A robust approach to text line grouping in online handwritten Japanese documents

Xiang-Dong Zhou*, Da-Han Wang, Cheng-Lin Liu

National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, 95 Zhongguancun East Road, Beijing 100190, PR China

ARTICLE INFO

Article history:
Received 22 May 2008
Received in revised form 7 September 2008
Accepted 18 October 2008

Keywords:
Online handwritten documents
Text line grouping
MCE training
Temporal merge
Spatial merge

ABSTRACT

In this paper, we present an effective approach for grouping text lines in online handwritten Japanese documents by combining temporal and spatial information. With decision functions optimized by supervised learning, the approach has few artificial parameters and utilizes little prior knowledge. First, the strokes in the document are grouped into text line strings according to off-stroke distances. Each text line string, which may contain multiple lines, is segmented by optimizing a cost function trained by the minimum classification error (MCE) method. At the temporal merge stage, over-segmented text lines (caused by stroke classification errors) are merged with a support vector machine (SVM) classifier for making merge/non-merge decisions. Last, a spatial merge module corrects the segmentation errors caused by delayed strokes. Misclassified text/non-text strokes (stroke type classification precedes text line grouping) can be corrected at the temporal merge stage. To evaluate the performance of text line grouping, we provide a set of performance metrics for evaluating from multiple aspects. In experiments on a large number of free form documents in the Tokyo University of Agriculture and Technology (TUAT) Kondate database, the proposed approach achieves the entity detection metric (EDM) rate of 0.8992 and the edit-distance rate (EDR) of 0.1114. For grouping of pure text strokes, the performance reaches EDM of 0.9591 and EDR of 0.0669.

© 2008 Elsevier Ltd. All rights reserved.

1. Introduction

With the increasing use of tablet PCs, electronic whiteboards, and digital pens on paper, users can draw various heterogeneous structures such as text, drawings and table forms freely on a large writing area. Such freely handwritten ink documents bring new challenges to automatic analysis and recognition. An ink document (online handwritten document) comprises a sequence of strokes, which are to be grouped into various structural units (drawings, text lines, words, characters, etc). Text lines are the most salient structures in ink documents and their reliable extraction is the pre-condition to further processing tasks such as character recognition, text editing and retrieval. This paper addresses the problem of text line grouping in freeform online handwritten Japanese documents.

Earlier systems of ink document analysis [1,2] used to assume horizontal text lines and simply separate the text lines according to the projection profile. Liwichi et al. [3] proposed a text line detection method based on dynamic programming (DP) with a cost function involving hand-tuned parameters. This system was designed for ink pages with parallel text lines, and the online information was used merely in post-processing. It is important to point out that the assumption of text line regularity is often violated in ink documents because of the arbitrary line directions and inter-line distances [4]. Nevertheless, compared to offline documents, online handwriting has the advantage that the temporal order of strokes is available, which provides useful cues for text line grouping besides spatial information. Higher accuracy of grouping is achievable by utilizing both the temporal and spatial information.

Ye et al. [5] proposed a global optimization method integrating the likelihood of the resulting lines and the consistency of their configuration for text line grouping. The initial segmentation by DP minimizes a cost function involving the number of text lines as a constraint, which can effectively avoid over-segmentation. Then, a local gradient-decent algorithm is used to evaluate the splitting and merging hypotheses iteratively to minimize the global cost function. The global optimization method is superior to heuristic rules because it incorporates both the prior knowledge and the information from data in a principled formulation. However, without a training procedure, the parameters in the cost function have to be tuned manually.
In their enhanced system [6], a high-confidence-first (HCF) method was proposed to group text lines and writing regions, with an AdaBoost classifier for making merge/non-merge decisions. The classifier was trained from ink data and provides confidence of merge. For Japanese documents, Nakagawa et al. [7] describe a system in which the stroke sequence is segmented into text lines according to off-stroke (pen-up) distances and changing of writing directions based on the fact that off-strokes within a text line are mostly shorter than those between text lines and the text lines are usually straight. However, due to the variability of character size and spacing, this segmentation method does not perform reliably.

To better utilize the temporal and spatial information in online handwriting, we propose a text line grouping approach with very few artificial parameters and little prior knowledge. We only assume that the text lines are approximately straight (curved lines will be segmented as piecewise linear text lines by the proposed approach), while the writing directions can be arbitrary and need not to be parallel with each other. Unlike the projection based methods [1,2], we do not estimate the inter-line distance, so the text lines can be arbitrarily close to each other. The above merits also favor the extensibility of the proposed method for text line grouping in non-Japanese ink documents.

The proposed text line grouping process performs in several stages. Initially, strokes are grouped into text line strings according to off-stroke distances. A discriminant function is trained under the string-level minimum classification error (MCE) criterion [8,9] to separate the over-merged text lines with the beam search strategy. To correct the stroke classification errors and merge the misclassified strokes to text lines in documents of mixed text/drawing (in this case, a text/non-text classification procedure precedes text line grouping), an support vector machine (SVM) classifier is trained to make merge/non-merge decisions. Last, the errors caused by delayed strokes are amended by a spatial merge step.

Partitioning stroke sequence into text lines based on optimization has been tried by previous works [3–5], but without a training procedure, the parameters in the cost function have to be tuned manually. We take into account the similarity of this problem to character string recognition integrating character segmentation and classification [10,11] and use string-level MCE training [8,9], which has been applied to numeral string recognition [12], for training the parameters of cost function. The merge/non-merge decision by an SVM classifier in temporal merge stage is similar to that of Ref. [6], which uses an AdaBoost classifier. By our temporal merge module, text/non-text stroke classification errors can be corrected.

As for performance evaluation of text line grouping algorithms, there has not been a unified criterion. In Ref. [5], the recall metric, which is defined as the number of correct lines divided by the number of labeled lines in each page, is used to measure the accuracy of the system. In Refs. [4] and [6], the edit-distance metrics are employed to evaluate the system performance. The edit-distance between the text line detection result and the ground-truth is defined as the minimum number of split or merge operations needed to correct all errors, and the error rate is defined as the total number of edit operations divided by the total number of labeled text lines [6]. The system proposed by Liwichi et. al. [3] adopts the stroke classification rate defined as the number of correctly assigned strokes divided by the total number of strokes, and the document classification rate defined as the number of correctly processed documents divided by the total number of documents. Each of the above metrics can evaluate the system from a certain aspect, while to achieve an overall evaluation, we need a systematic methodology. Inspired by the performance evaluation methods for graphics recognition systems [13], which are also used in the ICDAR page segmentation competitions and offline handwriting segmentation contest [14], we give an extensive set of evaluation metrics.

To demonstrate the effectiveness of the proposed approach, we have experimented on the Tokyo University of Agriculture and Technology (TUAT) Kondate database [7] with two settings: one with perfect text/non-text separation (stroke type labels given) and the other with a stroke classification module. The results show that the proposed text line grouping method is robust for both the two cases. This paper is an extension to a conference paper [15]. The extension is in several respects: more details of techniques description, improved features in MCE training, extensive performance metrics, experimental results and discussions. The rest of this paper is organized as follows: Section 2 gives an overview of our ink document analysis system. Section 3 details the text line grouping approach. Section 4 describes the performance evaluation metrics. Section 5 presents the experimental results and Section 6 offers our concluding remarks.

2. System overview

Text line grouping is one of the key parts of our online handwritten document analysis system (Fig. 1). After text/non-text separation by stroke classification, the text strokes are grouped into text lines. Some misclassified strokes can be corrected in the process of text line grouping. Last, each text line is recognized using a character string recognition algorithm [11]. The flows between the three parts in Fig. 1 are bidirectional. The stroke classification module may leave behind misclassified strokes, which can be corrected in text line grouping utilizing the temporal and spatial relationship of text lines and strokes. Similarly, without the help of character string recognition, the text line grouping module cannot give perfect text line partitioning. Possibly, multiple candidates of text line partitioning can be verified by string recognition to select a most plausible partitioning.

Stroke type classification is a feasible way for separating the text and non-text (drawing) regions in freeform documents, where little prior knowledge is available for top–down parsing. Considering that spatially adjacent strokes usually have the same type, we have proposed a stroke classification method based on the Markov random field (MRF) framework [16], which results in significant improvement of accuracy compared to individual stroke classification. However, stroke classification errors still remain. The following text line grouping process can correct some misclassified strokes.

The text line grouping process consists of five stages from temporal to spatial. First, to alleviate the computation burden, consecutive strokes with small off-stroke distance are merged as blocks. Then, the pre-segmentation module performs coarse segmentation of the block sequence according to off-stroke distance at the risk of merge errors, and the over-merged text lines are split at the temporal segmentation stage. Due to misclassification of text strokes as non-text ones, a genuine text line may be over-segmented into multiple lines. At the temporal merge stage, these over-segmented text lines are connected, and meanwhile, misclassified strokes are corrected. The temporal merge module uses an SVM classifier for making merge/non-merge decisions. It can perform either on text lines or on stroke blocks, but merge/non-merge decisions on text lines are more reliable than those on stroke blocks. Hence, the preceding pre-segmentation and temporal segmentation steps are beneficial. Last, a spatial merge module is designed to merge the delayed strokes and the collinear text lines which were separated in the temporal grouping process.
3.1. Block grouping

The strokes are initially grouped into small blocks according to off-stroke distances. The succeeding grouping based on blocks other than strokes can significantly reduce the computation cost. An off-stroke is defined as the vector from the ending point of the preceding stroke to the starting point of the succeeding stroke. If the off-stroke distance between two successive strokes is smaller than a threshold, and the two strokes are not separated spatially by long non-text strokes, they are marked as belonging to the same block. The threshold should be small enough to guarantee that strokes belonging to different text lines are assigned to different blocks. We empirically set the threshold to be 0.3 times the average character size in our experiments. Strokes of different types (text or non-text) cannot be merged into one block, so the blocks can also be labeled as text or non-text. After block grouping, we get a sequence of stroke blocks, which are further grouped in following steps.

3.2. Pre-segmentation

For pre-segmentation, we first define the off-stroke between two consecutive blocks as the off-stroke between the last stroke of the preceding block and the first stroke of the succeeding block. Off-strokes within a text line are usually shorter than those between text lines. If the off-stroke between two consecutive text blocks is longer than a threshold, or the two blocks are separated by a long non-text stroke, this off-stroke is regarded as a segmentation position. The threshold is empirically set as five times the average character size in our experiments to guarantee merging within-line blocks but risk over-merging multiple text lines with short off-strokes between them. On splitting the sequence of blocks at segmentation positions, each sub-sequence is to be split into text lines in succeeding temporal segmentation considering the linearity of stroke blocks.

3.3. Temporal segmentation

After pre-segmentation, each sub-sequence of stroke blocks (also called a text line string) can be split into multiple text lines at internal off-strokes (candidate separation points). Since local information in the block sequence is not reliable enough to segment the sequence, we adopt a classification-based method with string-level training which can evaluate the segmentation with a global objective function. To segment the sequence, each off-stroke between blocks can be taken as a candidate separation point, and the combination of all the candidate separation points form a candidate lattice (Fig. 3), where each edge represents a candidate text line and a path from the start to the end represents a partitioning of text lines. The paths are evaluated using a trained discriminant function with the global features of the segmentation as inputs and the optimal path is obtained by beam search. At the end of this step, the text lines that have very small size and do not overlap with other lines are removed as non-text.

3.3.1. Segmentation candidate lattice

Inspired from character string recognition by integrated segmentation and recognition approach [10,11], which converts the string recognition problem into optimal path search in the segmentation candidate lattice, we build a lattice for a sequence of stroke blocks (text line string), in which candidate text lines composed of consecutive blocks correspond to candidate character patterns in string recognition (Fig. 3). Denoting the number of stroke blocks in the text line string by \( n \), the block number of the candidate text lines in the segmentation candidate lattice ranges from 1 to \( n \) and the total number of candidate text lines is \( n(n+1)/2 \). Each path from the start node to the terminal node corresponds to a candidate segmentation (partitioning) of text lines, and the total number of partitioning possibilities is \( 2^n-1 \).

3.3.2. Path scoring and search

To distinguish the correct segmentation from the incorrect ones in the candidate lattice, we design a linear discriminant function to evaluate the goodness of the paths such that the correct path has the highest score. The discriminant function is defined as

\[
g(X,S) = \sum_{k=1}^{M} \lambda_k f_k(X,S) = A^T F, \tag{1}
\]

where \( X \) denotes the text line string, \( S \) is a segmentation of \( X \) into text lines, \( A = [\lambda_1, \lambda_2, \ldots, \lambda_M]^T \) is a set of weighting parameters, \( F = [f_1, f_2, \ldots, f_M]^T \) is the feature vector characterizing the segmentation \( S \), and \( M \) is the number of features.

With the discriminant function (1), to segment a text line string is to find the optimal path with maximum score:

\[
S^* = \arg \max_S g(X,S). \tag{2}
\]

To find the optimal path, the DP algorithm, which retains solely the optimal partial path among the partial paths ending at an intermediate node in the candidate lattice, is an effective search method.
Whereas DP requires that the discriminant function should be a summation of the terms along the path, we have to select the path features with summation nature (as used in our previous work [15]). For the features without this property, the DP algorithm does not guarantee finding the optimal path under the criterion (2). So we instead use beam search [10,11] to find the path of approximately maximum score in moderate computation time. In beam search, the partial search paths ending at an intermediate search node are sorted and only a limited number (five at maximum in our experiments) of partial paths with maximum partial scores are retained for extension.

3.3.3. Path features

We use six features for characterizing a segmentation of the text line string, some of them do not satisfy the summation nature. To satisfy the requirement of maximization in Eq. (2), all the features have negative values. The first feature is the summation of the linear regression error of each candidate text line along the segmentation path, which reflects the overall linearity of the composed text lines [5]:

\[ f_1(X, S) = - \alpha \sum_{i=1}^{n} e(l_i), \]

where \( n \) denotes the number of text lines in path \( S \), \( l_i \) is the \( i \)-th text line in \( S \), and \( e(l_i) \) is the linear regression error of \( l_i \). To calculate the linear regression error, a candidate text line is fitted by LS model with the vertexes of the convex hull of the stroke blocks.

The text line number which is used to depict the model complexity in Ref. [5] is selected as the second feature:

\[ f_2(X, S) = - n, \]

This feature is effective to prevent over-segmentation.

For the segmentation of text line strings, it is intuitively hoped that the height of each text line should not be too large, so we select the following feature to characterize this property:

\[ f_3(X, S) = - n \cdot \max_{1 \leq i \leq n} (H(l_i)), \]

where \( H(l_i) \) is the height of the bounding box of text line \( l_i \).

Note that the within-line block gaps and off-stroke distances are usually smaller than those between lines, so the following two features are used to depict the compactness of the text lines:

\[ f_4(X, S) = - n \cdot \max_{1 \leq i \leq n} (d_{\text{off}}(l_i)), \]

\[ f_5(X, S) = - n \cdot \max_{1 \leq i \leq n} (d_{\text{off}}(l_i)), \]

where \( d_{\text{off}}(l_i) \) denotes the maximum within-line gap of \( l_i \) along the fitting line, and \( d_{\text{off}}(l_i) \) denotes the maximum within-line off-stroke distance projected onto the fitting line (Fig. 4(a)).

The last feature is the summation of the intersection part area between the preceding text line (extended at the terminal end along its writing direction) and the succeeding one, which is formulized as

\[ f_6(X, S) = - \sum_{i=1}^{n} A(l_{i-1}, l_i), \]

where \( A(l_{i-1}, l_i) \) is the intersection part area (Fig. 4(b)), and \( A(l_0, l_1) = 0 \) is defined. This feature is effective to partition consecutive text lines which are not parallel.

For accelerating the convergence in training, all the features except the second one (text line number) are normalized by the square of average character size estimated for each text line string. With the above six features, the DP algorithm does not guarantee finding the optimal path with the maximum score defined in Eq. (1), so we use the beam search strategy.

Compared to the optimization based segmentation method described in Ref. [5], where the cost function has hand-tuned parameters, in our system, the parameters in Eq. (1) are estimated by string-level MCE training.

3.3.4. String-level MCE training

The objective of training is to tune the weighting parameters such that correct segmentations of text line strings are encouraged while the incorrect ones are discouraged. The segmentation of text line strings can be viewed as a string classification problem: a path (partitioning of text lines) in the candidate segmentation lattice is judged to be correct or not, or classified to one of \( 2^n-1 \) classes (\( n \) is the number of stroke blocks). Yet for text line strings with arbitrary block number, the total number of string classes will be huge. The MCE method of Juang et al. [8] can handle this situation efficiently, because it can approximate the classification error with the discriminant values of only two classes: the correct class and the closest rival class.

In string-level training, the weighting parameters are estimated on a dataset of string samples \( D_X = \{X^n, S^n \mid n = 1, \ldots, N_X \} \), where \( S^n \) denotes the correct segmentation (corresponding to a set of separation points) of the sample \( X^n \), by optimizing an objective function related to string recognition performance. Because the number of string classes is huge, the misclassification measure considers only a finite number of classes (N-best list, including the genuine class and \( N-1 \) competing classes) [9]. Each class \( S \) of the N-best has a discriminant function \( g(X, S, A) \), where \( A \) is the parameter set.

In string-level MCE training described in Ref. [9], the misclassification measure for correct string class \( S_c \) is defined by

\[ d(X, A) = -g(X, S_c, A) + \log \left\{ \frac{1}{N-1} \sum_{S \neq S_c} e^{\eta g(X, S, A)} \right\}, \]

where \( \eta \) is a positive number. When \( \eta \to \infty \), the misclassification measure becomes

\[ d(X, A) = -g(X, S_c, A) + g(X, S_r, A), \]

where \( g(X, S_r, A) \) is the score of the closest rival class:

\[ g(X, S_r, A) = \max_{k \neq c} g(X, S_k, A). \]

In practice, this simplification does not influence the recognition performance, but significantly reduces the computation effort in training by stochastic gradient decent [17].

The misclassification measure is transformed to loss function by the sigmoidal function

\[ l(X, A) = \frac{1}{1 + e^{-\eta g(X, A)}}, \]
where $\xi$ is a parameter that controls the hardness of nonlinearity. On the training sample set, the empirical loss is

$$L(A) = \frac{1}{N_L} \sum_{n=1}^{N_L} L(X,A). \quad (13)$$

In minimizing $L(A)$ by stochastic gradient decent, the training patterns are fed into the classifier repeatedly. On a training pattern, the classifier parameters are updated by

$$A(t + 1) = A(t) - \alpha(t) \nabla L(X,A), \quad (14)$$

where $U$ is a positive definite matrix, $\alpha(t)$ is the learning step, $U$ is related to the inverse of Hessian matrix and is usually approximated to a diagonal matrix. In this case, the diagonal elements $U$ are absorbed into the learning step. The parameters converge to a local minimum of $L(A)$ (where $\forall l(X,A) = 0$) under the following conditions:

$$\lim_{t \to \infty} \alpha(t) = 0, \quad \sum_{t=1}^{\infty} \alpha(t) = \infty, \quad \sum_{t=1}^{\infty} \alpha^2(t) < \infty.$$ 

In practice, the parameters are updated in finite iterations, and setting the learning rate as a sequence starting with a small value and vanishing gradually leads to convergence approximately.

In MCE training for text line string segmentation, each training sample is segmented by comparing the path scores with the current parameters $A(t)$. By beam search, we obtain the rival segmentation which is most confusable with the correct one. Each segmentation, the correct or the rival, has a candidate text line sequence from which the features are extracted. Given the correct and rival segmentations $S_c$ and $S_r$, the corresponding feature vectors $F_c$ and $F_r$, the classification loss (misclassification measure) is defined as

$$d(X,A) = -g(X,S_c,A) + g(X,S_r,A) = A^T(F_r - F_c). \quad (15)$$

By stochastic gradient decent, the parameters of the discriminant function are updated on training sample $X$ by

$$A(t + 1) = A(t) - \alpha(t) \frac{\partial L(X,A)}{\partial A}\bigg|_{A=A(t)}$$

$$= A(t) - \alpha(t) \varepsilon (1 - l) \frac{\partial d(X,A)}{\partial A}\bigg|_{A=A(t)}$$

$$= A(t) - \alpha(t) \varepsilon (1 - l) (F_r - F_c). \quad (16)$$

### 3.4. Temporal merge

After stroke classification (prior to text line grouping), some text strokes are misclassified as non-text ones, which will split a text line into multiple ones. The temporal merge module is designed to correct such stroke classification errors and merge the over-segmented text lines. For this stage, an SVM classifier is trained to make the merge/non-merge decision for each hypothesis. The temporal merge algorithm is inspired by the HCF method [6] which was proposed for grouping text lines and writing regions based on a neighborhood graph and an AdaBoost classifier.

#### 3.4.1. Temporal merge procedure

The temporal merge algorithm proceeds iteratively. Each time we select the longest text line which has not been checked. The selected text line has a preceding neighbor and a succeeding neighbor, which is either a text line or a non-text block. We extract features for the pair of the selected line with its predecessor and the pair with its successor (two hypotheses of merge). The features are fed to an SVM classifier for merge/non-merge decision. If one or two pairs are decided “merge”, we accept the hypothesis with the highest confidence and merge the pair. The merged text line is then recursively paired with its predecessor and successor for hypothesis until it is decided “non-merge”. The “non-merge” text line is marked “checked”. Recursively, the next unchecked longest line is selected for hypothesis of merge, until all the text lines are marked.

#### 3.4.2. Training sample collection

To train the SVM classifier, we need to collect training samples (hypotheses) which could be a pair of two successive text lines or a pair of a text line and a non-text block (which may be a text block misclassified to be non-text), labeled as “merge” or “non-merge”. The samples are collected by running the merge procedure from the temporal segmentation results. Whenever a pair of neighboring text lines or a pair of text line and a non-text block is hypothesized, we label the pair as a positive (merge) sample if the pair belong to a ground-truthed text line, otherwise a negative (non-merge) sample.

#### 3.4.3. Feature extraction

Before feature extraction, each sample, a pair of text lines or a pair of a line and a non-text block, are tentatively merged and fitted by LS regression. In our previous work [15], eight features are extracted from each sample, the first three are: the average regression error (regression error divided by the number of convex hull vertexes), the squared maximum inner distance along the fitting direction, and the squared maximum stroke length. These three features are normalized by the squared average character size. The 4–7th features are the sine and cosine of the fitted line direction, and the sine and cosine of the angle difference between the fitted line direction and the longer text line direction in the pair. Another important feature is the difference between the outputs of the discriminant function (1) for the merged line and two separate ones. From Section 3.3, we can see that if the difference is positive, it is likely to merge the pair, otherwise, they are likely to be separated.

Besides the above eight features, three new ones are added. The first two are the maximum within-line gap and off-stroke distance projected onto the fitting line, as detailed in Section 3.3.3. For the third new feature, we check the text line (or non-text block) with fewer strokes in the pair, to see whether it has a stroke intersecting with other non-text strokes or not (1 or 0). A value 1 implies that the pair is unlikely to be merged. Now, there are totally 11 features used in temporal merge.

### 3.5. Spatial merge

In handwritten documents, strokes are mostly written in character order, but there are still some delayed strokes, which are added to a former character after a later character of the text line is written. In temporal grouping of text lines, such delayed strokes cause two types of segmentation errors: Case A and Case B (Fig. 5). In Case A, the block (or short text line) of delayed strokes is embraced by a long text line. And in Case B, the ends of two collinear text lines are close to each other but are temporally separated by delayed strokes. The spatial merge module is intended to merge such over-segmented text lines.

At the spatial merge stage, we first process Case A and then Case B, because a short text line embraced in a longer one is more likely to be merged. The spatial merge procedure still adopts the HCF method [6], and we need to build the neighborhood graph to generate and update the hypotheses and train a classifier to produce the confidence for each hypothesis.
3.5.1. Discriminant function

Considering the fact that there are not many instances requiring spatial merge and consequently we have only a small number of training samples, a simple linear discriminant function is trained for the two cases on the training samples collected for temporal segmentation, and only the first four features (the regression error, the text line number, the height of the text line bounding box and the maximum within-line gap) of those mentioned in Section 3.3.3 are included, i.e.,

\[ g(X,S) = \sum_{k=1}^{4} \lambda_k f_k(X,S). \]  

(17)

The parameters are estimated under the string-level MCE criterion.

3.5.2. Neighborhood graph

For Case A, we build the neighborhood graph according to the intersection ratio between two text lines. The intersection ratio is defined as the area of the common part between two text line bounding boxes divided by the minimum area of the two bounding boxes. And for Case B, the neighborhood graph is built according to spatial line distance, which is defined as the minimum of the two distances between the start end of one line and the terminal end of the other. To avoid over-merge, we build the neighborhood graphs rather conservatively: two text lines are said to be neighbors if the intersection ratio is larger than 0.7 for Case A or the spatial line distance is smaller than 0.5 times the average character size for Case B.

3.5.3. Spatial merge procedure

The spatial merge algorithm is an iterative procedure. For each pair of neighboring text lines, we compute the features for both “merge” and “non-merge”, and calculate the difference between the discriminant function outputs of the two hypotheses. This difference can be viewed as the confidence for merge/non-merge decision. From the description in Section 3.3, we know that if the difference is positive, the hypothesis is decided as “merge”, otherwise as “non-merge”. If a merge decision is made, the hypothesis is added to a queue together with its confidence score. After all the hypotheses in the graph are scored, the hypothesis with the highest confidence is accepted, and accordingly the neighborhood graph and hypotheses queue are updated. This operation iterates until the queue becomes empty.

4. Performance evaluation

Systematic evaluation metrics are needed to measure the overall performance of text line grouping algorithms. We first define matches and errors between the text lines detected by the algorithm (result lines) and those in the ground-truth (ground-truthed or labeled lines), based on which a set of evaluation metrics are calculated. Some of the metrics are originally proposed to evaluate graphics recognition systems [13] and further used in the ICDAR page segmentation competitions and handwriting segmentation contest [14].

4.1. Matches and errors

If a result line and a ground-truthed line contain identical strokes, the match between them is called one-to-one match. A g_segmentation error and two d_segmentation errors, (b) two g_merge errors and one d_merge error, and (c) One g_merge error and one d_merge error, two non-text strokes (the arrow) are misclassified as text and wrongly merged with a text line.
4.2. Performance metrics

After the definition of the matches and errors, we can define the metrics for performance evaluation. Let one2one, g_one2many, d_many2one, misses, false_alarms, g_segmentation, d_segmentation, g_merge and d_merge denote the counts of the corresponding matches and errors, \( N \) is the count of ground-truthed text lines, and \( M \) is the count of detected text lines. Below, we define the performance metrics.

Detection rate (DR)

\[
DR = \frac{\text{one2one}}{N} + \frac{g\_\text{one2many}}{N}.
\]

DR is the percentage of ground-truthed text lines that are detected by the grouping algorithm. The pre-determined coefficients \( \omega_1 \) and \( \omega_2 \) are used to weight the significance of the two terms. In our experiments, they are both set to 1 because both one-to-one and g_one-to-many matches are proper for character string recognition. It is easy to merge the multiple result lines that match a ground-truthed line. DR is similar to the recall rate in the context of information retrieval.

Missed detection rate (MDR)

\[
MDR = \frac{\text{misses}}{N}.
\]

MDR is the percentage of ground-truthed text lines that are not detected by the grouping algorithm.

Recognition accuracy (RA)

\[
RA = \frac{\text{one2one}}{M} + \frac{d\_\text{many2one}}{M}.
\]

RA indicates the percentage of detected text lines that match with ground-truthed lines. We again set \( \omega_3 \) and \( \omega_4 \) equally to 1. RA is similar to the precision in the context of information retrieval.

False-alarm rate (FAR)

\[
FAR = \frac{\text{false\_alarms}}{M}.
\]

FAR is the percentage of detected text lines that do not match with any text line in the ground-truth data.

Entity detection metric (EDM)

\[
EDM = \frac{2 \cdot DR \cdot RA}{DR + RA}.
\]

By combining the values of DR and RA, EDM is defined as an overall performance measure. EDM is similar to the F1 measure in the context of information retrieval.

Edit cost index (ECI)

\[
ECI = 1 - \frac{2 \cdot \text{one2one}}{N + M}.
\]

ECI, whose values range from 0 to 1, measures the human post-editing effort to make the grouping result and the ground-truth match perfectly. The lower the value, the less post-editing one has to do, and the more matches are one-to-one.

Edit-distance rate (EDR)

\[
EDR = \frac{d\_\text{segmentation} + d\_\text{merge}}{N}.
\]

The edit-distance between the detected text lines and the ground-truth is the minimum number of split or merge operations needed to correct all errors, and EDR is the normalization of edit-distance, which measures the relative edit cost compared to the ground-truthed line number [6].

### Table 1: Specification of training and test set

<table>
<thead>
<tr>
<th>Data sets</th>
<th>#Page</th>
<th>#Text-dominant</th>
<th>#Mixed</th>
<th>#Text lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>2043</td>
<td>1680</td>
<td>363</td>
<td>15,679</td>
</tr>
<tr>
<td>Test</td>
<td>1560</td>
<td>1120</td>
<td>440</td>
<td>11,868</td>
</tr>
</tbody>
</table>

Segmentation error rate (SER)

\[
SER = \frac{2 \cdot g\_\text{segmentation} \cdot d\_\text{segmentation}}{M \cdot g\_\text{segmentation} + N \cdot d\_\text{segmentation}}.
\]

SER measures the average percentage of segmentation errors in the matches between the ground-truth and the detected result.

Merge error rate (MER)

\[
MER = \frac{2 \cdot g\_\text{merge} \cdot d\_\text{merge}}{M \cdot g\_\text{merge} + N \cdot d\_\text{merge}}.
\]

MER measures the average percentage of merge errors in the matches between the ground-truth and the detected text lines.

Among the metrics, DR, RA and EDM are used to measure the accuracies, MDR, FAR, SER and MER are used to measure the errors, and ECI as well as EDR are used to measure the post-edit cost. EDM and EDR are taken as overall performance measures.

5. Experiments

To evaluate the performance of the proposed text line grouping approach, we have experimented on the TUAT HANDS-Kondate_tbf-2001-11 (in brief, Kondate) database, of online freeform handwritten Japanese documents from 100 people without any writing constraints [7]. We have also compared our temporal segmentation method with a previous method proposed in Ref. [7].

5.1. Database and experimental setting

The TUAT Kondate database contains text lines in arbitrary directions and font sizes, and often mixed with non-text structures, such as drawings, table forms and flowcharts. The stroke types cover text, formula, figure, ruled line and editing mark. The formulas, which contain both characters and non-characters and were labeled as non-text strokes in the database, are excluded in our experiment, thus the non-text strokes are composed of figure, ruled line and editing mark strokes.

We used the first 60 files (corresponding to 60 writers) for training (MCE and SVM) and the left 40 files for testing. The documents contain both text and non-text strokes. Some documents are dominant of text strokes: some with all text strokes, some with a few editing marks only. All the text lines in these pages have been manually labeled, for training and evaluation, respectively. The details of the training and test data are listed in Table 1 and some examples of pages are shown in Fig. 7.

After pre-segmentation of the training pages, 12,896 text line strings were selected for string-level MCE training of linear discriminant function. Similar to Ref. [18], the parameter \( \xi \) in loss function (11) was set to be proportional to the reciprocal of the scatter estimated on the samples with genuine segmentation. The string samples were processed iteratively for five cycles in stochastic gradient decent. After the last cycle of training, the accuracy on the training samples is 0.991, and the parameters in Eq. (1) are as follows: \( \lambda_1 = 0.2252, \lambda_2 = 2.5754, \lambda_3 = 1.2249, \lambda_4 = 0.4224, \lambda_5 = 0.0672, \lambda_6 = 0.0891 \). These values indicate that the first four features are relatively more important than the last two ones in temporal segmentation.

The SVM classifier for temporal merge, with fourth order polynomial kernel, was trained on 30,183 samples (25,848 merge and 4335
non-merge). For the classifier training in spatial merge, the samples used and the settings for the training procedure are identical to the string-level MCE training in temporal segmentation.

The system was implemented in Microsoft Visual C++ 6.0 and all the experimental results were produced on an AMD Dual Core 3800+ CPU 1 GB-RAM PC.

Since text line grouping is described as a module with the stroke classification results as inputs in our ink document analysis system mentioned in Section 2, we first evaluate its performance with a stroke classification preprocessor [16]. As a comparison, the system is then evaluated with the genuine stroke labels given. The above two settings are called in Ref. [5] crude writing/drawing (CrudeWD) input and perfect writing/drawing (PerfectWD) input, respectively. Temporal merge is intended for CrudeWD to correct the over-segmentation errors caused by misclassification of strokes. It is not necessary for PerfectWD since there is no stroke classification error causing over-segmentation of text lines, while the segmentation errors caused by delayed strokes can be corrected in the spatial merge stage.

5.2. Results and discussions

Using the proposed approach with trained decision functions, our grouping results are listed in Tables 2 and 3 for CrudeWD and PerfectWD, respectively, and some examples of the grouping process are illustrated in Figs. 8 and 9.

In Tables 2 and 3, each row gives the results after the corresponding operation. Each column, from the second to the tenth, corresponds to a metric mentioned in Section 4, and the last column (T) is the time cost (seconds per page) for each operation.

From the results in Tables 2 and 3 we can see that:

1. Comparing the metrics between consecutive operations, the system performance is improved step by step for both CrudeWD and PerfectWD. At the last step, the highest EDM rate and the lowest EDR rate are achieved. And from the ECI values, we can see that most of the matches are one-to-one.

2. With the stroke classification accuracy 0.9779 on the test set by the MRF based algorithm, the system performs much better for PerfectWD, which confirms that the stroke classification errors affect the text line grouping accuracy significantly.

3. By comparing MER and SER, we can see that pre-segmentation brings in merge errors more than segmentation errors, for the segmentation threshold is selected to be larger than most of the within-line distances, and is large enough to prevent segmentation within characters. Note that the high SER after pre-segmentation in Table 2 is caused by stroke classification.

4. Temporal segmentation can effectively correct the merge errors while bringing in minor segmentation errors.

5. Temporal merge operation can significantly reduce the segmentation errors caused by stroke classification, with the MER increased slightly. By merging the misclassified strokes to text lines, the stroke classification accuracy is improved from 0.9779 to 0.9940 after temporal merge.

### Table 2

Text line grouping results for CrudeWD

<table>
<thead>
<tr>
<th>Operations</th>
<th>DR</th>
<th>MDR</th>
<th>RA</th>
<th>FAR</th>
<th>EDM</th>
<th>ECI</th>
<th>EDR</th>
<th>SER</th>
<th>MER</th>
<th>T(s/P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-segmentation</td>
<td>0.6611</td>
<td>0.0050</td>
<td>0.5752</td>
<td>0.0466</td>
<td>0.6152</td>
<td>0.3944</td>
<td>0.4710</td>
<td>0.2275</td>
<td>0.1082</td>
<td>0.0188</td>
</tr>
<tr>
<td>Temporal segmentation</td>
<td>0.7760</td>
<td>0.0075</td>
<td>0.6563</td>
<td>0.0258</td>
<td>0.7111</td>
<td>0.2987</td>
<td>0.4102</td>
<td>0.2304</td>
<td>0.0413</td>
<td>0.5679</td>
</tr>
<tr>
<td>Temporal merge</td>
<td>0.9052</td>
<td>0.0072</td>
<td>0.8822</td>
<td>0.0279</td>
<td>0.8936</td>
<td>0.1195</td>
<td>0.1271</td>
<td>0.0563</td>
<td>0.0537</td>
<td>0.0054</td>
</tr>
<tr>
<td>Spatial merge</td>
<td>0.9047</td>
<td>0.0072</td>
<td>0.8939</td>
<td>0.0242</td>
<td><strong>0.8992</strong></td>
<td>0.1097</td>
<td><strong>0.1114</strong></td>
<td>0.0450</td>
<td>0.0582</td>
<td>0.0009</td>
</tr>
</tbody>
</table>

### Table 3

Text line grouping results for PerfectWD

<table>
<thead>
<tr>
<th>Operations</th>
<th>DR</th>
<th>MDR</th>
<th>RA</th>
<th>FAR</th>
<th>EDM</th>
<th>ECI</th>
<th>EDR</th>
<th>SER</th>
<th>MER</th>
<th>T(s/P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-segmentation</td>
<td>0.8024</td>
<td>0.0000</td>
<td>0.8891</td>
<td>0.0012</td>
<td>0.8435</td>
<td>0.1702</td>
<td>0.1219</td>
<td>0.0277</td>
<td>0.1160</td>
<td>0.0186</td>
</tr>
<tr>
<td>Temporal segmentation</td>
<td>0.9547</td>
<td>0.0000</td>
<td>0.9494</td>
<td>0.0011</td>
<td>0.9520</td>
<td>0.0656</td>
<td>0.0870</td>
<td>0.0424</td>
<td>0.0246</td>
<td>0.5926</td>
</tr>
<tr>
<td>Spatial merge</td>
<td>0.9573</td>
<td>0.0000</td>
<td>0.9609</td>
<td>0.0010</td>
<td><strong>0.9591</strong></td>
<td>0.0530</td>
<td><strong>0.0669</strong></td>
<td>0.0298</td>
<td>0.0265</td>
<td>0.0009</td>
</tr>
</tbody>
</table>

![Fig. 7. Examples of pages in TUAT Kondate database: (a) a text-dominant page and (b) a page mixed of text and non-text strokes.](image-url)
Fig. 8. Grouping process for CrudeWD. The grouping results are bounded with red rectangular boxes: (a) stroke classification results, (Blue: text strokes; gray: non-text strokes.) (b) pre-segmentation results, (c) temporal segmentation results, and (d) temporal merge results.

6. The number of spatial merge to amend the segmentation errors is relatively small, because there are not many instances (delayed strokes) requiring spatial merge in Japanese documents.

7. Compared to the MDRs, the FARs are much larger in Table 2, because the non-text strokes are more likely to be misclassified than the text strokes by our stroke classification algorithm. The nonzero FARs in Table 3 owe much to label noise, e.g. wild points or strokes which are labeled as non-text in ground-truth.

8. Compared to the other operations, the temporal segmentation step is relatively time consuming. The total processing time spent is less than one second per page for both CrudeWD and PerfectWD.

Compared to our previous work [15], the system performance is improved remarkably with the newly added features in temporal segmentation and merge steps.

In the test pages, 68 pages have whirled or curved text lines (Fig. 10). For such cases, it is difficult to determine the text line boundaries even manually. By our text line grouping approach, the segmentation and merge errors are inevitable, yet the detected piecewise linear test lines are meaningful. Without these pages, the grouping results for CrudeWD and PerfectWD are listed in Tables 4 and 5, respectively.

5.3. Comparison with previous methods

Our experimental results cannot be compared with those reported in the literature because there is not a common database for evaluation. For reference, in text line grouping on English documents with PerfectWD input, Ye et al. reported EDRs of 0.1494 and 0.1633 for an optimization based method and a machine learning based method, respectively [6]. Our approach achieved an EDR of 0.0669 on Japanese documents.

As a comparison to the temporal segmentation module, we have implemented the method proposed in Ref. [7] which can deal with text lines with arbitrary writing directions. By this approach, for each text line string after pre-segmentation, we first find the block with maximum distance to the straight line connecting the starting and ending points of the text line string. If the maximum distance is larger than a pre-defined threshold, the text line string is tentatively partitioned at the off-stroke between the found block and its succeeding one. The separation proceeds recursively until the maximum distance is smaller than the threshold for each result text line. Then, from the first text line, we calculate the bounding box along its writing direction (fitted by LS model), and recursively merge the first block of the next line to it if the block falls into the extension area of the first text line’s bounding box. This step is performed in turn for each text line. We tuned the pre-defined threshold on our test set to get the best EDM rate. To draw a fair comparison, we just replaced the temporal segmentation step with this method in our system. The grouping results with the new temporal segmentation step are listed in Table 6, in which the first column lists the indices of Tables 2–5 corresponding to the settings (Crude and PerfectWD, with and without curved text lines). Table 7 lists the final results after
Fig. 9. Grouping process for PerfectWD. The grouping results are bounded with red rectangular boxes: (a) ink page, (b) pre-segmentation results, (c) temporal segmentation results, and (d) spatial merge results.

Fig. 10. Examples of whirled handwriting (a) and the temporal segmentation result (b).

Table 4
Text line grouping results for CrudeWD (without whirled and curved text lines)

<table>
<thead>
<tr>
<th>Operations</th>
<th>DR</th>
<th>MDR</th>
<th>RA</th>
<th>FAR</th>
<th>EDM</th>
<th>ECI</th>
<th>EDR</th>
<th>SER</th>
<th>MER</th>
<th>T(s/P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-segmentation</td>
<td>0.6850</td>
<td>0.0051</td>
<td>0.5830</td>
<td>0.0476</td>
<td>0.6299</td>
<td>0.3801</td>
<td>0.4680</td>
<td>0.2268</td>
<td>0.0973</td>
<td>0.0189</td>
</tr>
<tr>
<td>Temporal segmentation</td>
<td>0.7930</td>
<td>0.0077</td>
<td>0.6674</td>
<td>0.0269</td>
<td>0.7248</td>
<td>0.2849</td>
<td>0.3856</td>
<td>0.2207</td>
<td>0.0283</td>
<td>0.5132</td>
</tr>
<tr>
<td>Temporal merge</td>
<td>0.9261</td>
<td>0.0074</td>
<td>0.8988</td>
<td>0.0290</td>
<td>0.9122</td>
<td>0.1004</td>
<td>0.0973</td>
<td>0.0426</td>
<td>0.0394</td>
<td>0.0051</td>
</tr>
<tr>
<td>Spatial merge</td>
<td>0.9252</td>
<td>0.0074</td>
<td>0.9101</td>
<td>0.0252</td>
<td><strong>0.9176</strong></td>
<td>0.0911</td>
<td><strong>0.0825</strong></td>
<td>0.0317</td>
<td>0.0439</td>
<td>0.0009</td>
</tr>
</tbody>
</table>

spatial merge with the new temporal segmentation step. Note that in Table 6, the last column (T) shows the time cost of the new temporal segmentation step, while in Table 7, the last column gives the total time cost of grouping. Comparing the metrics of Tables 6 and 7 against the corresponding results of temporal segmentation and spatial merge in Tables 2–5, it is evident that our proposed method yields much higher accuracies. On the other hand, the method in Ref. [7] runs very fast.
Table 5
Text line grouping results for PerfectWD (without whirled and curved text lines)

<table>
<thead>
<tr>
<th>Operations</th>
<th>DR</th>
<th>MDR</th>
<th>RA</th>
<th>FAR</th>
<th>EDM</th>
<th>ECI</th>
<th>EDR</th>
<th>SER</th>
<th>MER</th>
<th>T(s/P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-segmentation</td>
<td>0.8307</td>
<td>0.0000</td>
<td>0.8972</td>
<td>0.0011</td>
<td>0.8627</td>
<td>0.1514</td>
<td>0.1169</td>
<td>0.0271</td>
<td>0.1038</td>
<td>0.0191</td>
</tr>
<tr>
<td>Temporal segmentation</td>
<td>0.9752</td>
<td>0.0000</td>
<td>0.9661</td>
<td>0.0010</td>
<td>0.9707</td>
<td>0.0467</td>
<td>0.0593</td>
<td>0.0302</td>
<td>0.0102</td>
<td>0.5440</td>
</tr>
<tr>
<td>Spatial merge</td>
<td>0.9776</td>
<td>0.0000</td>
<td>0.9774</td>
<td>0.0010</td>
<td>0.9775</td>
<td>0.0342</td>
<td>0.0397</td>
<td>0.0179</td>
<td>0.0120</td>
<td>0.0010</td>
</tr>
</tbody>
</table>

Table 6
Text line grouping results after temporal segmentation for the settings in Tables 2–5 (with the temporal segmentation method proposed in Ref. [7])

<table>
<thead>
<tr>
<th>Table</th>
<th>DR</th>
<th>MDR</th>
<th>RA</th>
<th>FAR</th>
<th>EDM</th>
<th>ECI</th>
<th>EDR</th>
<th>SER</th>
<th>MER</th>
<th>T(s/P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 (CrudeWD)</td>
<td>0.7298</td>
<td>0.0075</td>
<td>0.6178</td>
<td>0.0243</td>
<td>0.6622</td>
<td>0.3769</td>
<td>0.2427</td>
<td>0.0624</td>
<td>0.0005</td>
<td></td>
</tr>
<tr>
<td>3 (PerfectWD)</td>
<td>0.8933</td>
<td>0.0000</td>
<td>0.8904</td>
<td>0.0010</td>
<td>0.8945</td>
<td>0.1779</td>
<td>0.2494</td>
<td>0.0124</td>
<td>0.0004</td>
<td></td>
</tr>
<tr>
<td>4 (CrudeWD)</td>
<td>0.7482</td>
<td>0.0077</td>
<td>0.6319</td>
<td>0.0254</td>
<td>0.6851</td>
<td>0.3613</td>
<td>0.5124</td>
<td>0.2862</td>
<td>0.0004</td>
<td></td>
</tr>
<tr>
<td>5 (PerfectWD)</td>
<td>0.9228</td>
<td>0.0000</td>
<td>0.9125</td>
<td>0.0010</td>
<td>0.9176</td>
<td>0.1561</td>
<td>0.2099</td>
<td>0.1149</td>
<td>0.0366</td>
<td>0.0003</td>
</tr>
</tbody>
</table>

Table 7
Text line grouping results after spatial merge for the settings in Tables 2–5 (with the temporal segmentation method proposed in Ref. [7])

<table>
<thead>
<tr>
<th>Table</th>
<th>DR</th>
<th>MDR</th>
<th>RA</th>
<th>FAR</th>
<th>EDM</th>
<th>ECI</th>
<th>EDR</th>
<th>SER</th>
<th>MER</th>
<th>T(s/P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 (CrudeWD)</td>
<td>0.8587</td>
<td>0.0072</td>
<td>0.8528</td>
<td>0.0235</td>
<td>0.8557</td>
<td>0.1544</td>
<td>0.1684</td>
<td>0.0744</td>
<td>0.0838</td>
<td>0.0242</td>
</tr>
<tr>
<td>3 (PerfectWD)</td>
<td>0.9053</td>
<td>0.0000</td>
<td>0.9115</td>
<td>0.0010</td>
<td>0.9084</td>
<td>0.1343</td>
<td>0.1749</td>
<td>0.0863</td>
<td>0.0568</td>
<td>0.0193</td>
</tr>
<tr>
<td>4 (CrudeWD)</td>
<td>0.8803</td>
<td>0.0073</td>
<td>0.8716</td>
<td>0.0245</td>
<td>0.8759</td>
<td>0.1338</td>
<td>0.1358</td>
<td>0.0707</td>
<td>0.0588</td>
<td>0.0248</td>
</tr>
<tr>
<td>5 (PerfectWD)</td>
<td>0.9287</td>
<td>0.0000</td>
<td>0.9329</td>
<td>0.0009</td>
<td>0.9308</td>
<td>0.1122</td>
<td>0.1408</td>
<td>0.0716</td>
<td>0.0399</td>
<td>0.0196</td>
</tr>
</tbody>
</table>

6. Conclusion

We presented a robust text line grouping approach for analyzing freeform online handwritten Japanese documents. After the relatively coarse pre-segmentation, we use a linear discriminant function trained under the string-level MCE criterion to separate over-merged text lines. Then the HCF method is employed for both temporal and spatial merge. To evaluate the performance, we give a set of metrics from multiple aspects. The experiments on the TUAT Kondate database demonstrate the effectiveness of our approach.

Our approach has no artificial parameters in the cost functions, which are trained in supervised learning (string-level MCE and SVM). We combine both temporal information and spatial information in a series of temporal operations and a spatial merge operation. The text line grouping process can effectively correct stroke classification errors left by text/non-text stroke classification preceding text line grouping. Our future work aims to further improve the system performance via integrated stroke classification and text line grouping, and interaction between text line grouping and character string recognition.

Acknowledgments

This work was partially supported by the Central Research Laboratory of Hitachi Ltd., Tokyo, Japan. The authors thank the Nakagawa Laboratory of Tokyo University of Agriculture and Technology (TUAT) for providing the Kondate database.

References

About the Author—XIANG-DONG ZHOU received the B.S. degree in Applied Mathematics and the M.S. degree in Management Science and Engineering both from National University of Defense Technology, Changsha, China, in 1998 and 2003, respectively. Currently, he is working toward the Ph.D. degree in Pattern Recognition and Intelligent System at National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing, China. His research interests include character string recognition and ink document analysis.

About the Author—DA-HAN WANG received the B.S. degree in Automation Science and Electrical Engineering from Beihang University, Beijing, China, in 2006 and in the same year he became a member of National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences, Beijing, China. Now he is still working toward the Ph.D. degree in NLPR, CASIA. His research interests is character recognition.

About the Author—CHENG-LIN LIU received the B.S. degree in Electronic Engineering from Wuhan University, Wuhan, China, the M.E. degree in Electronic Engineering from Beijing Polytechnic University, Beijing, China, the Ph.D. degree in Pattern Recognition and Artificial Intelligence from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 1949, 1992 and 1995, respectively. He was a postdoctoral fellow at Korea Advanced Institute of Science and Technology (KAIST) and later at Tokyo University of Agriculture and Technology from March 1996 to March 1999. From 1999 to 2004, he was a research staff member and later a senior researcher at the Central Research Laboratory, Hitachi, Ltd., Tokyo, Japan. From 2005, he has been a Professor at the National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences, Beijing, China, and is now the deputy director of the laboratory. His research interests include pattern recognition, image processing, neural networks, machine learning, and especially the applications to character recognition and document analysis. He has published over 70 technical papers at international journals and conferences. He won the IAPR/ICDAR Young Investigator Award of 2005.