Pergamon

Pattern Recognition, Vol. 28, No. 1, pp. 67–80, 1995
Elsevier Science Ltd

0031-3203/95 $9.50 + .00

0031-3203(94)00068-9

MACHINE PRINTED CHARACTER SEGMENTATION—AN OVERVIEW

YI LU
Department of Electrical and Computer Engineering, The University of Michigan–Dearborn, Dearborn, MI 48128-1491, U.S.A.

(*Received* 2 *September* 1993; *in revised form* 9 *May* 1994; *received for publication* 1 *June* 1994)

Abstract—This paper is part I of a two-part review series. We present here an overview of the character segmentation techniques in machine-printed documents. So far, as to this point, in most Optical Character Recognition (OCR) systems, either commercial products or systems described in the published literature, recognition algorithms are developed on isolated characters. Character segmentation is all too often ignored in the research community, yet broken and touching characters are responsible for the majority of errors in the automatic reading of both machine-printed and handwritten text. We will cover techniques for segmenting uniformed or proportional fonts, broken and touching characters; techniques based on text image features and techniques based on recognition results.

Character recognition Character segmentation OCR Touching characters
Broken characters

1. INTRODUCTION

The objective of automatic document processing is to recognize text, graphics and pictures in digital images and extract the intended information as would a human. Textual and graphical are two categories of document processing dealing, respectively, with the text and the graphics components of a document image.[1–3] Textual processing includes[1]

- determining the skew (any tilt at which the document may have been scanned);
- finding columns, paragraphs, text lines, and words; and
- performing optical character recognition (OCR).

In most existing OCR systems, character recognition performs on individual characters. Character segmentation is a technique which partitions images of lines or words into individual characters. Character segmentation is all too often ignored in the research community, yet broken and touching characters are responsible for the majority errors in automatic reading of both machine- and hand-printed text.[4] Character segmentation is fundamental to character recognition approaches which rely on isolated characters. It is a critical step because incorrectly segmented characters are not likely to be correctly recognized. This paper presents a comprehensive overview of character segmentation techniques for machine-printed text images. The complexity of character segmentation in machine-printed text stems from the wide variety of fonts, rapidly expanding text styles, and image characteristics such as poor-quality printers, and including sparse dot matrix printers. The thinning and/or thresholding processes in the binarization step can cause more broken, touching and merged characters. Unfortunately, most OCR systems have binarization as a preprocessing step and the segmentation and recognition algorithms are developed upon binary images. Our discussion in this paper will be within the scope of binary images unless otherwise indicated.

The printed text can be categorized in order of increasing difficulty as follows:

(1) Uniformly spaced characters, namely, fixed-pitch fonts, in which each character occupies an invisible box that is the same width as the boxes for all the other characters in the face.
(2) Well-separated and unbroken characters in proportional spacing, in which characters occupy different amounts of horizontal space, depending on their shapes.
(3) Broken characters—single characters have more than one component.
(4) Touching characters—more than one character in a single connected component.
(5) Broken and touching characters.
(6) Broken and italic characters.
(7) Touching and italic characters.
(8) Script characters.

Figure 1 shows examples of text in different fonts. Text in italic fonts have often been processed for slant correction before character segmentation process.[2,5] Hence, we do not discuss any segmentation techniques that apply directly to text in italic fonts. The techniques examined in this paper are not applicable to script characters; the segmentation of script characters is hardly addressed in the literature. However, some of the techniques used for segmenting handwritten

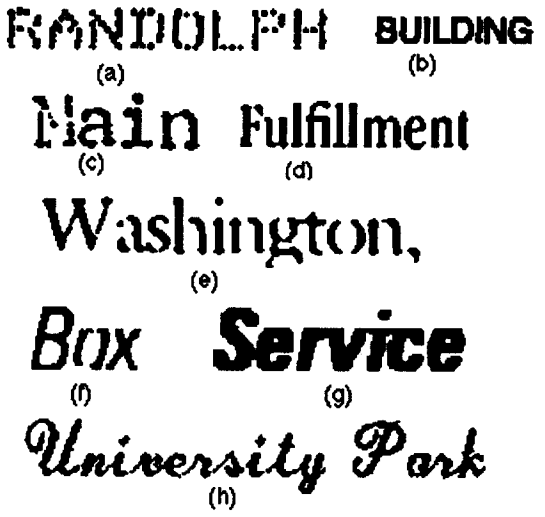


Fig. 1. Examples of text in different fonts. (a) Text in fixed pitch. (b) Text in proportional fonts. (c) Text containing broken characters. (d) Text containing touching characters. (e) Text containing broken and touching characters. (f) Broken characters in an italic font. (g) Touching characters in an italic font. (h) Text in a script font.

documents are also applicable to script characters. The techniques for handwritten character segmentation will be discussed in another paper.

This review is organized based on the intrinsic characteristics of the segmentation techniques being presented. The most common character segmentation algorithms are based on vertical projection, pitch estimation or character size, contour analysis, or segmentation–recognition coupled techniques. Before we discuss various segmentation techniques, we first present in Section 2 the common terms and measurements used in character segmentation. Section 3 presents algorithms for segmenting fixed pitch text. Sections 4, 5 and 6 present feature-based techniques for segmenting broken characters, kerned characters and touching characters, respectively. Section 7 presents algorithms that combine character segmentation with character recognition.

The evaluation of a segmentation technique is not easy as we know that there is no one universal definition of "correct" segmentation. In many systems, segmentation is considered as an integral factor in recognition and that recognition results determine the performance of segmentation algorithms. However, this measurement is constrained and influenced by the particular recognition algorithm employed. Even human observers can achieve different results in some difficult cases. Hence in this paper, we will not evaluate the performance of any segmentation algorithms.

2. COMMON TERMS AND MEASUREMENTS

The most common measurements of character features used in character segmentation are:

- character width: the number of columns a character occupies

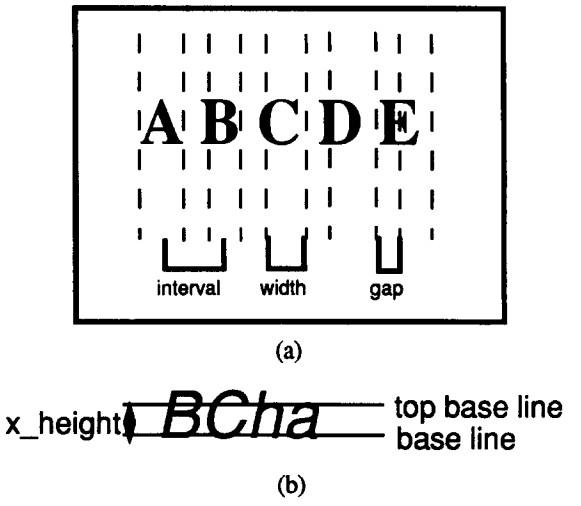


Fig. 2. Illustration of commonly used measurements (a) Character interval, width and gap. (b) Baseline, top baseline and x-height.

- character height: the number of rows a character occupies
- character gap: distance between two characters
- character interval: distance from the center of a character to the center of its successive character
- aspect ratio of a character: the character width divided by the character height
- occupancy ratio: the number of on-pixels in the area/total number of pixels in the area
- x-height of a line: the number of rows between the ascender and descender
- top baseline: the row where the ascender starts
- baseline: the row where the descender starts.

Figure 2 illustrates these measurements.

2.1. *Uniform spacing/fixed pitch*

Each character occupies an invisible box that is the same width as the boxes for all the other characters in the face.

2.2. *Proportional spacing*

Characters occupy differing amounts of horizontal space, depending on their shapes.

3. SEGMENTATION OF FIXED PITCH TEXT

Vertical projection is widely used in character segmentation. The vertical projection $V(x)$ is the histogram obtained by counting the number of black bits in each vertical scan at position x. In a single line vertical projection, a peak occurs for each vertical stroke of the character. If the characters are well separated, $V(x)$ should have zero values between characters.

Apparently, if a text is printed in a perfect condition, that is, the characters are well-separated and unbroken, character segmentation can be accomplished directly

from the vertical projection function on each text line. However, most digital text images are not in such perfect condition. Nevertheless, vertical projection is still served as an important feature in segmenting characters in general.

3.1. *Segmentation based on multiline projections*

Lu *et al.*[2] developed two multiline vertical projections based segmentation methods applicable to fixed pitch text.

In both algorithms, the first step is to group the lines of text according to their x-height. For example, the image in Fig. 3 has two groups, the first line forms one group, and the second and third lines form another group. Then the vertical projection is performed on each group of lines. The multiline vertical projection is obtained simply by adding on-pixels appearing at each column. If all the text lines in the same group are properly aligned, characters locating at the same column but in different lines have the same character boundary. It means that the intervals of zero values in the multiline vertical projection function correspond to the gaps between horizontally connected components in the binary image (see Fig. 4). In other words, the character gaps can be found through multiline vertical projection.

In most cases the broken characters in each individual line do not affect the multiline vertical projection. On the other hand, the noise between characters can mask character gaps and make segmentation impossible.

The first segmentation method is based on the statistics of character width and intervals of the regions separated by zero values in the multiline projection function. In order to obtain an accurate estimation, the statistics are counted only from the regions containing more than one line. For example, in Fig. 4, the statistics were computed to the end of the second-longest line.

A decision procedure is then invoked to determine if the multiline group will be accepted or sent to other segmentation procedures. The decision procedure checks the variance of character width, gap, and interval statistics to determine whether they are within the range of acceptance. If the statistics are within a certain range, a merging and splitting procedure is invoked to segment the single line sections. The merging process uses the interval, width, and gap statistics collected based on the vertical projection on the multiline sections to combine small neighboring regions. The splitting process splits any region that is greater than the maximum width computed from the vertical projection of multiline sections.

If the variances are outside the acceptable range, but, on the other hand, the group is deemed uniform, a merging or splitting process based on the estimated sum of the character width and gap is applied to the regions of the group vertical projection. If the statistics indicate the text is not uniformly spaced, a merging process based on estimated character width and gap is applied to the entire multiline projection in an attempt

Fig. 3. Two different fonts with the last two lines possessing uniform spacing.

Fig. 4. Character segmentation based on multiline vertical projection.

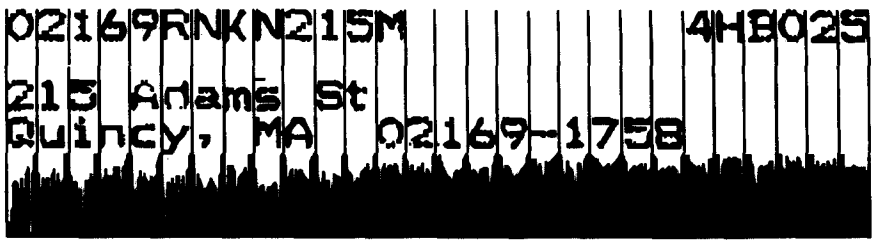


Fig. 5. GPD applied to a fixed-pitch dot matrix AB.

Fig. 6. GPD applied to a proportional pitch AB.

to group broken characters. The statistics are recomputed after the merging and splitting processes. The decision module is applied again. If the multiline group is accepted at this level then the segmentation is applied to all of the lines in the group, otherwise each line image in the group is sent to the single line process.

The other approach taken by Lu *et al.*[2] is the gap periodicity detection (GPD) algorithm. This method exploits the fact that gaps must appear at periodic

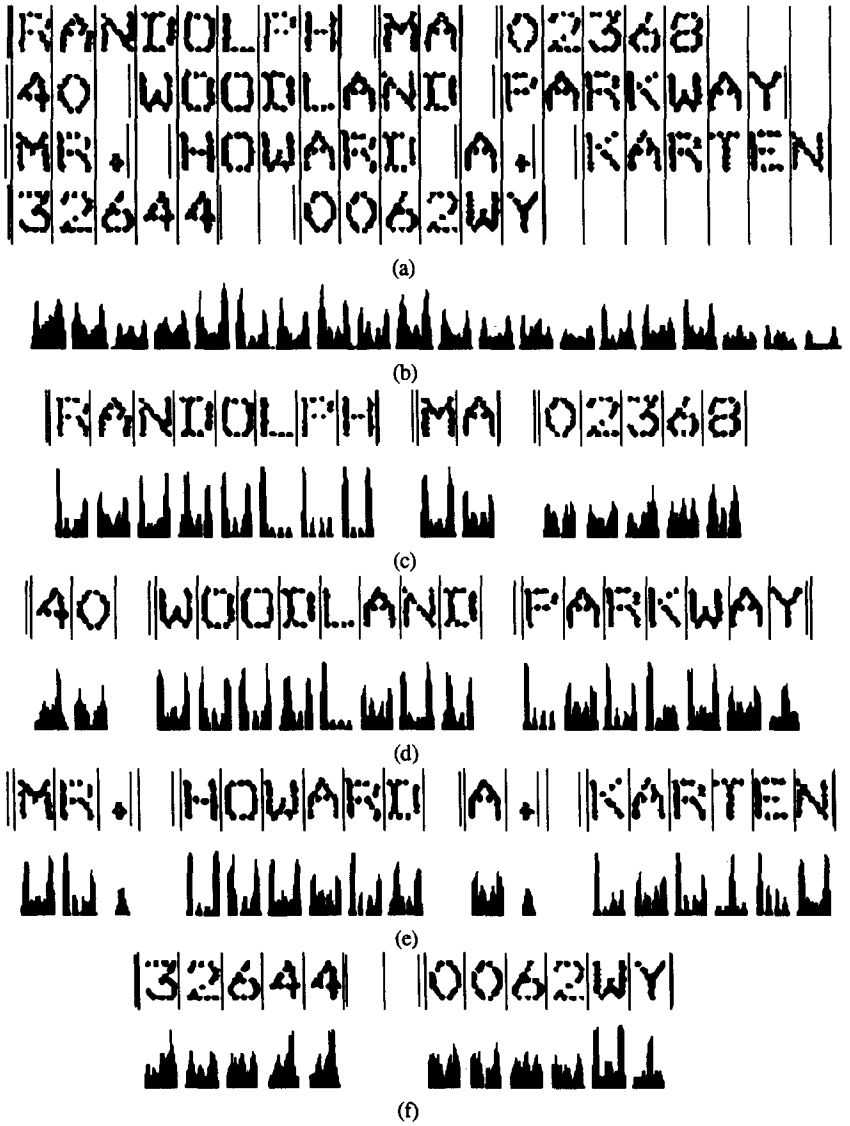


Fig. 7. Multiline segmentation of an image in dot matrix style.

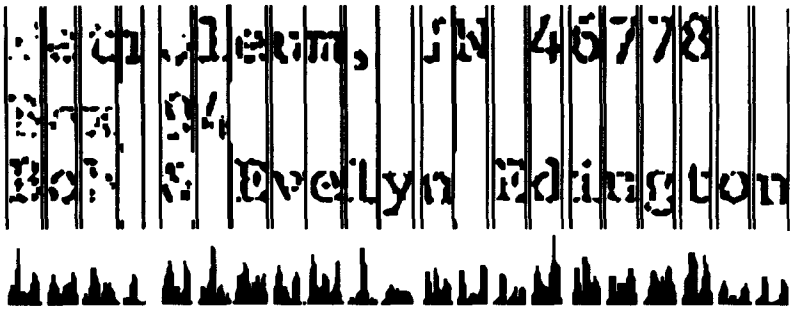


Fig. 8. Multiline segmentation of broken characters.

intervals in a fixed pitch font, but not in a proportionally pitched font. First, the approach uses the vertical projection of the multiline sections to determine the best single offset and pitch combination for the address block. The pitch estimation is computed based on the mean gap length and the mean run length of the inverted vertical projection. If the variance in the estimates for these two values is above an empirically determined threshold, or the estimated pitch is much greater than the mean line height, the text is not fixed-pitch and hence it should be processed by other procedures.

Based on the estimated pitch, the algorithm cycles through potential combinations of pitch and offset, which are evaluated by a measure of the "goodness" of the resulting cuts. The distance from the beginning of the inverted vertical projection to the first line is the current offset and the distance between each line is the current pitch. Figure 5 illustrates a dot matrix image successfully segmented by the GPD algorithm. Figure 6 shows an image that contains proportionally pitched text. The GPD detects that the text is not fixed pitch and so passes this image on to other methods for character segmentation.

The motivation of the multiline process is to use the cumulative projection to overcome errors or uncertainties in single line projection. Both Figs 7 and 8 are examples that multiline segmentation can overcome the difficulties encountered at the single line projection. The image in Fig. 7 is in dot matrix style, and the image in Fig. 8 contains broken characters. The single line projection showed gaps within characters, but the multiline projection showed only the gaps between characters. In Fig. 7, (a) and (b) show the result image by the multiline process and the multiline vertical projection, respectively, (c)–(f) show the individual line images and their corresponding vertical projection.

The two methods presented above were designed to segment fixed-pitch text and are especially effective for broken characters and characters in dot matrix style. They were motivated by the inherent property of fixed-pitch fonts that gaps must occur at fixed intervals in lines of text. Both methods are highly efficient since they operate on groups of line images.

4. SEGMENTATION OF BROKEN CHARACTERS

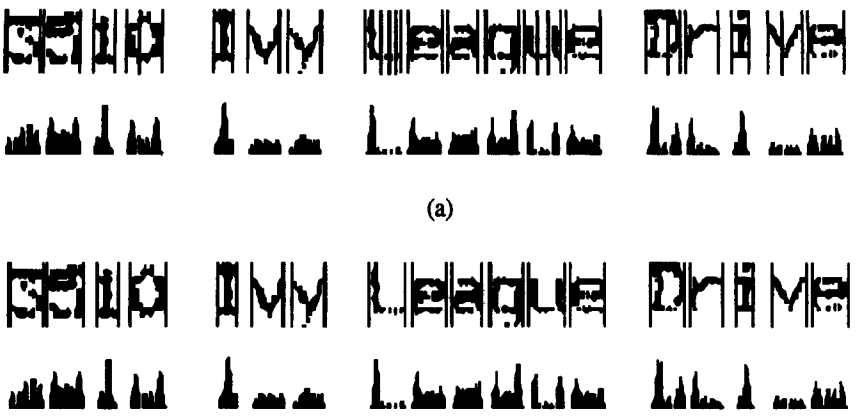
There are two types of methods for segmenting broken characters; one is to employ a merging process based on the estimated character width and intervals, and the other is to combine character components based on recognition results. This section will discuss the methods of the first type only. The recognition based segmentation approaches will be discussed in Section 7.

Lu *et al.*[2] proposed an algorithm for segmenting broken characters. It consists of an estimation procedure, a sequential merging procedure, a grouping procedure based on the estimated character width, and a decision procedure.

The algorithm first segments a text line into regions based on the vertical projection. Statistics are then calculated for the character intervals, widths, and gaps over these regions. An estimation procedure was designed to estimate whether a line image is in dot matrix style. The decision is based on the number of regions whose region and gap widths are smaller than a given threshold.

The sequential merging procedure was developed to segment line images in dot matrix style. It begins by combining neighboring regions with the smallest gap. Statistics are then recomputed and the next smallest gap is used to merge the neighboring regions. This process continues until the statistics are within an acceptable range.

If the line image is estimated to be in proportionally spaced fonts, then a merging procedure is invoked to merge small neighboring regions with small gaps. The merging procedure uses the estimated character width and gap to combine the neighboring regions. If the neighboring regions are small and the total width after being combined is less than a maximum threshold, the regions are candidates to be merged. The maximum threshold is computed dynamically. Figure 9 shows an example generated by this merging process. The characters "O", "L", "u", "D" and "v" were oversegmented using a single line vertical projection [see Fig. 9(a)], and then their components were combined by the merging procedure. As part of the merging process, a character analysis routine was incorporated



(a)

Fig. 9. Merging broken characters. (a) Broken characters were oversegmented based on a single line vertical projection. (b) Neighboring regions were merged to recover broken characters.

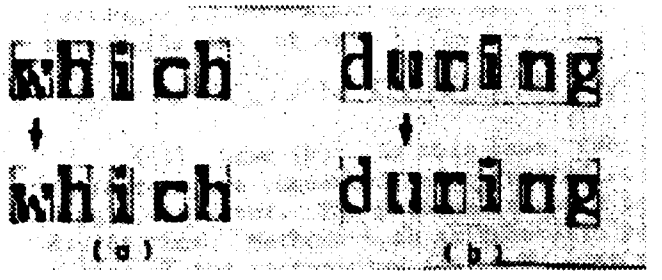


Fig. 10. Example of connected character block [adopted from reference (11)].

to avoid merging narrow characters, such as ll, ti, ri, il, ff. The routine analyzes neighboring regions to determine whether they represent two separate characters. The analysis is based on the relative heights of the regions and the occupancy ratio. If the conditions for separate characters are met the two regions are not joined.

After the merging process the statistics are recomputed and a decision module is invoked. The decision module checks the statistics of character width, gap, and interval to determine whether the segmentation results are within an acceptable range.

Merging broken characters based on estimated character width and/or aspect ratio works in the cases where most characters are well-formed. It is not a reliable approach in more complicated cases.

Nakamura *et al.*[6] and Okamoto *et al.*[7] proposed a character segmentation algorithm based on propagation and shrinking in the vertical and horizontal directions. A line image is first propagated three times with 4 pixels in the horizontal direction and 1 pixel in the vertical direction. After the propagation–shrinking process, the connected regions were computed. The advantage of computing connected regions based on the results from the propogation–shrinking process is that it does not separate components in the vertical direction. Figure 10 shows an example excerpted from reference (6). In the figure, the second portion of "W" has two separate components and the two components of character "i" are in one region. The next step is to extract word regions by another propagation–shrinking process, propagating and shrinking three times with 4 pixels in the horizontal direction and 1 pixel in the vertical direction. Then the mutual locations of the individual character regions in the word were examined successively. Let W_n be the nth word bounding box, and $W_{n,1}, W_{n,2}, \ldots, W_{n,z}$ be z individual regions contained in W_n. If

$$|W_{n,p} \oplus W_{n,p+1}| \leq T_q, \quad p = 1, 2, \ldots, z-1$$

is satisfied, then both $W_{n,p}$ and $W_{n,p+1}$ are two components of the same character, therefore they should be combined. In the formula above, $|W_{n,p} \oplus W_{n,p+1}|$ represents the width of the bounding box containing $W_{n,p}$ and $W_{n,p+1}$, and the threshold T_q was set to $T_q = 25$ in the algorithm. Let the combined region be $W_{n,p}$. Then region $W_{n,p}$ and $W_{n,p+2}$ were examined similarly. In Fig. 10, the broken "W" and "U" (see the two words on the top) were correctly segmented (see the two words at the bottom).

5. SEGMENTING KERNED CHARACTERS

Kerned characters are the characters who overlap with neighboring characters. The "TA" and "PA" in Fig. 11 are examples of kerned characters. Vertical projections, for example the vertical projection of "TA" and "PA" in Fig. 11, do not provide much information for either segmenting kerned characters, or separating punctuation marks from the characters when they are

Fig. 11. Structural analysis module used to segment "TA" and "PA".

Fig. 12. Detection of punctuation results in the separation of "P." and "W."

overlapped. Kerned characters are often segmented by using structural analysis.

Lu *et al.* proposed an algorithm based on the analysis of neighboring components.[2] The algorithm was aimed at segmenting kerned characters, characters in italic fonts ("*IN*"), and punctuation marks ("P"). However, it must also avoid separating components in broken characters and in multiple stroke characters such as "i" and "j" in reference (8).

The algorithm consists of two computational steps: compute the connected components and analyze the structure of neighboring components. The connected components were obtained by searching runs of "1"s and the connected components were allowed to have 2 pixel gaps in both the x- and y-directions. The structural analysis of neighboring components consists of two major procedures: kerned character analysis and punctuation detection. Kerned character analysis was designed to group broken or two-stroke characters and to separate kerned characters. In order to accomplish this, every two neighboring components are examined for vertical overlap and possible grouping. If the degree of overlap exceeds a threshold, the components are candidates to be joined. Figure 12 shows an example of kerned characters segmented by the structural analysis. Because punctuation marks are frequently overlapped by the preceding character, Lu *et al.*[2] developed a punctuation detection procedure. The punctuation detection procedure determines whether the second of the two neighboring components is a punctuation mark. The analysis is based on the occupancy ratio, size of components, relative locations, distance between the neighboring components, and locations of components with respect to the base and the top-baseline of the line image. If the second component is determined to be a punctuation mark, then it is not joined, as depicted in Fig. 12 (see "W." and "P."). The punctuation marks detected by this procedure are period, comma, colon and semi-colon.

The combination of these two procedures groups broken and two-stroke characters, but separates kerned characters and punctuation marks from the neighboring characters.

6. SEGMENTING TOUCHING CHARACTERS

Merged characters, sometimes referred to as composite or merged characters, are components of connected multiple characters. Segmentation of touching characters has been the most difficult problem in character segmentation. There are two key issues involved in this problem

- determine which segments contain multiple characters, and
- find break locations.

The techniques for segmenting merged characters can be divided into two categories, featured-based or recognition-based. Some components of touching characters can be segmented only by recognizing the individual components. On the other hand, some characters can be recognized only after they have been isolated. Therefore, techniques in both categories are equally important. Techniques in the first category often transfer the vertical projection to a function which provides more direct information for finding the break points. A decision process is then formed based on the function value, characteristics of the word or line images, such as width, height of the word and candidate segment, aspect ratio, foreground pixel density, contour analysis, etc.[7–10] The techniques in the second category often involves a segmentation-and-recognition process,[3,5,11] which will be discussed in the next section.

6.1. *Discriminating multiple character components*

The most commonly used feature for detecting multiple character segment is the width and the aspect ratio of the character.[6,7] In reference (12), Lu suggested that character width can be dynamically estimated during the segmentation process. Each candidate segment is then examined by comparing its width with the estimated character width or by measuring the aspect ratio of the segment. It is well understood that most characters have widths smaller than their height and a single character segment should have a width of less than twice of the estimated character width. The com-

Fig. 13. Examples of touching characters in proportional fonts.

bination of these two measurements works well on characters in uniform spacing and most characters in proportional fonts, but fail in some special cases. Figure 13(a) shows examples of touching character segments, "In", "rp", "ed", that are wider than single characters in the image, and so we can identify them by using either the segment width or the aspect ratio. However, in Fig. 13(b), the touching character segment "tt" is narrower than "J" and "w", and the touching character segment "LI" is no wider than "N" and "G" in the same string. Furthermore, the aspect ratios of "tt" and "LI" are as normal as a single character segment. Hence the character width and aspect ratio are not sufficient information for identifying multiple character segments, and different type of features are required for solving this problem.

Lu further proposed generating single and multiple character profile models for discriminating multi-character segments from single character segments.[12] The profiles of characters are constructed from the smoothed vertical projection contour. Figure 14 shows the profiles of some single and touching characters. A projection contour of multiple characters has at least two significant peaks. However, characters "H", "n", "N", "u" and "U" also have two significant peaks in their projection contour, and characters "m", "w", "M" and "N" have two or more significant peaks, depending on fonts. Since there are only a handful of such characters and the projection contours of such single characters are different from multiple characters, Lu suggested constructing profile models for the characters with two or more significant peaks in their projection contour. The features used in the model construction are number of peaks, height and spread of the peaks, width, depth and smoothness of the valleys, and the symmetry of the smoothed vertical projection curves. Lu's profile model based algorithm was designed to err on the side of not separating touching characters rather than incorrectly dividing a character into more than one segment. The most multiple character segments missed by the algorithms are the ambiguous character pairs that can easily be recognized as a single character.

Other people used classifiers to discriminate multi-character segments. Kahan and Pavlidis used a recognition algorithm to test multiple character segments.[8] If the probability computed by the classifier for a segment being a character is less than the *a priori* probability that the blob is that character, then they assume the segment is a merged character. Tsujimoto and Asada used recognition result to identify multi-character components.[5] If a component had a similarity measure greater than *a priori* threshold, it was a single character component; otherwise it was a multi-character component.

6.2. *Feature-based segmentation*

The key issue in the splitting process is to determine where to split the multi-character regions. Kahan and Pavlidis proposed a function, the segmenting objective function, for finding breaking points within merged characters.[8] The function is the ratio of the second difference of the vertical projection function $V(x)$ to $V(x+1)$, namely,

$$\frac{V(x-1)-2*V(x)+V(x+1)}{V(x)}.$$

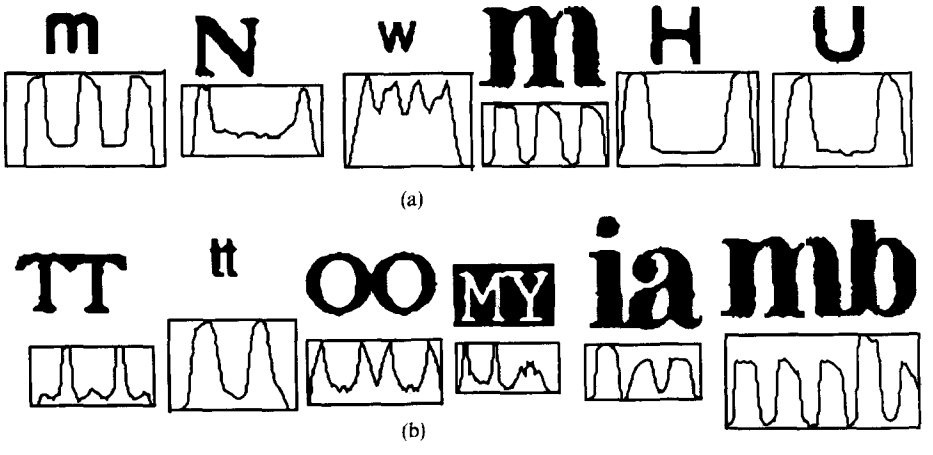


Fig. 14. Examples of projection contours for single and multiple characters.

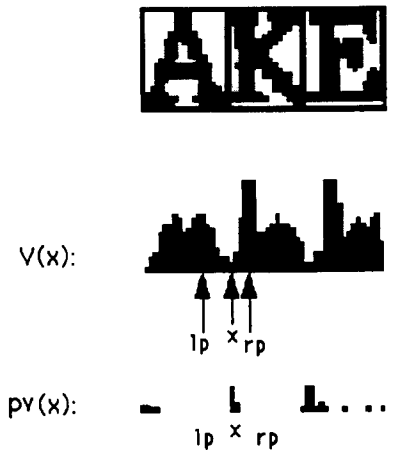


Fig. 15. The peak-to-valley function emphasizes local minima in the vertical projection.

The maxima of the segmenting objective function were used as the possible breakpoints.

Lu *et al.* proposed a generalized differential technique[2] for splitting touching characters. The vertical projection of the region is mapped on to a second projection, called the peak-to-valley ratio, by applying a generalized differential technique. The peak-to-valley function is illustrated in Fig. 15. Sharp minima in the vertical projection are represented as maxima in the peak-to-valley ratio (shown in Fig. 16). These maxima are then considered as potential break points between the merged characters. Splitting locations are determined from both the maxima in the peak-to-valley function and the estimated character width. The splitting process first determines an allowable splitting region based on the estimated character width. The position of the maximum of the peak-to-valley function within the splitting region is the splitting location. Then the widths of the two subregions separated by the splitting location are examined. If a subregion has an acceptable size, it is considered to be a character and accepted. If the width of a subregion is large enough to contain more than one character, the search for a splitting location in the subregion is performed again. If no maxima exist within the splitting regions, the process rejects the region. Figure 17 illustrates splitting locations found within splitting regions. Two splitting regions (SR) are located in the peak-to-valley function. S1 and S2 are the first and second splitting locations found within the two splitting regions.

Hoffman and McCullough developed two algorithms, quasi-topological and topological segmentation, for segmenting machine-printed characters using feature extraction techniques.[13]

The quasi-topological segmentation algorithm is a combination of stroke count and the contour measurement methods. The horizontal stroke count is obtained by counting different part of the character that a given vertical scan encounters. Figure 18 illustrates this concept. Hoffman and McCullough identified 11 classes of stroke-counting patterns for subclasses of the entire alphabet of upper-case alphanumeric characters. For example, {M, N, T, U, W, Y} exhibit a (12121) pattern, {D, V, 7} exhibit a (121) pattern, etc. Segmentation decisions were made by finding the closest match between the exhibited stroke-count pattern and a set of stored stroke-count reference patterns. The references were determined *a priori* and each was tuned to data from the particular type of font under consideration. During each scan, the stroke count and the number of scans for which that count has existed are computed

Fig. 16. Potential breakpoints derived from the peak-to-valley function.

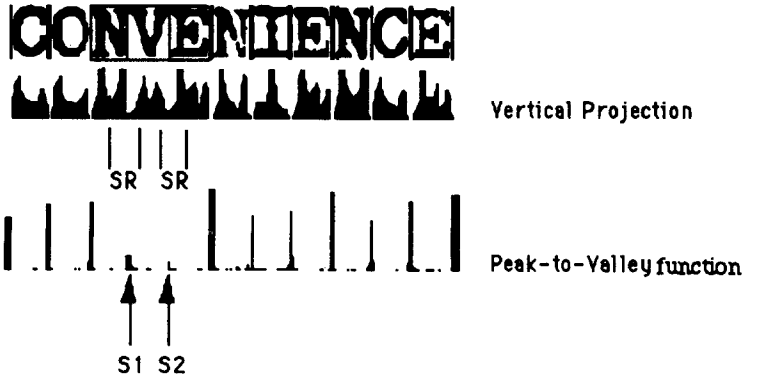


Fig. 17. Splitting regions (SR) and locations.

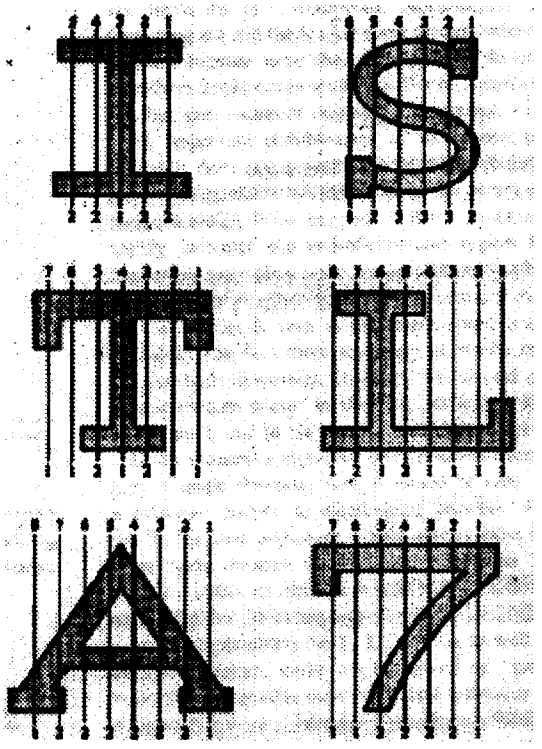


Fig. 18. Horizontal stroke patterns. The numbers above the character identify the scan number and those below, the stroke count [adopted from reference (5)].

and stored. A table of six items, the current, the previous and the second previous stroke count and the number of scans for which it existed, is matched to the stored references.

The topological segmentation algorithm is based on the detection and measurement of the leading and trailing edges of character strokes. A bit pattern is scanned from right to left and at each scan three parameters are computed, P the number of leading-edge bits, N the number of trailing-edge bits and L the number of black bits. A leading-edge bit is defined as a black bit with a horizontally adjacent white bit in the previous scan, and a trailing-edge bit as a white bit with a horizontally adjacent black bit in the previous scan. In segmenting touching and noisy characters, the parameters are combined in an equation of the form:

$$T_n = \sum_{i=0}^{n} U_i P_i - \sum_{i=0}^{n} V_i N_i + Y_n P_n - W_n L_n - X_n.$$

When $T_n \leq 0$, sectioning occurs at scan n. The design problem is to choose suitable definitions for P, L and N and appropriate values for the coefficients U, V, W, X and Y so that sectioning will occur in the desired region of the character. The coefficients U and Y should be constant for all scans and their specific values can be determined through experiments. The other parameters have the following form:

$$V_n = \gamma_v + \delta_v F_n + \varepsilon_v \bar{F}_n$$

$$W_n = \gamma_W + \varepsilon_W \bar{F}_n$$

$$X_n = \delta_x F_n, .. U_n = \gamma_U, \quad Y_n = \gamma_Y$$

where γ, δ and ε are positive constants determined experimentally, and

$$F_n = \begin{cases} 0 & (\mathrm{CFR})_n < \lambda \\ (\mathrm{CFR})_n - \lambda & (\mathrm{CFR})_n \geq \lambda' \end{cases} 0$$

$$\bar{F}_n = \begin{cases} \bar{F}_{n-1} + 1 & (\mathrm{CFR})_n \geq 1 \\ & \text{and} \quad (\mathrm{CFR})_n = (\mathrm{CFR})_{n-1} \\ \bar{F}_{n-1} & (\mathrm{CFR})_n = (\mathrm{CFR})_{n-1} + 1 \end{cases}$$

The Connective Feedback Request function, $(\mathrm{CFR})_n$, should be determined at the end of each scan by the following formula:

$$(\mathrm{CFR})_n = \begin{cases} (\mathrm{CFR})_{n-1} + 1 & G_n - \eta F_n > \theta \\ (\mathrm{CFR})_{n-1} & G_n - \eta F_n \leq \theta' \end{cases} \quad \text{and}$$

$$G_n = \sum_{i=0}^{n} (U_i P_i - V_i N_i),$$

where λ, η and θ are positive constants experimentally determined from the geometry of the characters to be segmented, and F_0, $\bar{F}_0$ and $(\mathrm{CFR})_0$ are set to zero.

The topological segmentation steps are as follows. First, the measurement coefficients are set to their initial values. The bit pattern is examined scan-by-scan. At each scan, the P, N, and L values are computed and the CFR function is updated. If the threshold function T_n indicates a sectioning, a segmenting procedure is executed, otherwise the next scan is examined. The segmenting procedure makes use of two algorithms applied in parallel, "density increase" and "terminating line element". The character is segmented if the decision criterion for either algorithm is satisfied. The bit-density function was computed as

$$\sum_{j=1}^{n} P'_j - N'_n > f,$$

where n is the current scan, P'_i is the count of black bits that horizontally adjacent to a white bit in scan $j-1$, and bits that adjacent to a black bit in scan $j-1$ and a white bit in scan $j-2$; N'_n is the count of white bits in scan n that are horizontally adjacent to a black bit in scan $n-1$.

Based on the analysis of experimental results, Hoffman and McCullough recommended that topological segmentation algorithm was a better choice over the quasi-topological algorithm.

Tsuji and Asai[10,14] and Tsuji *et al.*[15] proposed an adaptive character segmentation algorithm applicable to both fixed and variable pitch characters. The algorithm consists of two parts, character pitch estimation and character sectioning position determination. The character pitch estimation is performed through the best linear unbiased estimator P' which is derived by calculating candidate character pitch P that minimizes the linear square-error function $\varepsilon^2(p)$. The linear square-error function is calculated from the number of characters in each cluster $f(k)$, the number of samples and the sample mean of character cluster distance, which are determined from the vertical projection function. The accuracy of the estimated character pitch P' is

represented by minimum linear square-error and the linear expression for variance $V^2(f(k))$, in regard to the number of characters k. In the case of variable character pitch, more than one character pitch exists, and the linear expression for variance $V^2(f(k))$ becomes larger than in the fixed character pitch.

Based on the estimated character pitch, the character sectioning positions are determined by dynamic programming. The character sectioning positions are found by setting permissible stages in a character line image. The space in a character line is included in the permissible stage, which is determined by using the threshold $(1+\tau_x)*P'$, where τ_x is the weighting factor. The start stage is defined as the space contained in the range from the start of the character cluster and is determined based on the estimated character pitch P', from the previous character sectioning position to the beginning of the next-detected character cluster. The adjacent stages satisfy the following relation:

$$(1-\tau_x)*P' \le d_{i,j}(k,k+1) \le (1+\tau_x)*P'$$

where $d_{i,j}(k,k+1)$ is the distance between the two candidate positions $x(k,i)$ and $x(k+1,j)$, $(j=1,2,\ldots,h_{k+1})$, and $x(k,i)$ indicates the ith candidate position for character sectioning in the kth character sectioning stage. The end stage can be either detected beforehand, using threshold, or set by detecting the partial disturbance of a fixed character pitch line. The candidate positions are further evaluated by the criterion F

$$F(0,k+1;s,j) = (1-\tau_\varepsilon)*\sigma_d^2(0,k+1;s,j) + \tau_\varepsilon*(\mu_d(0,k+1;s,j)-P')^2$$

where $\mu_d(0,k+1;s,j)$ and $\sigma_d^2(0,k+1;s,j)$ are the mean and the variance for the $k+1$ distances from stage 0 to the stage $k+1$. The evaluation is performed through dynamic programming. The algorithm was tested on a set of mail pieces within which 65 were fixed pitch and other 50 were unspecified character pitch. The mail pieces were binarized and the lines were extracted. The binary line images were input to the algorithm. The experimental results showed that the algorithm reached an accuracy of 99.2%.

Nakamura *et al.*[6] and Okamoto *et al.*[7] proposed an algorithm for partitioning touching characters based on propagation and shrinking processes. If the width of a connected region after the propagation–shrinking process is within a certain range, then it contains two characters.

There are two possible situations:

- characters were well separated before the propagation–shrinking process, and
- characters were touching before the propagation–shrinking process.

Let W_n be the nth word bounding box, and $W_{n,1}$, $W_{n,2},\ldots,W_{n,z}$ be z individual regions contained in W_n. Let the width of the current region $W_{n,p}$ be W. In the first case, the region between $W/3$ and $2W/3$ was scanned to find a scanning line, and the two characters are partitioned at the scanning line. In the second case, the algorithm searches a specific range within the region $W_{n,p}$ to find the point of minimum run-length. The search range was H points away from the middle point of $W_{n,p}$, i.e.

$$H = \left[\frac{m-T}{2}\right],$$

where m is the shorter width of the two characters, T is the spacing between characters, and [ ] is the Gaussian symbol. If there exists more than one minimum run-length, the point closest to the middle is used. In implementation, T was set as the maximum spacing between the regions within the word W_n, and m was the minimum width of the regions within the word W_n.

7. RECOGNITION-BASED SEGMENTATION

Many people think that the existence of reliable features to distinguish boundaries in all fonts from interior regions is arguable,[4,16] open-loop approaches, segmentation-to-recognition, render errors irrecoverable. Therefore character segmentation should be closely coupled with character recognition. The recognition based segmentation techniques we discuss in this section are:

- sliding window method
- closed-loop segmentation and recognition and
- multiple hypothesis scheme.

Kovalevsky described a sliding method[17] as follows. A delimiting window moves along the line image. The portion of the line image falling in the window is sent to a recognizer. If it matches a prototype or, more generally, satisfies some condition for recognition, it is accepted for processing. However, Kovalevsky pointed out that errors can arise in practice because portions of two neighboring characters falling in the window simultaneously may closely imitate a third symbol. For instance, vertical lines belonging two neighboring characters with serifs above and below may form the letter "II", the neighboring characters "O" and "C" may form an image reminiscent of the character "X". Kovalevsky, therefore, recommended that we must recognize whole lines rather than individual characters.

Casey and Nagy proposed a closed-loop, segmentation-and-recognition algorithm that accepts a string of connected characters only if it can be decomposed into a set of recognized characters.[4] Their algorithm can be illustrated by Fig. 19.

The algorithm took a pattern array as an input. The Segmentation Supervisor (SS) initializes the classification window as the full width of the pattern array so that if the array contains a single character, the classifier can recognize it in one step. If the classifier rejects the pattern, then the viewing window is narrowed from the righthand side and the classifier is applied to the truncated array. This process is repeated until either the truncated array is successfully recognized, or the window becomes so narrow that the search is abandoned. If the pattern inside the window is recognized,

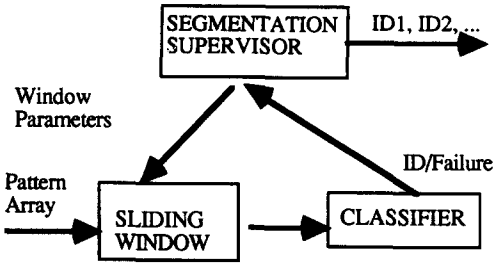


Fig. 19. Combined segmentation/classification [adopted from reference (3)].

the SS records the ID (identity) and the truncation point. The window is then reset with its left-hand margin at the truncation point, and its right-hand margin is once more set to the end of the input array. The combined windowing and classification process is repeated on this reduced array. The segmentation terminates successfully if the residual array after a positive classification is either null, or narrower than any character in the alphabet. In the case where no classification can be found before the window closes completely, if the failure is on the initial input, the entire array is rejected; otherwise, if a truncated array fails, the SS assumes the previous classification was in error. It therefore negates the doubtful classification and resets the truncation boundaries to the values that were recorded for that decision. Then it narrows the window from the right and attempts to reclassify the input. This algorithm terminates successfully only if the input array can be decomposed into a sequence of individual pattern arrays, each of which is classified as a valid character.

The segmentation algorithm has been included in an adaptive image entry system for encoding arbitrary scanned documents. The system was designed to accept arbitrary typefaces. The classification used in segmenting touching characters was to match the pattern array within a window against a library of prototypes. The prototypes were generated from the scanned data either automatically or interactively. The touching character array was generated by a preprocessing that partitions each text line into a sequence of pattern arrays using a rudimentary segmenter: character boundaries were specified at the places where there is sufficient white space between successive black regions.

During the classification process, Casey and Nagy used a decision network for preclassification. The decision network selects prototype candidates after examining only a small subset of the input pixels. Only these prototype candidates are matched against the input pattern. There are two possible problems.

First the touching character arrays can contain incomplete characters at the beginning or end of these arrays. Since they used white area as character boundaries, broken characters can give erroneous results. Second, in the touching character segmentation loop, if the window is sliding past every pixel, the computational time is costly. If the window slides past more than one pixel, it is possible the process may give inaccurate boundaries.

Kahan and Pavlidis proposed an segmentation algorithm which combines vertical features with classification results.[8] From a line image, the algorithm first constructs a line adjacency graph (LAG). The LAG has nodes that correspond to dark segments in the image scan-lines and branches that join nodes whose corresponding segments touch. "Blobs" are found as connected components of the LAG. A statistical Bayesian classifier using the structural description produced by the LAG traversal was used to discriminate the single character blobs from multicharacter blobs. The criterion they set for a blob being a multiple character is that for each character class, the probability computed by the classifier for the blob being that character is less than the *a priori* probability. Then algorithm applies the second difference function introduced in the last section to the multicharacter blobs to find the possible break points. Both the left and the right portion of each break point are sent to a character classifier. The classifier will return two new lists of alternatives and the corresponding probability scores. If the new scores are worse than the original score, other break points can be tested. If none of them produces better result, the original blob is reinstated but reported as a probable reject. Note Kahan and Pavlidis' algorithm works well only on blobs of two touching characters.

Tsujimoto and Asada proposed a recursive segmentation-and-recognition algorithm for segmenting touching characters.[5] It first finds candidates of break positions, and then confirms these break positions by using a recursive segmentation-and-recognition process, and finally by the determination of the linguistic context. In order to resolve ambiguity at each stage in the segmentation procedure, Tsujimoto and Asada embedded knowledge about character composition and knowledge about omni-fonts. The algorithm achieved a 120 character per second text reading speed.

The algorithm first computes all the connected components in the line image and combines the components that are above and below one another, such as two components in "i" and"j". As a result, each component defines a single character or touching characters. Multicharacter components are identified by a character recognition algorithm. If a component is having a similarity measure with a certain class greater than *a priori* threshold, it is a single character component, otherwise it is a multicharacter component. The multicharacter components are further processed by the following procedures.

Tsujimoto and Asada[5] introduced a break cost function as a metric to evaluate the degree of touching. The break cost is defined at each position between neighboring columns and is calculated by accumulating the black pixels of the image obtained by the AND operation between neighboring columns. The candidates for break position are obtained by finding local minima of a smoothed break cost function. The start and end positions of touching characters are also set

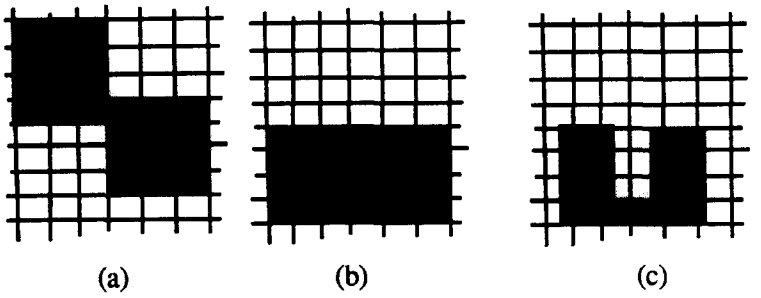


Fig. 20. A new metric to evaluate the degree of touching. (a) An image of two touching blobs. (b) Vertical projection of (a). (c) Result of the new metric.

as candidates and multiple candidates are set at smooth local minima. Candidates with dominant sharp peaks in the break cost distribution are selected as final break positions, and the others are considered weak candidates. Tsujimoto and Asada argued that this new metric gives a more accurate measure of touching characters than vertical projection. Figure 20 shows an example. In this example, the vertical projection does not provide any information on the touching position. However, the new metric is capable of exhibiting a prominent connection between the two blobs.

A decision tree is constructed to represent the search order for break position candidates. The root has a Null value. Each node represents a subset of the component area and has a list of the segmented area, whose first element is geometrically connected to the last element of the preceding node in a depth-first order. The depth of the tree represents the order in which the characters are being segmented. The descendants of each node are sequentially ordered in the following ways.

(1) Among segmented areas ended by advantaged candidates, smaller segmented areas precede the larger ones.

(2) Among adjacent segmented areas ended with weak candidates, larger segmented areas precede the smaller ones.

(3) A segmented area ended with a weak candidate succeeds the segmented areas ended by advantaged candidates.

The order of the decision tree allows for the advantaged candidates to be searched first in depth-first order search. During the search, the segmented area represented by the node is sent to a recognizer. If a node is recognized with acceptance, then its descendants are searched. Otherwise, its siblings are searched. If all the nodes on a path from the root to a leaf are accepted, the search is finished.

A pattern represented by each node in a decision tree is recognized by a multiple similarity method, which is designed to be insensitive to the varieties of omnifonts. The method produces multiple recognition candidates. The decision for accepting a candidate is based on the similarity measures of the candidates. The authors also employed knowledge about character composition in their algorithm to reduce the search cost. The algorithm was implemented on a hand-made recognition board consisting of a RISC processor and VRAM. A scanner with 300 dpi was directly connected to the recognition board. In an experiment with touching characters taking out from 32 documents in various fonts, the performance of the segmentation and recognition reached 99.7%.

8. CONCLUSION

We have presented various techniques for character segmentation in machine-printed documents. Some techniques work on fixed-pitch fonts, others on proportional fonts, some use feature-based approaches others use recognition-based results. However, all the techniques we presented here have one thing in common, they all work only on binary line or word images. As we pointed out in the Introduction, a binarization preprocess can affect the segmentation results, and recognition results in the sequel, in a negative way. We have found that many of the segmentation and recognition errors are actually induced by binarization processes. In many cases the broken and touching characters in binary images were well-formed characters in the original gray-scale images. The broken characters can be caused by low thresholds, and the touching characters can be caused by high thresholds used in the preprocesses. Figure 21 shows an example. The first binary image contains touching characters and the second one contains broken characters, although the characters in the gray-scale image are neither

Fig. 21. Illustration of different binarization results.

touching nor broken. Gray-scale images contain rich information for character segmentation. We believe that segmentation directly from gray scale images will give promising results.

Acknowledgement—The work is supported in partial by the research and fellowship grant of The University of Michigan Rackham Graduate School.

REFERENCES

1. L. O'Gorman and R. Kasturi, Document image analysis systems, *Computer* July, 5–8 (1992).
2. Yi Lu, B. Haist, L. Harmon, J. Trenkle and R. Vogt, An accurate and efficient system for segmenting machine-printed text, *U.S. Postal Service 5th Advanced Technology Conference*, Washington D.C., November, Vol. 3, pp. A-93–A-105 (1992).
3. S. N. Srihari and G. Zack, Document image analysis, in *The Eighth International Conference on Pattern Recognition*, France, pp. 434–436 (1986).
4. R. G. Casey and G. Nagy, Recursive segmentation and classification of composite character patterns, in *Proc. 6th International Conference on Pattern Recognition*, Munich, Germany, pp. 1023–1026 (1982).
5. S. Tsujimoto and H. Asada, Resolving ambiguity in segmenting touching characters, in *The First International Conference on Document Analysis and Recognition*, pp. 701–709 (1991).
6. O. Nakamura, M. Ujiie, N. Okamoto and T. Minami, A character segmentation algorithm for mixed-mode communication, *Systems Computers Japan* **16**, 1277–1284 (1985).
7. N. Okamoto, O. Nakamura and T. Minami, Character segmentation for mixed-mode communication, in *Information Processing* (R. E. A Mason, Ed.), Elsevier Science North-Holland (1983).
8. S. Kahan and T. Pavlidis, On the recognition of printed characters of any font and size, *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-9**, 274–287 (March 1987).
9. D. G. Elliman and I. T. Lancaster, A review of segmentation and contextual analysis techniques for text recognition, *Pattern Recognition* **23**, 337–346 (1990).
10. Y. Tsuji and K. Asai, Character image segmentation, in *SPIE Vol. 504 Applications of Digital Image Processing VII*, pp. 2–10 (1984).
11. M. Shridhar and A. Badreldin, Context-directed segmentation algorithm for handwritten numeral strings, *Image Vision Comput.* **5**(1), 3–9 (1987).
12. Yi Lu, On the segmentation of touching characters, in *The Second International Conference on Document Analysis and Recognition* (October 1993).
13. R. L. Hoffman and J. W. McCullough, Segmentation methods for recognition of machine-printed Characters *IBM J. Res. Dev.* 153–165 (March 1971).
14. Y. Tsuji and K. Asai, Adaptive character segmentation method based on minimum variance criterion, *Systems Computers*, Japan **17**(7) (1986).
15. Y. Tsuji, J. Tsukumo and K. Asai, Document image analysis for reading books, in *SPIE Advances in Image Processing, Vol. 804*, pp. 237–244 (1987).
16. R. G. Casey, Coarse–fine OCR segmentation, *IBM TDB* **23**(8) (January 1981).
17. V. A. Kovalevsky, *Image Pattern Recognition*. Springer, Berlin (1980).

About the Author—YI LU graduated with a diploma majored in mathematics from Shanghai Teacher's College, Shanghai, China, in 1980. She received a M.S. degree in Computer Science from Wayne State University, Detroit, Michigan in 1983 and a Ph.D. degree in Computer, Information, Control Engineering from the University of Michigan, Ann Arbor, Michigan in 1989.

From 1989 to 1992, she was a research scientist at the Environmental Research Institute of Michigan, Ann Arbor, Michigan. Currently she is an assistant professor at the Department of Electrical and Computer Engineering at the University of Michigan–Dearborn. Her research interests include computer vision, pattern recognition and artificial intelligence. She has publications in the areas of low-level vision processes, knowledge-based computer vision processes, handwritten digit recognition, machine-printed character segmentation and recognition, medical image analysis, and knowledge integration. Currently, she is an associate editor for *Pattern Recognition*.