A Segmentation Method of Single- and Multiple-Touching Characters in Offline Handwritten Japanese Text Recognition

Kha Cong NGUYEN\textsuperscript{a)}, Nonmember, Cuong Tuan NGUYEN\textsuperscript{b)}, Member, and Masaki NAKAGAWA \textsuperscript{c)}, Fellow

SUMMARY This paper presents a method to segment single- and multiple-touching characters in offline handwritten Japanese text recognition with practical speed. Distortions due to handwriting and a mix of complex Chinese characters with simple phonetic and alphanumeric characters leave optical handwritten text recognition (OHTR) for Japanese still far from perfection. Segmentation of characters, which touch neighbors on multiple points, is a serious unsolved problem. Therefore, we propose a method to segment them which is made in two steps: coarse segmentation and fine segmentation. The coarse segmentation employs vertical projection, stroke-width estimation while the fine segmentation takes a graph-based approach for thinned text images, which employs a new bridge finding process and Voronoi diagrams with two improvements. Unlike previous methods, it locates character centers and seeks segmentation candidates between them. It draws vertical lines explicitly at estimated character centers in order to prevent vertically unconnected components from being left behind in the bridge finding. Multiple candidates of separation are produced by removing touching points combinatorially. SVM is applied to discard improbable segmentation boundaries. Then, ambiguities are finally solved by the text recognition employing linguistic context and geometric context to recognize segmented characters. The results of our experiments show that the proposed method can segment not only single-touching characters but also multiple-touching characters, and each component in our proposed method contributes to the improvement of segmentation and recognition rates.

key words: handwritten text recognition, offline recognition, touching characters, stroke width, bridge separation, Voronoi diagram, Support Vector Machine

1. Introduction

The research on Optical Character Recognition (OCR) has a long history \cite{1, 2}. It started as printed character recognition, was extended to handwritten character recognition and printed document recognition, and then finally evolved into handwritten document recognition. The largest success was achieved in postal address recognition and form recognition of fixed formats.

When format and vocabulary are unrestricted, however, the OHTR is still a big challenge because of various distortions due to handwriting (variations of character size, unstable gaps between characters, touching of characters, irregular spacing between lines and so on). For Japanese OHTR, the problem is more difficult because different groups of characters are mixed with various sizes (Kanji of Chinese origin, Katakana and Hiragana of phonetic characters, alphabets, numerals, Greek letters, punctuation marks and so on) in a document without word spacings as shown in Fig. 1. Hiragana characters used as conjugation parts are often written many times smaller than Kanji characters. Moreover, many Chinese characters are composed of radicals, which themselves can be Chinese characters.

After achieving success with Postal code and address recognition, research and development spread to problems where format, vocabulary and/or layout were not restricted. Postal code and address recognition and some commercially successful applications were well-designed compromises between technologies and demands. Making a step ahead to the format-free and vocabulary-free handwritten Japanese text recognition, however, the challenge has not been met yet. Due to this difficulty, manual typing in low labor-cost countries is still continuing even today.

The diversity of character shapes, and distortion due to handwriting make deterministic segmentation and recognition difficult, and require efficient optimization strategies to segment and recognize Japanese handwritten text. This is the so-called over-segmentation approach, where over-segmented parts are combined and recognized by the best-path search for text recognition with character recognition, considering the linguistic and geometric contexts \cite{3}–\cite{5}. The linguistic context restricts the relation of characters with adjacent characters in daily contexts while the geometric context represents geometric relation between character patterns and within a character pattern.

Another approach to handwritten text recognition is the segmentation-free approach such as Messina et al. \cite{6}. This approach has become feasible for handwritten Chinese or Japanese text due to the progress of deep neural networks. For practical applications, however, problems remain in recognition speed, difficulty of analyzing misrecognitions, and so on. In any case, both the approaches are competitive. In this paper, we focus on the over-segmentation approach.

In almost all OHTR systems, text lines are segmented, and then characters are separated in each segmented text.
line. For text-line segmentation, current methods are classified into five groups: methods making use of projection profiles [7], those based on the Hough transform [8], smearing methods [9], component grouping methods [10] and other methods [11], [12]. We employ a combination of the smearing method and the method using projection profiles [5], which we do not discuss in this paper.

Here, we focus on character segmentation. Many methods have been proposed until now. Zhao et al. apply two stages of coarse segmentation and fine segmentation [13]. The first stage employs background skeleton and vertical projection while the second stage identifies segmentation candidates on the foreground skeleton. Fuzzy decision rules are also applied to the segmentation paths in the coarse and fine stages considering several geometric features. The method segments an oversized segment if the segment is separated by removal of a single path between two fork points. Since multiple-touching characters have multiple paths, however, it cannot segment multiple touches. Xu et al. propose a learning-based filter for segmenting Chinese handwriting [14], [15]. This method is again for single-touching Chinese characters. The method seeks a touching path for each oversized segment by tracing along its skeleton from left to right once, so that unable to find multiple touches.

In handwritten Japanese text recognition, Ikeda et al. employ stroke width analysis and projection image analysis in the over-segmentation and recognition approach for Japanese address reading [3]. Suwa et al. apply thinning, and formulate over-segmentation in a graph-cut framework. Actual pruning of touching-point candidates is done based on the geometric features of edges, vertices and connected components. They incorporate this method in their over-segmentation approach [16]. This method can theoretically segment multiple touches, but the proper selection of cutting candidates is not trivial. Moreover, it is not combined with context as mentioned in their remarks. Nishimura et al. adopt specific features to segment touching Kanji characters used in addresses [17]. Yamaguchi et al. apply thinning, and represent the thinned patterns as graphs. They prune the touching-point candidates by projection analysis and select the appropriate ones based on the configurations around each candidate point. Some restrictions on the number of touching points are introduced to decrease the candidates [18]. However, these last two proposals are not combined with recognition.

Although we cannot compare all these approaches fairly because their datasets are different and the programs are not available, segmentation without recognition seems hard for handwritten Japanese text recognition. Projection profile is useful but it is not adequate to locate touching points within allowable precision due to overhanging strokes and multiply connected strokes as discussed in [16]. A common problem with all these approaches is their weakness to handle multiple-touching characters.

In real samples as shown in Fig. 2 from OffTouchDB presented in Sect. 5, there are many multiple-touching characters as well as single-touching characters. They often occur when characters are written with a felt pen, pencil, or thick ball pen, when scanning resolution is low, when a document image is blurred and the banalization threshold is not optimal and so on. The conventional methods fail to segment them.

In general, the previous methods have been seeking to locate constricted parts, which are effective to find single touches but ineffective to find multiple touches. On the other hand, the proposed method seeks to locate centers of characters and finds all touching paths between them. Along each touching path, candidate segmentation points are located. The combination of candidate segmentation points create the candidate segments.

In our approach, we employ the projection profile and stroke-width extraction in the first stage. Then, we adopt a graph-based approach after thinning in the second stage. Specifically, we propose a new method of bridge separation and revise Voronoi diagrams for our purpose. In the bridge separation, we draw vertical lines explicitly at the estimated centers of the character patterns in the thinned image of an oversized horizontal segment, and seek to separate the touching points. Japanese characters often include some unconnected components. The vertical lines prevent vertically unconnected components from being left behind in the bridge finding. We produce multiple candidates of separation by removing the touching points combinatorially. Character boundaries thus located are reflected to the original image of the text line. For vertical writing, we can draw horizontal lines in the same way. Here, however, we assume horizontal text lines.

The proposed method is to segment not only single touches but also multiple touches between characters or radicals (components of characters), but this may split single characters into pieces. This problem, however, is solved by the best-path search for text recognition, using the model that integrates the linguistic and the geometric contexts with the recognition results [19]. We call the touching characters and radicals as touching pairs collectively.

Fig. 2 Single- and multiple-touching characters underlined by single lines and double lines, respectively.

Preliminary results of this research were reported in a conference paper [20], but since then we have added Support Vector Machine (SVM) to classify hypothetical segments [21] added more geometric features, revised their contributions to an over-segmentation recognition model and improved performance, which is reflected in Sect. 6.

The rest of the paper is organized as follows. Section 2 describes the flow of the proposed method. Section 3 pro-
poses the character segmentation method, which copes with single and multiple touches. Section 4 shows a recognition model. Section 5 summarizes data collection. Experiments and some discussions are presented in Sect. 6 and the conclusion is drawn in Sect. 7.

2. Flow of Handwritten Text Recognition

Figure 3 shows the flow of the handwritten text recognition system to which the proposed character segmentation method is incorporated. Here, we only state that the text-line segmentation uses a morphological method, and zone projection with the state-of-the-art performance.

3. Character Segmentation

This section describes the process to separate characters in each text line. Character segmentation is made in two steps: coarse segmentation and fine segmentation. The coarse stage is for segmenting non-touching and single-touching pairs and the fine stage is for multiple-touching pairs. In fact, single-touching pairs can be separated in the fine stage, but the coarse segmentation gives better time efficiency than the fine stage alone without degrading segmentation and character recognition rates.

3.1 Coarse Segmentation

Because our OCR is robust enough to recognize character patterns that are missing some strokes partially or contain some noise, it is unnecessary to separate touching pairs at the exact touching positions. This stage reduces the execution time of the fine segmentation stage that comes next. The steps of coarse segmentation are as follows:

(i) Estimation of the average width (AveWid) of characters: the average height of text lines ($H_{tl}$) is estimated from the extracted text lines and AveWid is estimated from $H_{tl}$ according to AveWid = $\delta \times H_{tl}$. Because of touching, estimation of AveWid from unconnected components is not stable. Therefore, we estimate it from the heights of text lines. The coefficient $\delta$ is determined from sample text lines.

(ii) Estimation of the average stroke width (AW): Several methods have been proposed to extract stroke width [22], [23]. We simply employ the Stroke Width Transform (SWT) algorithm to calculate the average width $AW$ of all the strokes in a text page.

(iii) Smearing: the closing morphology is applied with the element matrix that has the row number larger than the column number for each text line. This step is to enhance characters vertically, but does not affect the horizontal direction (touching direction).

(iv) Zero point segmentation: a vertical projection profile is computed for each text line and the text line is separated at points with the projection value of zero.

(v) Segmentation by AW: oversized segments are defined as their widths being larger than AveWid. Each oversized segment is further divided at the points where the projection values are less than $AW$. If the separation creates very small components, however, the segment is not separated.

This coarse segmentation separates non-touching pairs from each other and single-touching pairs effectively. Without stroke width estimation, it is difficult to know precisely the common stroke width of characters to set up the AW threshold, because the common stroke width is dependent on each text page. The coarse segmentation by the projection profile with $AW$ is shown in Fig. 4 and Fig. 5. Figure 4 shows handwritten text with a thick felt pen, and Fig. 5 shows another text written with a thin felt pen. In the first example, as shown in Fig. 4, vertical projection with the $AW$ threshold is shown in (b) and the coarse character segmentation is shown in (c). Red solid vertical lines show segmentation boundaries and the top-left number in each segment shows its index. Zero-point segmentation works between Segment 1 and Segment 2 (hereafter we denote $S_n$ instead of Segment n), between $S_2$ and $S_3$ and between $S_6$ and $S_7$ as shown in (c). Then the oversized segment including $S_3$, $S_4$, $S_5$, and $S_6$ is further divided into smaller segments $S_3$, $S_4$, $S_5$, and $S_6$ with the $AW$ threshold. $S_4$ is not separated into two smaller segments because the segmentation boundary shown in the red dotted line creates a very small com-

![Fig. 3 Flow in the proposed method.](image)

![Fig. 4 Coarse segmentation result (Ex.1).](image)
ponent. This is similar to the segment $S_7$. Figure 5 shows the second example in which some characters shown in blue boxes are touched while characters shown in red boxes have unconnected components. The result of the coarse segmentation is shown in Fig. 5 (c) where zero point segmentation does not work but segmentation by $AW$ produces $S_1, S_2, S_3,$ and $S_4$.

3.2 Fine Segmentation

For oversized segments not separated by the coarse segmentation such as $S_4, S_7$ in Fig. 4 (c) and $S_2$ in Fig. 5 (c), the fine segmentation is applied as follows:

(i) Thin each oversized segment: each oversized segment is thinned to its skeleton by the algorithm [24]. This step is necessary for steps (iii), (iv) and (v) since these steps work on skeleton images.

(ii) Draw vertical lines: from the vertical projection profile of each oversized segment, local maximal peaks are sought as shown in Fig. 5 (b). The number of local maximal peaks depends on the ratio between the segment width and $\text{AveWid}$. Then, a vertical black line ($VL$) is drawn at each local peak position to the bit map of the skeleton image produced from the oversized segment. As shown by red rectangles in Fig. 5 (a), Japanese characters often include some unconnected components. $VLs$ are to connect vertically unconnected components and make the bridge finding in the next step work even between segments including unconnected components. If bridges exist, they can be always found between $VLs$.

(iii) Seek and separate the shortest bridges between every $VL$ pairs: this step splits even multiple-touching pairs into untouched components. It is formed by the iteration of the following steps:

(iii-1) Find the shortest bridge: between every pair of neighboring $VLs$, a shortest bridge is sought, which is the shortest path from the center of the one $VL$ to that of the other $VL$. The algorithm to find the shortest path is $A^*$, the greedy searching algorithm, which achieves better time performance than traditional algorithms like Dijkstra.

(iii-2) Find and remove fork points: we first define fork points as shown in Fig. 6. A point is a fork point if it has at least three non-zero neighbors and each non-zero neighbor is not adjacent to others. After finding the shortest bridge, fork points are located on its skeleton along the bridge and removed (turned to a white pixel). Here, we exclude the fork points on $VLs$ since $VLs$ are considered as the centers of the touching objects.

(iv) Separate by Voronoi diagrams: all segments with the width more than $\text{AveWid}$ are separated by Voronoi diagrams [25]. Separation by Voronoi is performed on the skeleton image. Voronoi diagrams show natural borders called Voronoi edges between untouched components as shown in Fig. 7.

When the Voronoi diagram method is applied, fork points on a bridge are considered exclusively. Assume that between a pair of $VLs$, $n$ fork points are located ($f_{i1}, f_{i2}, \ldots, f_{in}$) on the first bridge; $m$ fork points ($f_{j1}, f_{j2}, \ldots, f_{jm}$) are located on the second bridge; $p$ fork points ($f_{k1}, f_{k2}, \ldots, f_{kp}$) are located on the third bridge and so on. Every fork point on the first bridge is combined with every fork point on the second bridge and so on. The Voronoi diagram method considers every combination, e.g., $(f_{i1}, f_{j1}, f_{k1})$, $(f_{i2}, f_{j2}, f_{k2})$, and so on. When a fork point on a bridge is being considered, no other fork point of the same bridge is considered at the same time. Each combination creates a segmentation candidate to be considered in the recognition model, which will be presented in Sect. 4.

Two improvements are also performed for the Voronoi diagram method as shown in Fig. 7, where the number in each component shows its index. $C_n$ denotes the component $n$.

1) Discard the edges splitting the upper component and the lower component vertically such as edges
among $C_1$, $C_2$ and $C_3$ and those between $C_4$ and $C_5$.

2) Remove Voronoi edges between a small component and a very large component such as the borders between $C_6$ and $C_7$.

(v) Reflect the segmentation boundaries to the original image: the segmentation boundaries obtained from its skeleton image by the fine segmentation are applied to the original image. The boundaries by Voronoi on the skeleton image are also the component boundaries in the original image.

Figure 8 enlarges the oversized segment $S_2$ in Fig. 5 (c) and shows an example to find and separate bridges between touching characters. By AveWid, we can estimate that the oversized segment is composed of four touching characters. VLs at peak positions are drawn and then bridges between two every neighboring VLs are found. Between VL1 and VL2, the shortest bridge is found and two fork points are located. The bridge is separated at the two removed fork points. Between VL2 and VL3, there is no bridge. Between VL3 and VL4, the shortest bridge is found and three fork points are located. The bridge is separated at the three removed fork points. This is the first round. In the second round, between VL1 and VL2, the second shortest bridge is found and a fork point is located. The bridge is separated at the removed fork point. Between VL3 and VL4, the second shortest bridge is found and a fork point is located. The bridge is separated at the removed fork point. This is the second round. In the third round, no bridge is found between any VL pairs, so that the oversized segment is separated by Voronoi. Fork points on a bridge are considered exclusively. We have $2 \times 1 = 2$ combinations for the first touching pair, and $3 \times 1 = 3$ combinations for the second touching pair. Then all of the segmentation possibilities are considered by the recognition model described in Sect. 4.

Figure 9 shows the result after applying the fine segmentation. Oversized segments are first separated on skeleton images as shown in Fig. 9 (a) and then reflected to their original images. The segment $S_5$ in Fig. 9 (c) is not separated into upper and lower components (which would be in error) due to the improvement 1) for Voronoi diagrams. Figure 9 (d) shows the erroneous segmentation candidates at fork points (between $S_2$ and $S_3$, and between $S_4$ and $S_5$), which are not the touching points. This erroneous separation, however, is not selected in the best-path search for text recognition.

4. Recognition Model

After the coarse segmentation and the fine segmentation, all the segmentation candidates are classified by an SVM model into three groups: $S$ (segmentation), NS (non-segmentation), and $U$ (undecided) using seven geometric features as shown in Table 1. The SVM model is trained so that $S$ must include true segmentation boundaries but may include false segmentation boundaries, while NS must exclude true boundaries as much as possible. $U$ is treated as either $S$ or NS. $U$ provides robustness for recognition but slows recognition speed. NS is removed and not considered any more. The NS deletion reduces the execution time as well as increases the recognition rate. A segment bounds by two neighbor $S$ or $U$ boundaries forms a candidate character pattern. Moreover, a sequence of consecutive segments delimited by $U$ may be combined to form a candidate character pattern. Concatenation of segments is limited by their total lengths. Each candidate character pattern is recognized by the OCR into candidate classes. The combination of all candidate character patterns and candidate classes of each character pattern is represented by a segmentation and recognition candidate lattice as shown in Fig. 10.
We employ our latest OCR to recognize a candidate character pattern \([5]\). Trained with the training set in Nakayosi database [26], this OCR can achieve the recognition rate of 97.64\% for the testing set of this database and 96.26\% for character patterns in the JEITA-HP databases [27]. It gives several candidate classes (say 30 candidates) for each candidate character pattern with confidence scores \(f_{ocr}\).

For each candidate character pattern \(P_k\) as shown in Fig. 10, the distance \(d_k\) by SVM for the boundary between it and its preceding segment is converted into a confident score by the sigmoidal function as shown in Eq. (1). We assume \(f_{sk}\) and \(f_{nk}\) are the segmentation score and the non-segmentation score for the candidate character pattern \(P_k\).

\[
\begin{align*}
  f_{sk} &= \ln \frac{1}{1 + e^{-\lambda_1 d_k}} \\
  f_{nk} &= \ln \frac{1}{1 + e^{-\lambda_2 d_k}}
\end{align*}
\]

We also employ single-character geometric features and between-character features as shown in Table 2 and Table 3 to get the recognition result of the whole text line. These features are normalized with \(H_{it}\) (divided by \(H_{it}\)). Figure 11 depicts these features.

To calculate the gap features \((G_1\sim G_6)\), a character pattern is split into three stripes along vertical and horizontal axes and projection profile on each stripe is computed. Each gap feature is the width of zero projection in each projection profile.

To reduce the memory size for the binary feature \(B_{1,2}\), character classes are clustered into six super classes: Kanji, Hiragana, Katakana, alphabet, digit, and symbol. The mean vectors, eigenvectors and eigenvalues are calculated for each super class.

In Fig. 10, each path of the diagram is evaluated by combining the scores of the candidate classes with their scores in the linguistic context \(f_{lc}\) and the scores of geometric features \(f_{G}, f_{BB}, f_{U}, f_{B}\). From the geometric features shown in Table 2 and Table 3, we transform them to geometric scores using quadratic discriminant functions \(QDF)\[28\]. For the linguistic context, we employ tri-gram. The total evaluation function for each path \(C_i\) is defined as follows:

\[
f(C_i) = \sum_{k=1}^{N} (f_{sk} + f_{nk} + f_{ocrk} + \lambda_3 \times f_{Bk} + \lambda_4 f_{BBk} + \lambda_5 f_{Uk} + \lambda_6 f_{Gk} + \lambda_7 \times f_{Uk})
\]

where the weights \(\lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7\) and also \(\lambda_1, \lambda_2\) in Eq. (2) are optimized using the Genetic algorithm, and \(N\) is the number of segmentation candidates. Finally, the Viterbi algorithm is used to find the best path [29], which will become
5. Data Collection

To evaluate our method, we prepared a database of Japanese handwritten text pages because no public database is available. The database stores format-free and vocabulary-free handwritten Japanese text patterns. The database has been constructed by the contribution of 25 Japanese people (two left-handed people, one woman) and made up of 45 pages (426 text lines and 11,550 characters). They were requested to write text according to Aozora Bunko (book collection), which publishes free copyright works of great Japanese literature. After scanning handwritten text pages, we annotated the touching information of touching pairs, which includes codes of characters, exact touching coordinates and acceptable touching coordinates to evaluate segmentation. The acceptable touching coordinates as shown in Fig. 12 denote a range within which segmentation could be made without affecting the character recognition result of both the sides. State-of-the-art OCRs are robust to deformed character patterns missing some parts or sticking some noises so that segmentation could be allowed if it is made within acceptable touching coordinates.

Overall, we have 1,871 touching pairs in 11,550 characters (1,552 single touching and 319 multiple touching pairs). We call this database OffTouchDB.

6. Experiments and Discussion

This section presents experiments to evaluate character segmentation before being combined with character recognition, and segmentation combined with character recognition.

6.1 Segmentation Performance

We evaluated segmentation performance after the coarse and fine segmentations on OffTouchDB. After those two steps, we obtained a total of 2,194 segmentation boundaries for single-touching pairs and 1,928 boundaries for multiple-touching pairs. We measured the performance of segmentation with the recall \( R \) and precision rates \( P \) as well as the harmonic average (F-Measure) defined as Eq. (3), Eq. (4) and Eq. (5).

\[
R = \frac{N_c}{N_G} \times 100 \quad (3)
\]
\[
P = \frac{N_c}{N_D} \times 100 \quad (4)
\]

\[
F = \frac{2 \times R \times P}{R + P} \times 100 \quad (5)
\]

where \( N_e \) is the number of correct segmentation boundaries. A segmentation boundary is correct if it crosses the exact touching positions or acceptable touching positions. \( N_G \) is the number of ground truth segmentation boundaries and \( N_D \) is the number of detected boundaries by the coarse and fine segmentations. Table 4 shows the final segmentation performance. Although the precision rates are low, especially for multiple-touching pairs, the recall rates are high for both single- and multiple-touching pairs, which is most important for the over-segmentation approach.

6.2 Applying SVM for Classifying Segmentation Boundaries

A) Training SVM for \( S \) and \( NS \) classification

We separate the Kondate database \([30]\) into two subsets: a training set of text pages from 75 people and a validating set from 25 people. From the ground-truth, each character pattern is associated with its bounding box as shown in a blue rectangular box in Fig. 13. By the character segmentation, segmentation boundaries are obtained. A segmentation boundary is classified into \( NS \) if it cuts the bounding box of a single character pattern, or classified into \( S \) if it is between bounding boxes of two character patterns. In the training set, the number of character segmentation boundaries in the \( NS \) class is 8,149 and that in the \( S \) class is 96,359 (ratio 1:12). In the validating set, the former is 2,544 for \( NS \) and the latter is 21,512 for \( S \). We trained the SVM model with the training set. Because of the unbalance between the number of the \( S \) boundaries and that of the \( NS \) boundaries, we employed different weighting values to balance them when we built the SVM model. The kernel function to build the SVM model is the radial basis function. The parameter gamma in the radial basis function is 0.4. Table 5 shows the classification result of character segmentation on the validating set by the SVM model built with different weighting values. In this experiment, we used the hard decision, i.e., classification result was either \( S \) or \( NS \).

As shown in Table 5, the SVM model trained with the weight value of around 3.0 offers the highest classification performance.

![Fig. 12](image-url) Exact touching coordinates (red) and acceptable touching coordinates (white).

<table>
<thead>
<tr>
<th></th>
<th>( R ) (%)</th>
<th>( P ) (%)</th>
<th>( F ) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Singly (ST)</td>
<td>92.65</td>
<td>65.54</td>
<td>76.77</td>
</tr>
<tr>
<td>Multiply (MT)</td>
<td>94.36</td>
<td>15.61</td>
<td>26.79</td>
</tr>
</tbody>
</table>

![Fig. 13](image-url) \( S \) and \( NS \) segmentation boundaries.
Table 5 Character segmentation by SVM with several weighting values.

<table>
<thead>
<tr>
<th>Weighting value</th>
<th>NS classified to NS correctly (%)</th>
<th>S classified to S correctly (%)</th>
<th>Total correct rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>7.43</td>
<td>98.35</td>
<td>88.73</td>
</tr>
<tr>
<td>2.00</td>
<td>32.90</td>
<td>97.20</td>
<td>90.40</td>
</tr>
<tr>
<td>2.75</td>
<td>40.63</td>
<td>97.05</td>
<td>91.08</td>
</tr>
<tr>
<td>2.85</td>
<td>48.32</td>
<td>97.01</td>
<td>91.86</td>
</tr>
<tr>
<td>3.00</td>
<td>53.14</td>
<td>96.98</td>
<td>92.34</td>
</tr>
<tr>
<td>3.15</td>
<td>53.97</td>
<td>97.05</td>
<td>92.49</td>
</tr>
<tr>
<td>3.25</td>
<td>58.02</td>
<td>94.91</td>
<td>91.01</td>
</tr>
<tr>
<td>5.00</td>
<td>65.37</td>
<td>90.11</td>
<td>87.49</td>
</tr>
<tr>
<td>7.00</td>
<td>74.08</td>
<td>83.26</td>
<td>82.29</td>
</tr>
</tbody>
</table>

Fig. 14 Determining $\beta_s$ not to classify NS incorrectly to S when the weight = 3.15.

Table 6 Classification by SVM.

<table>
<thead>
<tr>
<th>Threshold $\beta_s/\beta_{ns}$</th>
<th>Touching type</th>
<th>Seg. classified to S class (%)</th>
<th>Seg. classified to NS class (%)</th>
<th>Seg. classified to U class (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.4/1.8</td>
<td>ST</td>
<td>10.35</td>
<td>82.45</td>
<td>7.20</td>
</tr>
<tr>
<td></td>
<td>MT</td>
<td>18.83</td>
<td>58.28</td>
<td>21.89</td>
</tr>
</tbody>
</table>

ST: single-touching, MT: multiple-touching.

6.3 Character segmentation combined with recognition

A linguistic dictionary of tri-grams is built based on the year 1993 volume of ASAHI newspaper and the year 2,000 volume of NIKKEI newspaper. The single character geometric features scoring function is trained on the Nakayosi database [26]. For optimizing weight parameters ($\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7$) and between-character geometric features scoring function, we used patterns from 25 people in the Kondate database like the validating set used for...
determining the weighting value for the SVM model. We obtained the optimized weight parameter set ($\lambda_1 = 11.430$, $\lambda_2 = -16.710$, $\lambda_3 = 0.440$, $\lambda_4 = -0.330$, $\lambda_5 = 42.720$, $\lambda_6 = 3.520$, $\lambda_7 = -11.791$) after 150 rounds running GA. The coefficient for character width estimation $\delta$ is determined as 0.72 statistically again from the training set of 75 people in Kondate.

Table 7 presents the segmentation rates and recognition rates with a series of improvements. The segmentation rate here is calculated again by Eq. (3), however, $N_c$ is the number of correct segmentation boundaries after classified by the SVM model and chosen in the lattice. The over-segmentation by SVM improves the recognition rate significantly for multiple-touching pairs because many unpromising paths are discarded, which results in considering fewer promising paths in the lattice. The final segmentation rate is 85.95% for single-touching pairs and 81.19% for multiple-touching pairs when segmentation positions are within acceptable or exact coordinates. The final character recognition rate is 81.02% for single-touching pairs and 75.86% for multiple-touching pairs. The coarse segmentation without stroke width estimation segments neither single- nor multiple-touching characters. The coarse segmentation with stroke width estimation segments single-touching characters. The coarse segmentation with the bridge separation and Voronoi diagrams is effective for segmenting both single-touching and multiple-touching pairs. The fine segmentation with the bridge separation and Voronoi diagrams significantly increases the recognition rate for multiple-touching pairs, but there is little improvement for single-touching pairs. The fine segmentation with the bridge separation and Voronoi diagrams is effective for segmenting both single-touching and multiple-touching pairs. On the other hand, the two improvements of the Voronoi diagram method significantly increase the recognition rate for multiple-touching pairs, though they decrease the segmentation rate slightly. This is because the two improvements remove spurious boundaries.

The final remark is on the execution time. Table 8 shows the time by the best method with and without SVM in Table 7 for recognizing 45 text pages in GtTouchDB on an Optiplex 990 Dell PC with 8 GB RAM and Core i7-2600 3.4 GHz CPU. The execution time is largely reduced when SVM is applied.

### Table 7

<table>
<thead>
<tr>
<th>Seg. method</th>
<th>Seg. rate for ST (w SVM, w.o. SVM) (%)</th>
<th>Seg. rate for MT (w SVM, w.o. SVM) (%)</th>
<th>Char. rec. rate for ST (w SVM, w.o. SVM) (%)</th>
<th>Char. rec. rate for MT (w SVM, w.o. SVM) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse seg. w.o. AW</td>
<td>0.00</td>
<td>0.00</td>
<td>0.97</td>
<td>0.61</td>
</tr>
<tr>
<td>Coarse seg. w. AW</td>
<td>75.26</td>
<td>71.97</td>
<td>0.94</td>
<td>0.63</td>
</tr>
<tr>
<td>Fine seg. (Bridge sep. + Voronoi)</td>
<td>86.73</td>
<td>85.89</td>
<td>81.50</td>
<td>82.76</td>
</tr>
<tr>
<td>Fine seg. (Bridge sep. + Voronoi + 2 imp.)</td>
<td>85.95</td>
<td>86.63</td>
<td>81.19</td>
<td>80.86</td>
</tr>
</tbody>
</table>

ST: single touching, MT: multiple touching.

### Table 8

<table>
<thead>
<tr>
<th>Total Execution time (second)</th>
<th>Time per page (second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without SVM</td>
<td>4.752</td>
</tr>
<tr>
<td>With SVM</td>
<td>331</td>
</tr>
</tbody>
</table>

### 6.4 Discussion

In our method, we used SVM to prune improbable segmentation boundaries. If the evaluation function were ideal, pruning by SVM for segmentation boundaries would only decrease the recognition time but degrade the recognition rate because the correct path might be pruned. Since it is difficult to obtain the ideal evaluation function, however, wrong segmentation and recognition paths, producing better scores than the correct path, are pruned by SVM. An example is shown in Fig. 16 where the correct answer is returned by applying SVM. Another solution is to reflect the scores by the SVM pruning in the evaluation function, then wrong paths would be evaluated less than the correct path. The employed SVM pruning, however, not only brings about the similar effect for correct recognition but also decreases the recognition time, which is very important for practical OCR.

Figure 17 shows some examples of segmentation. The two text lines shown in (a.1) and (b.1) include many single- and multiple-touching pairs, but the system can segment and recognize them. In the first text line, the components 10 and 11 are a single-touching pair, abbreviated as $p_{10,11}$, but they can be separated by the coarse segmentation. The multiple-touching pairs $p_{4,5}$, $p_{19,20}$ and $p_{20,21}$ are segmented by the fine segmentation. In the second text line, the coarse segmentation with stroke width estimation
is effective for single-touching pairs \( p(11, 12), p(12, 13) \) and \( p(13, 14) \) while the fine segmentation works for a single-touching pair \( p(7, 8) \) and a multiple-touching pair \( p(14, 15) \). On the other hand, Fig. 17 (c.1) shows an example where segmentation failed at the pairs \( p(1, 2); p(4, 5) \) and \( p(8, 9) \). Although the pairs \( p(1, 2) \) and \( p(8, 9) \) are recognized owing to the robustness of our OCR, the pair \( p(4, 5) \) and the component 6 are misrecognized because the evaluation function and the SVM for pruning improbable boundaries are not perfected.

7. Conclusion

In this paper, we have proposed a method to segment characters in handwritten Japanese text pages. The proposed method is made in two steps: coarse segmentation and fine segmentation. The coarse step works for non-touching and single-touching pairs while the fine step separates multiple-touching pairs. It applies a new bridge-finding algorithm to a skeleton image and the Voronoi diagram method with two additional improvements. We use SVM to discard improbable segmentation boundaries, then employ a recognition model, combining the linguistic context and geometric contexts to recognize over-segmented characters and consider all possible touching positions. We have shown how the series of additional improvements are effective for segmentation and recognition of single- and multiple-touching pairs.

Acknowledgments

This work is being partially supported by the Grant-in Aid for Scientific Research (S)-25220401. We thank Prof. B. Indurkhya for discussion and refinement of our method.

References


Kha Cong Nguyen was born on 8 December 1989. He received the B.S. degree in Computer Science from Hanoi University of Science and Technology in 2012. Since April 2015, he is a Master student in Department of Computer and Information Sciences at Tokyo University of Agriculture and Technology (TUAT). He is currently working on offline handwritten Japanese text recognition and historical document processing.

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

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