-L., Carbune, V.: Multi-language online handwriting recognition. IEEE Trans. Pattern Anal. Mach. Intell. **39**, 1180–1194 (2017). https://doi.org/10.1109/TPAMI.2016.2572693
7. Liu, C.-L., Jaeger, S., Nakagawa, M.: Online recognition of Chinese characters: the state-of-the-art. IEEE Trans. Pattern Anal. Mach. Intell. **26**, 198–213 (2004). https://doi.org/10.1109/TPAMI.2004.1262182
8. Liwicki, M., Bunke, H.: IAM-OnDB: an on-line English sentence database acquired from handwritten text on a whiteboard. In: 8th International Conference on Document Analysis and Recognition, vol. 2, pp. 956–961. IEEE (2005)
9. Liwicki, M., Bunke, H.: HMM-based on-line recognition of handwritten whiteboard notes. In: 10th International Workshop on Frontiers in Handwriting Recognition. La Baule (France) (2006)
10. Liwicki, M., Bunke, H., Pittman, J.A., Knerr, S.: Combining diverse systems for handwritten text line recognition. Mach. Vis. Appl. **22**, 39–51 (2011). https://doi.org/10.1007/s00138-009-0208-9
11. Matic, N.P., Platt, J.C., Wang, T.: QuickStroke: an incremental on-line Chinese handwriting recognition system, vol. 3, pp. 435–439 (2002)
12. Matsushita, T., Nakagawa, M.: A database of on-line handwritten mixed objects named Kondate. In: 14th International Conference on Frontiers in Handwriting Recognition, pp. 369–374 (2014)
13. McDermott, E., Katagiri, S.: Minimum classification error for large scale speech recognition tasks using weighted finite state transducers. In: IEEE International Conference on Acoustics, Speech, and Signal Processing, pp. 113–116. IEEE (2005)
14. Nakagawa, M., Machii, K., Kato, N., Souya, T.: Lazy recognition as a principle of pen interfaces. In: INTERACT'93 and CHI'93 Conference Companion on Human Factors in Computing Systems, pp. 89–90. ACM Press, New York (1993)
15. Nakagawa, M., Matsumoto, K.: Collection of on-line handwritten Japanese character pattern databases and their analyses. Doc. Anal. Recognit. (2004). https://doi.org/10.1007/s10032-004-0125-4
16. Nakagawa, M., Zhu, B., Onuma, M.: A model of on-line handwritten Japanese text recognition free from line direction and writing format constraints. IEICE Trans. Inf. Syst. **88**, 1815–1822 (2005)
17. Nguyen, C.T., Nakagawa, M.: An improved segmentation of online English handwritten text using recurrent neural networks. In: 3rd IAPR Asian Conference on Pattern Recognition (ACPR), pp. 176–180. IEEE (2015)
18. Nguyen, C.T., Zhu, B., Nakagawa, M.: A Semi-incremental recognition method for on-line handwritten Japanese text. In: 12th International Conference on Document Analysis and Recognition, pp. 84–88. IEEE (2013)
19. Nguyen, C.T., Zhu, B., Nakagawa, M.: A Semi-incremental recognition method for on-line handwritten English text. In: 14th International Conference on Frontiers in Handwriting Recognition, pp. 234–239 (2014)
20. Nguyen, C.T., Zhu, B., Nakagawa, M.: Semi-incremental recognition of on-line handwritten Japanese text. IEICE Trans. Inf. Syst. **99**, 2619–2628 (2016). https://doi.org/10.1587/transinf.2016edp7051
21. Okamoto, M., Yamamoto, K.: On-line handwriting character recognition using direction-change features that consider imaginary strokes. Pattern Recognit. **32**, 1115–1128 (1999). https://doi.org/10.1016/S0031-3203(98)00153-8
22. Plamondon, R., Srihari, S.N.: On-line and off-line handwriting recognition: a comprehensive survey. IEEE Trans. Pattern Anal. Mach. Intell. **22**, 63–84 (2000). https://doi.org/10.1109/34.824821
23. Shivram, A., Zhu, B., Setlur, S., Nakagawa, M., Govindaraju, V.: Segmentation based online word recognition: a conditional random field driven beam search strategy. In: 12th International Conference on Document Analysis and Recognition, pp. 852–856. IEEE (2013)
24. Tanaka, H., Akiyama, K., Ishigaki, K.: Realtime box-free on-line handwriting string recognition using layer-delayed segmentation method. IEICE Technical Report, vol. 101, pp. 155–162. Institute of Electronics, Information and Communication Engineers (2002)
25. Wang, D.H., Liu, C.L., Zhou, X.D.: An approach for real-time recognition of online Chinese handwritten sentences. Pattern Recognit. **45**, 3661–3675 (2012). https://doi.org/10.1016/j.patcog.2012.04.020
26. Wang, Q., Yin, F., Liu, C.-L., Member, S.: Handwritten Chinese text recognition by integrating multiple contexts. IEEE Trans. Pattern Anal. Mach. Intell. **34**, 1469–1481 (2012). https://doi.org/10.1109/TPAMI.2011.264
27. Zhou, Xiang-Dong, Wang, Da-Han, Tian, Feng, Liu, Cheng-Lin, Nakagawa, M.: Handwritten Chinese/Japanese text recognition using semi-Markov conditional random fields. IEEE Trans. Pattern Anal. Mach. Intell. **35**, 2413–2426 (2013). https://doi.org/10.1109/TPAMI.2013.49
28. Zhou, X.-D., Wang, D.-H., Liu, C.-L.: Grouping text lines in online handwritten Japanese documents by combining temporal and spatial information. In: 8th IAPR International Workshop on Document Analysis Systems, pp. 61–68. https://doi.org/10.1109/das.2008.15 (2008)
29. Zhu, B., Gao, J., Nakagawa, M.: Objective function design for MCE-based combination of on-line and off-line character recognizers for on-line handwritten Japanese text recognition. In: 11th International Conference on Document Analysis and Recognition, pp. 594–598. IEEE

30. Zhu, B., Shivram, A., Setlur, S., Govindaraju, V., Nakagawa, M.: Online handwritten cursive word recognition using segmentation-free MRF in combination with P2DBMN-MQDF. In: 12th International Conference on Document Analysis and Recognition, pp. 349–353. IEEE (2013)
31. Zhu, B., Zhou, X.-D., Liu, C.-L., Nakagawa, M.: A robust model for on-line handwritten japanese text recognition. Int. J. Doc. Anal. Recognit. **13**, 121–131 (2010). https://doi.org/10.1007/s10032-009-0111-y