Retrieving Information from Document Images: Problems and Solutions

Fu Chang
Document Analysis and Recognition Laboratory
Institute of Information Science 20
Academia Sinica
128 Academia Road, Section 2
Nankang, Taipei 115
Taiwan R.O.C.
Email: fchang@iis.sinica.edu.tw

ABSTRACT

An information retrieval system that captures both visual and textual contents from paper documents can derive maximal benefits from DAR techniques while demanding little human assistance to achieve its goals. This article discusses technical problems, along with solution methods, and their integration into a well-performing system. The focus of the discussion is very difficult applications, for example, Chinese and Japanese documents.

In addition to their large group of potential readers, the latter types of documents create many technical issues that deserve experts' attention. The complicated Chinese or Kanji characters, for example, create a serious problem for image binarization. The coexistence of vertical and horizontal textlines on the same page makes document segmentation difficult. The large number of characters also creates challenges to the recognition and retrieval of textual contents.

Problems discussed in this article are related to these issues. Solution methods are also highlighted, with the emphasis placed upon some new ideas, including window-based binarization using scale measures, document layout analysis for solving the multiple constraint problem, and full-text searching technique capable of evading machine recognition errors.

1. Introduction

Faithful representations of paper documents need to cover both visual and symbolic information. The former includes icons, graphics, flow charts, tables, and the layout of documents while the latter covers all the textual contents, including words, textlines, paragraphs and articles. Fetching both types of information at fairly low expense is possible using today's technology. For example, a scanner can be used to obtain the visual contents (or images) of documents while OCR software can be used to transcribe their textual contents. Obtaining all the contents with the same level of faithfulness, however, is not an easy task.

The problem can be somewhat lessened if only the retrieval values of the contents are required. Thus, to prepare the *retrievable* contents out of paper documents, one can do the following.

(i) Use a scanner, and perhaps also image-processing software, to obtain visual representations of the documents.

(ii) Use OCR software to obtain text transcriptions of the same pages.

To retrieve the information thus obtained and stored in a database, one can further perform the following operations.

(iii) Use a search engine to find interesting information in the textual contents.

(iv) Retrieve the images of the corresponding documents that contain the information.

The combination of the above operations takes advantage of the fact that images are good for viewing purposes and textual contents are ideal for information searching. Admittedly, machine-transcribed texts can hardly be error-free. But such machine errors do not appear on images and, therefore, do not have any impact on the viewing process. Their impact on the search process can also be minimized if certain error-tolerance mechanisms are inserted in the process. The net result will be the elimination of the need for human assistance in preparing retrievable information (except for feeding documents into scanners) and successful fetching of information from the visual contents.

Load balancing is thus the major consideration. What is deficient in one working component (machine transcription of textual contents) can be compensated by other components (producing visual contents and fetching information from them). Certainly, requirements become more stringent for the components onto which the burden is shifted. First, the quality of visual contents must be sufficiently good for human eyes. Next, retrieving information requires correct reading of not only the words but also their orders out of the documents. Finally, the text-searching capability also needs to be strengthened so as to bypass possible errors occurred in machine transcription.

While all these issues may already fall within the realm of DAR, we want to address in this article some specific problems that are associated with Chinese documents (or similar types, for example, Japanese documents). Through five years of effort, our laboratory has put together a system that is capable of transforming paper documents into viewable and searchable contents. There are four major components embedded in this system. (1) Image Processing: the core is an algorithm that transforms gray-scaled images into binary images. (2) Document Analysis**:** work done by this component includes skew detection and correction (to detect the skew angle of the document relative to a standard vertical line, and also to adjust the angle to 0^0 as possible as one could), segmentation of pictures and frames from textual contents, extraction of textual blocks, textlines, and individual characters etc. (3) OCR Engine: this performs recognition of printed documents consisting of Chinese, English and numeral characters. (4) Search Engine: the searching capability of this component is enhanced by an OCR-error-tolerant function.

Undoubtedly, some of the methods used in other applications (for example, English documents) can readily solve problems associated with Chinese documents. Examples include skew detection and correction, segmentation of non-textual information, and some feature extraction methods. But the complicated nature of Chinese documents creates many new or, perhaps, more difficult problems. It was observed long time ago that Chinese characters are probably the most difficult type of characters for machine recognition. The large number of characters and their complicated structures are major causes for difficulties. Characters also require the capability of recognizing the subtle distinctions between foreground and background pixels. Chinese documents are also known to allow two types of word order—horizontal and vertical—to appear on the same page. They enlarge the degree of freedom for the possible document layout structures. The fact that many Chinese characters are themselves composed of elementary elements also increases the complexity of algorithms designed to analyze their structures.

In this article, we will thus focus our discussion on the problems that are specific to Chinese document applications and, above all, problems related to information retrieval. The rest of the paper is organized as follows. A discussion of binarization of document images is given in the next section. Section 3 covers general issues involved in analyzing layout structures of Chinese documents. Section 4 discusses a textual search mechanism that incorporates machine-error tolerance. Section 5 is a brief summary.

2. Binarization of Document Images

Scanners use internal mechanisms to transform gray-scale images (in which each pixel assumes one of 256 possible values) into binary images (in which each pixel assumes either black or white value). However, when one applies them to produce binary images of old Chinese documents, many small but complicated characters will become blurry.

Years ago, before we started to work on our own solution, we thought naively that a proper threshold would solve this problem. Here, by threshold, we mean a value, say T, based on which each pixel can be classified as either black or white, according to whether its gray value falls below or above T[1]. To test this hypothesis, we conducted a very simple experiment. We chose an old newspaper article and produced a gray-scale image as the source image. From it, we then created a series of binary images by using each possible level, ranging from 0 to 255, as the cutting value (namely, T) for blacks and whites. To our surprise, we observed the following outcomes.

For low values of T, simple characters appeared broken while complicated characters looked all right. As the value of T increased to a higher value, simple characters became intact but complicated characters became blurry (an example is now shown in Figure 1). Furthermore, we could not find any value of T at which both simple and complicated characters looked all right at the same time.

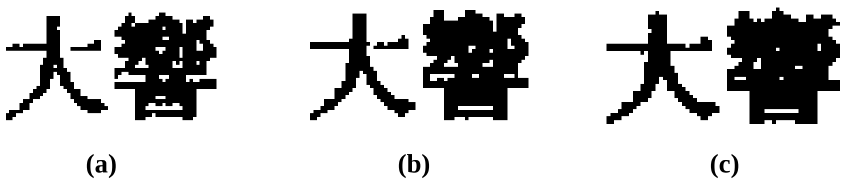


Figure 1. The binary images of two Chinese characters, one simple and one complicated. The three sets of images were obtained by setting (a) T = 65, (b) T = 80, and (c) T = 95. As the threshold value increased, simple character became intact while complicated character became blurry.

When we scrutinized the source image, we found the reason for this unhappy result. Like many other characters, Chinese characters are composed of strokes. Within a small but complicated character, the background regions that are surrounded by strokes become extremely small. Because of the point-spread function that is used by scanners to calculate gray values, the pixels within these "tiny valleys" appear darker (i.e., their gray values are lower) than the average background pixels. Thus, when a low value of T is chosen as the cutting value for blacks and whites, the tiny valleys are classified as white, but some weak pixels in simple characters are also classified

[1] By convention, low gray values represent dark grades.

as white, resulting in broken characters. When T increases to a certain higher level, the small characters become intact, but the tiny valleys are classified as black, resulting in blurry characters.

Our finding thus proves that a global threshold cannot solve the binarization problem. It also suggests that local comparison of gray values may be necessary in order to make distinctions between background and foreground pixels. Let us consider the following method.

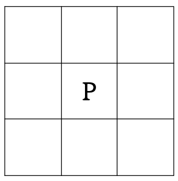


Figure 2. $W_p(3)$ gets all its members from a window of size 3×3 centered at pixel p.

For each pixel p, let $W_p(n)$ be the collection of all neighboring pixels of p that falls within a window of size n×n centered at p (Figure 2). Let

$$\mathrm{Mini}(p) = \min\{g(q)\text{: } q \varepsilon W_p(n)\},$$
$$\mathrm{Maxi}(p) = \max\{g(q)\text{: } q \varepsilon W_p(n)\}.$$

We then consider the quantities g(p) – Mini(p) and Maxi(p) – g(p). If the former is larger, then g(p) is closer to the highest gray value than the lowest value in this window. Since a pixel with a high gray value is most likely a background pixel, we classify p as a background (i.e., white) pixel. Otherwise, we classify p as a foreground pixel.

The merit of the above method is that it does not rely on any threshold value for classification. This method, however, does depend on a proper choice of the window size, as we will explain later. For this reason, we call it the *window-based binarization method* or *local binarization method*. In Figure 3, a result of applying this method is shown, where the window size was set as 3×3. It is seen that all the characters surface out rather nicely. The clear drawback is the appearance of noises throughout the background area.

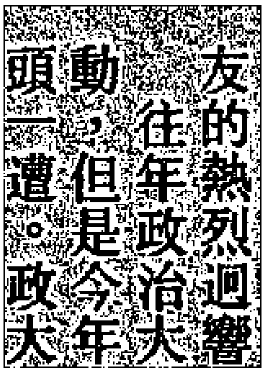
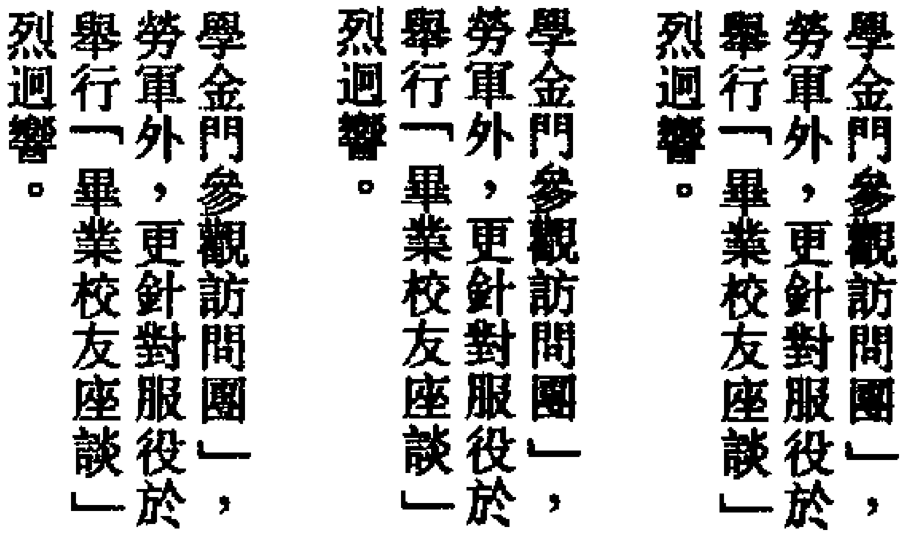


Figure 3. The result of applying the window-based binarization method, where the window size was set as 3×3.

A moment's reflection would immediately suggest a possible way to improve the solution: perhaps we should restrict the application of window-based binarization only to areas where it is really useful. But where is the method useful? Presumably, it is useful where a gray value fails to reflect the true status of a given pixel. Normally, when a gray value is very high or very low, it is a sure evidence for the (white or black) status of the pixel. The uncertain area is somewhere in between the extreme values.

This fact suggests that we can make combined use of the global threshold method and local binarization method. When the gray value g(p) of pixel p is far away from the global threshold T, we classify p as black or white according to whether g(p) lies to the far left or far right of T. If, however, g(p) is in the near neighborhood of T, the window-based binarization method will be used to determine the binary value of p.

The results of applying this hybrid method described above are shown in Figure 4, where global threshold was calculated using Otsu's method [1]. The window sizes were set to three different values so that we could make comparisons. Obviously, all


學金門參觀訪問團」，
勞軍外，更針對服役於
舉行「畢業校友座談」
烈迴響。

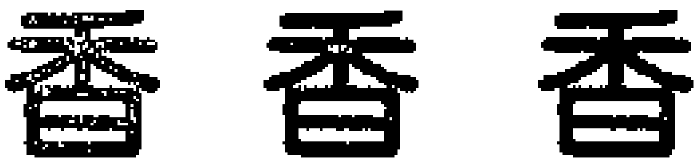
(a) **(b)** **(c)**

Figure 4. Results of the combined use of the global threshold method and window-based binarization method with the window size set to (a) 5×5, (b) 15×15, and (c) 21×21.

the results look better, since background noises have disappeared. However, one problem remains to be solved. What window size should we set for the local binarization method? As Figure 4 shows, the complicated characters become blurry as the window size increases. Therefore, based on this figure, it seems that we should choose a size of 5×5 or smaller.

However, if we look at Figure 5, we may conclude differently. It seems that a larger size is more appropriate in this case, for as the window size decreases, foreground noises (or white noises) also show up.

So what is going on? Why are different window sizes required for different cases?



(a) **(b)** **(c)**

Figure 5. Outputs of the hybrid method. The window size for the local binarization method was set to (a) 5×5, (b) 9×9, and (c) 13×13.

First of all, what is the difference between the two cases? Under preliminary scrutiny, we observe that the characters shown in Figure 4 are smaller than the character in Figure 5 and, therefore, have thinner strokes than the latter. Next, we note that the appropriate window size ought to be compatible with the average stroke width of the characters to which the local binarization method is applied.

But why is this magnitude so crucial? When we use the local binarization method, we compare a given pixel p with all the pixels falling within a window centered at p. If the size of this window is appropriate, then there is a good mixture of foreground and background pixels within this window. The window-based binarization method would be able to gather all the relevant information and make the correct decision.

Let us see what would happen if we employ an inappropriate window size. Suppose that the size is set smaller than it should be, and that p is a foreground pixel. Then all the pixels falling within the window are themselves foreground pixels. In this case, we are comparing unfairly a foreground pixel with all the foreground pixels! An incorrect conclusion can be drawn for p if p happens to be not as dark as its neighbors. This accounts for the white noises observed in Figures 5a and 5b.

On the other hand, suppose that the window size is set larger than it should be, and that p is a pixel lying within a tiny valley. Then, the window centered at p may contain some background pixels lying outside of the character. Since p lies within the valley, it appears darker than the remote background pixels. We may thus misclassify p as a black pixel. This accounts for the blurry characters in Figures 4b and 4c.

The above consideration suggests that certain magnitude, called a “scale” in this article, must be calculated before window-based binarization can be applied.

There may be many ways to obtain this magnitude. One straightforward way is to calculate the average run-length for each character. However, there is a serious problem with this approach. The run-length measure is not defined until a binary image is created. But once a binary image is initialized, it becomes a self-fulfilling process: if the character appears blurry in the image, its run-length estimate will become large. Applying window-based binarization based on this estimate will still result in a blurry character.

The second approach is to infer the stroke widths of characters based on the sizes of textlines. Here the size of a textline L is defined as the height or the width of L, depending on whether L is a horizontal or a vertical line. This is a workable approach since the extraction of textlines can be done independently of the binarization process. The inferences of stroke widths can also be reliable since stroke widths usually grow proportionally to the sizes of textlines. In this approach, however, success with window-based binarization must depend upon employing a flawless pre-step for textline extraction. This is rather burdensome. Another drawback is that the same textline may contain both boldface and non-boldface characters, whose stroke widths certainly differ from each other.

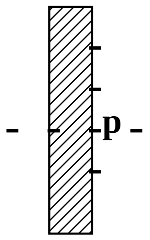
The third approach is to obtain multi-scale measures on the source image. The idea is as follows. We can use a set of “yardsticks” for the measurements. Each of them is suitable for a certain scale. We then select the best reading out of all the possible measurements.

A yardstick is actually a 2n-dimensional vector, where n is an even number. When n = 2m, the general form of the vector is

$$(\overbrace{1, . . . ,1}^{m}, \overbrace{-1, . . . , -1}^{n}, \overbrace{1 . . . , 1}^{m}).$$

The 2n-dimensional vector will be called the yardstick of scale n, abbreviated as Y_n. Thus, in this terminology, Y_2 = (1, -1, -1, 1), Y_4 = (1, 1, -1, -1, -1, -1, 1, 1), etc. There is no yardstick of scale 3 since we do not allow odd-numbered scales. The reason for this will be give later.

To obtain a measurement, a yardstick must be applied at a specific point. Thus, if a pixel p is given, we first gather its n neighboring pixels, all lying on the horizontal line passing through p. We then form a vector H_n from the n gray values. The *horizontal* reading of Y_n at p is then defined as $Y_n \bullet H_n$, where • is the sign for vector multiplication. To obtain the *vertical* reading of Y_n at p, we can gather n vertical neighbors of p and form a vector V_n from their gray values. The vertical reading is then defined as $Y_n \bullet V_n$. Finally, the total reading of Y_n at p is defined as max(vertical reading, 0) + max(horizontal reading, 0).



Coefficients of Y_2	1	-1	-1	1
Coefficients of H_2	205	35	38	212
Coefficients of V_2	30	34	35	29

Figure 6. Application of Y_2 in both directions at pixel p. Pixels are indicated by dashes.

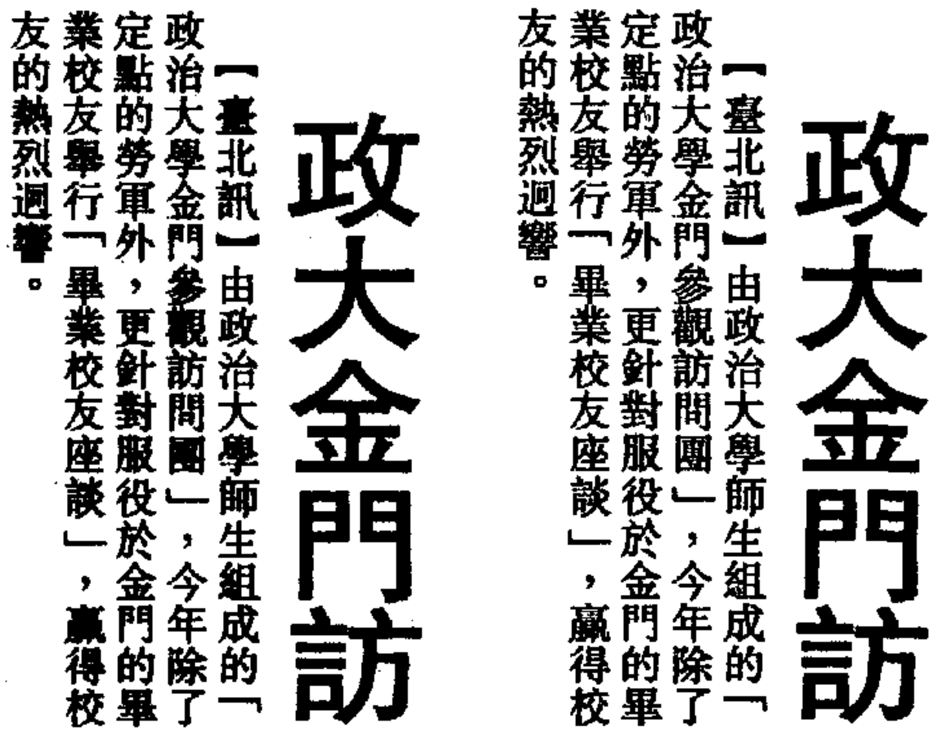
As an example, let us consider the effect of applying Y_2 at pixel p, as shown in Figure 6. The horizontal reading is (1, -1, -1, 1) • (205, 35, 38, 212) = 314. In this case, since p is located on a vertical stroke of 2 pixels in width, the reading is obtained by summing two background values and subtracting two foreground values. The result is a large positive number. The vertical reading, on the other hand, is obtained by summing two foreground values and subtracting two foreground values also. Since all the foreground values are close to each other, the resulting value (-10) is a number close to zero.

If we apply a yardstick to all the pixels of the same stroke, summing the readings and then taking the average, we get some information about this stroke. As a general rule, a stroke reacts strongly to yardsticks of compatible scales. Moreover, if a stroke gets the highest average reading from a yardstick of scale n, then its average width must be close to n. Strokes, however, are not easily identified from images. On the other hand, characters or parts of characters can be found with relative ease. Based on these ideas, we have come up with a process for determining the window sizes as follows.

In order to apply the window-based binarization method, we have to first come up with a global threshold T. We can thus employ this T to create a binary image and find all the connected components from the image. Using some techniques not described here, we can classify connected components into textual ones (i.e., characters or parts of characters) and non-textual ones (table lines, icons, graphics etc.)

Each textual connected component is either a whole character or part of a character. Thus, for such a component, say C, we can apply all the yardsticks described above and find the one that yields the highest average reading. The scale of this yardstick is defined as the scale of C. The window size for C is then set to s+1, where s is the scale of C.

Two facts are worth mentioning here. First, the window-based binarization method is quite robust to slight variations of scale estimates. A deviation of one or sometimes even two pixels is tolerable. This is why we are satisfied with even-numbered scales. The nice thing about even-valued scales is that the corresponding windows become symmetric with respect to their centers. Secondly, window-based binarization would not be helpful for components whose scales are equal to or higher than 8. This means that, for computing window sizes, we only need 4 yardsticks of scale 2, 4, 6, or 8 respectively. Those components reacting most strongly to the yardstick of scale 8 presumably have stroke widths larger than 8.



政大金門訪

【臺北訊】由政治大學師生組成的「政治大學金門參觀訪問團」，今年除了定點的勞軍外，更針對服役於金門的畢業校友舉行「畢業校友座談」，贏得校友的熱烈迴響。

政大金門訪

【臺北訊】由政治大學師生組成的「政治大學金門參觀訪問團」，今年除了定點的勞軍外，更針對服役於金門的畢業校友舉行「畢業校友座談」，贏得校友的熱烈迴響。

Figure 7. Left panel: binary image obtained using the global threshold method. Right panel: binary image obtained using the hybrid method.

In Figure 7, we show two binary images. One of them was obtained using the global threshold method, where the threshold value was calculated using Otsu's method. The other image was obtained using the hybrid method, that is, a combination of both the global and local binarization methods. The improvements for small and complicated characters obtained using the hybrid method should be clear when one compares the two output images. Note that the common input to both methods was a gray-scale image from a clipped newspaper article, dated July 13, 1993.

Readers may also be happy to learn that the hybrid method not only improves the visual quality of binary images, but also helps to improve the performance of OCR engines. To test this assertion, we gathered three sets of samples: recent articles, articles printed in 1996, and articles printed in 1969. Each set consisted of 20 testing samples, each sample containing a few hundred Chinese characters.

Two OCR engines were employed in the tests. Both were from commercial software so that there was no problem of possible bias. From each of the testing articles, we produced a gray-scale image and two binary images, one by way of the global threshold method and the other using the hybrid method. We applied the OCR engines to one and then the other binary image, and the increments in the recognition rate are recorded in Table I. It is clear from the last column of this table that the improvement obtained using the hybrid method is more apparent for the old aged articles than the recent ones.

Table I. Testing Results

	OCR Performance (Global Threshold Method)	OCR Performance (Hybrid Method)	Increased OCR Performance
Recent Articles	96%~99%	88%~99%	0%~10%
1996 Articles	66%~88%	89%~98%	5%~23%
1969 Articles	26%~66%	60%~88%	20%~37%

3. Document Layout Analysis

Binarization captures the visual contents of documents but does not interpret them. To understand printed documents, a feature one can take advantage of is the apparent layout structure, from which it is possible to infer the distinct locations of characters and also their reading order. Well-segmented characters make the job of machine recognition easier. Understanding the reading order is a necessary ingredient for understanding the meaning of textual contents, for it is their order of compositions that makes it possible for characters to convey meaning. While these tasks may seem easy for human readers, it is difficult to design machines that can do the same things.

Documents have their textual contents organized in hierarchical order. Individual characters are organized into textlines, which in turn are organized into paragraphs etc. In the past, many methods looked entering points into the hierarchical system. For example, the recursive X-Y cuts method [11] seeks to enter at the top of the hierarchy. It searches for large horizontal or vertical gap and decomposes an image into two sub-images. It then repeats the same operation for each of them. The maximal white-rectangles method [12] also seeks to enter the hierarchy at the top. But instead of decomposing the image based on projection profiles, it looks for all the maximal white rectangles that are imbedded in the layout structure. It then segregates the image from the large white rectangles. The run-length smearing method [13], on the other hand, seeks to enter at the bottom of the hierarchy. It goes along each scan line and blackens the small spaces between black pixels. The smearing operation has thus the effect of gluing foreground pixels into connected pieces that can be textlines, paragraphs, or mixtures of things, depending the parameter value being used. The textline construction method [14] may be said to enter the hierarchy in the middle. Taking advantage of the fact that textlines, in printed documents at least, are approximately aligned (from top and bottom for horizontal textlines, or from right and left for vertical textlines), this method endeavors to find them by means of a pair of rails (as in railways).

All the above methods are good in certain contexts but all have their limitations also. The recursive X-Y cut method is good in the context of single news article, but on a full page of a newspaper, there can be many articles, in which case this method can not be applied. The maximal white-rectangles method is applicable to more complicated environments. However, the size of the white rectangles may not be a reliable clue for segregation. In Chinese documents, textlines can go in either the horizontal or vertical direction. The groups of horizontal and vertical textlines are naturally separated from each other, even though the white rectangles between them are not very large. In Figure 8, for example, white rectangle A is a separation zone but its size is smaller than that of some other white rectangles (B and C) that are not separation

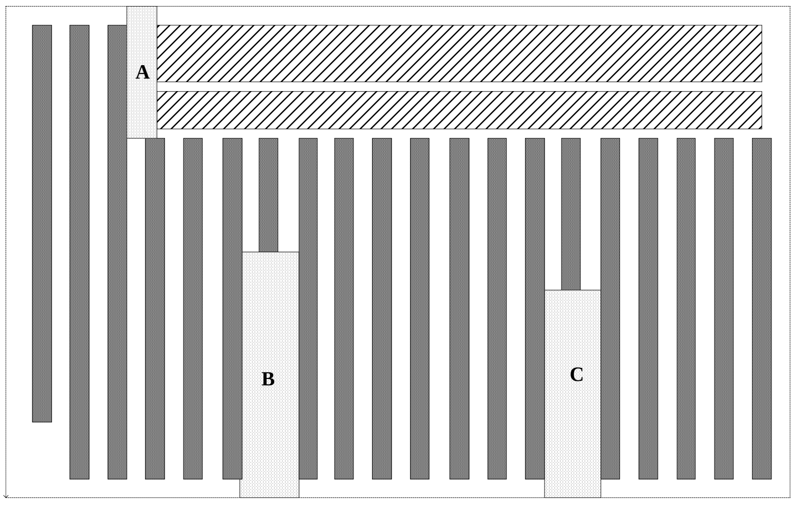


Figure 8. The layout structure of a Chinese article is displayed. Headlines (striped boxes) are horizontal. The rest of textlines (gray boxes) are vertical. Dashed lines form the borders of the image. Dotted boxes are white rectangles.

zones at all.

The run-length smearing method has a problem in the parameter setting. At the 300 dpi level, the inner spacing in small textlines is only 2 to 3 pixels wide, but it can grow to 80 pixels wide in headlines. Furthermore, even within the same textline, sizes of gaps can vary wildly. For example, the gaps around punctuation marks can be 18 pixels wide, while those between characters remain 2 or 3 pixels wide. Thus, counting on a constant size to bridge the gaps would loose all the flexibility that is desired.

The textline construction method employs more structural elements than other methods do. However, two issues arise when it is applied to Chinese documents. First, beginning from the same starting point, it is possible to obtain two textlines, one going horizontally and the other vertically. Only one of them is legitimate, but neither of them can be excluded a priori (Figure 9a). Secondly, a textline so constructed can overextend into the area of another textline, resulting in conflicts that have to be resolved (Figure 9b). Both issues are related to the fact that two possible directions, horizontal and vertical, are allowed for textlines in Chinese documents. The higher

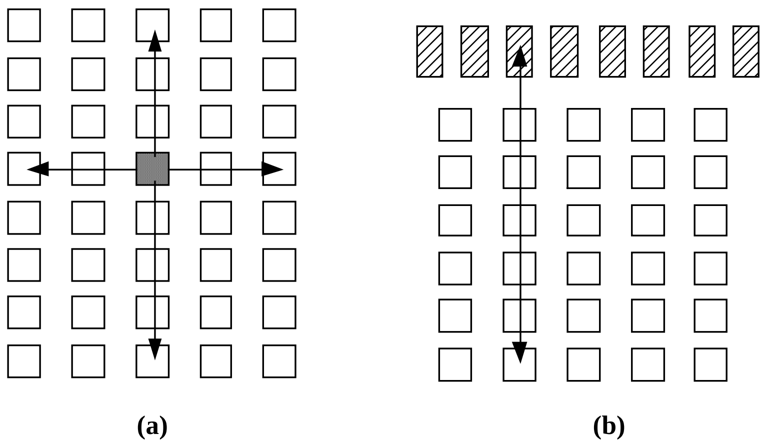


Figure 9. All boxes represent characters. (a) Starting from the same character (gray box), two textlines can be constructed, one going in horizontal direction, and the other vertical direction. (b) A vertical textline overextends into the area of a horizontal textline (striped boxes).

degree of freedom lies in the heart of these problems.

In fact, all the methods reviewed above can be useful in certain contexts, but it is important to first identify the context correctly before applying them. In our view, all methods for layout analysis employ some *split and merge* operations. Run-length smearing, for example, is a merge operation while X-Y cut is a split operation. To get a reasonable outcome, these operations rely upon certain parameter values to function. As explained in the previous section, the value of the scale needs to be computed before the window-based binarization can proceed. Unlike that situation, most of the parameter values associated with the document layout problem cannot be computed independently. They are involved in the problems that have to be solved.

Document layout analysis is a *multiple constraint problem*, where the objects (paragraphs, textlines, and characters, etc.) to be constructed or identified are the *unknown variables* built into the constraints. There are actually three sets of constraints that are met by most Chinese documents.

A. Composition Constraints: paragraphs are composed of contiguous textlines of similar sizes, which in turn are composed of contiguous characters of similar sizes; characters are composed of contiguous components, i.e., clusters of connected foreground pixels.

B. Alignment Constraints: characters contained in horizontal (vertical) textlines are properly aligned along their top (right) and bottom (left) edges; textlines contained in horizontal (vertical) paragraphs are properly aligned along their top (right) and bottom (left) edges.

C. Spacing Constraints: the spacing within a textline is less than the spacing between textlines; moreover, the spacing within a paragraph is less than the spacing between paragraphs (Figure 10).

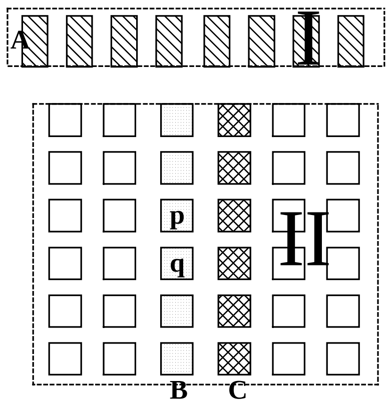


Figure 10. Two paragraphs, I and II, are enclosed in dashed lines. Paragraph I contains textline A, which is horizontal. Paragraph II contains B, C and other vertical textlines. Each labeled textline is identified by the same fill pattern. The spacing within B (= distance between p and q) is less than that between B and C. The spacing within paragraph I (= distance between B and C) is less than that between I and II.

Each set of constraints involves more than one unknown variable, and each unknown variable also occurs in more than one set of constraints. The third set of constraints, moreover, incorporates some parameters (inner spacing and outer spacing) whose values depend on other variables (textlines and paragraphs).

All the methods reviewed above attempt to solve this problem in a straightforward manner. They seek a solution for some unknown variable, which in turn can be used to solve for another unknown variable, and so on. Unfortunately, for most hard constraint problems, straightforward solutions rarely exist. As an alternative, we propose the following method. We start with an approximate solution, that is, a solution satisfying some but not all the constraints. Based on this temporary solution, we then estimate the unknown parameter values appearing in some other constraints. We can then derive a better solution (that is, a solution that satisfies more constraints) by referring to the estimated values. The solution thus obtained can still be improved if more or better knowledge is generated.

At the start, the choice of primitives is important. Foreground pixels can certainly be used as primitives, but they are not as handy as components. A component can be represented by its enclosing box, so that four numerical values (the x- and y-coordinates of its upper left and lower right corners) are sufficient to determine its location. The distance between two components can also be defined in terms of the distance between their enclosing boxes.

When working on components, run-length smearing and textline construction become almost the same method. They both look for collections of contiguous components that are properly aligned along the top and bottom edges, or along the right and left edges. The textline construction method tolerates a small amount of misalignment and is thus more robust. Our solution method employs the same technique to find all the textlines in document images.

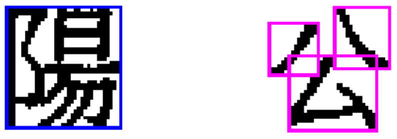


Figure 11. (a) Two logical components join into one physical component. (b) The enclosing boxes of three physically separated components come into contact with each other.

Many Chinese characters are composed of more than one component, and it is possible for two logical components of the same character to actually join to form one physical component on the images (Figure 11a). On the other hand, even if some of the components are physically separated, their enclosing boxes may touch each other and be joined into a single box (Figure 11b). For all these reasons, almost every legitimate textline in a document image contains at least one single-box character.

To take advantage of this fact, we can pick out the component boxes whose width-to-height ratios are close to 1. To ensure that they are not arbitrary boxes, we can further check if there are any other boxes whose sizes are similar to theirs. We can then allow textlines to grow only from those boxes whose sizes are similar to some other boxes. If, after this stage, there are still component boxes not included by any textlines, we proceed to search in their neighborhood for any other components that can be put together to form a box that is larger in size and better (closer to 1) in width-to-height ratio. If this is possible, we can proceed with textline construction for the newly formed boxes.

After the textline construction stage, two more things need to be done. First, those textlines growing with the wrong word order must be eliminated. Next, those textlines growing too long need to be pruned.

Eliminating textlines with incorrect word order has the effect of adjusting the solutions to bring them in line with the spacing constraint. According to this constraint, the inner spacing[2] of a textline is smaller than its outer spacing. Thus, if two textlines grow out of the same component box, the one that has wider inner spacing than its counterpart ought to be eliminated (Figure 12a).

[2] The gaps within a textline may not be constant in size. Therefore, the inner spacing of a textline is defined to be the gap whose size is close to that of most of the gaps within the textline.

For the purpose of textlines pruning, we first find all the white rectangles in the document image. They serve as clues for possible separation zones. Therefore, we examine those textlines that cross some white rectangles. However, rather than simply pruning them, we make a further comparison. For those textlines whose inner spacing is much smaller than the spacing created by the white rectangle (Figure 12b), we proceed with pruning.

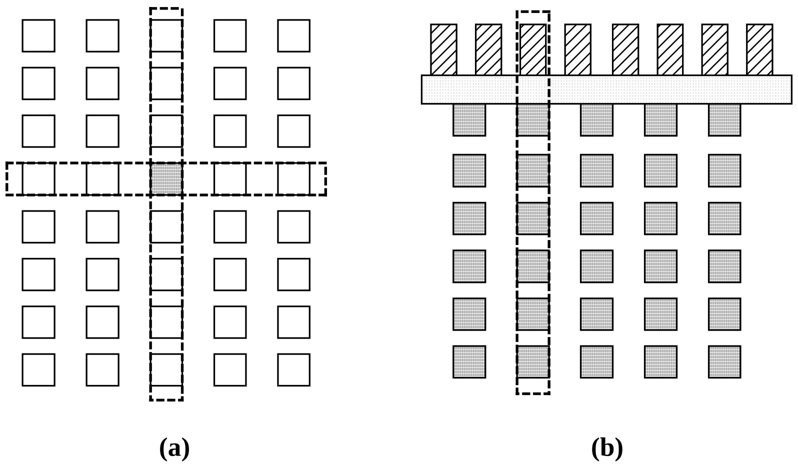


Figure 12. (a) Two textlines (enclosed by dashed lines) grow from the same (half-toned) component box. The horizontal line has wider inner spacing than the vertical line. (b) The vertical textline (enclosed by dashed lines) extends beyond a white rectangle (dotted box) with relatively wide spacing.

The pruning operation has the effect of forcing the solutions to comply with the spacing constraints for both textlines and paragraphs since the combination of the two constraints implies that the inner spacing of a textline is smaller than the outer spacing of the paragraph.

After the textline consolidation stage, we proceed with paragraph construction. This is rather a simple operation. It starts with any textline and proceeds to find, from any of the four possible directions, textlines of similar sizes. The same operation is then applied recursively to each of the textlines thus obtained.

Textlines and paragraphs are not simply end products of the process. They also serve as inputs to other adjustment procedures. Thus, after the textline construction stage, those irregular boxes appearing within the same textline can be put together to form regular boxes. In Figure 13, for example, boxes A and B can merge into one character box, based on the alignment requirement for textlines. Boxes C and D in the same figure can also merge into one box, thus conforming to the requirement that each textline is composed of similar-sized characters.

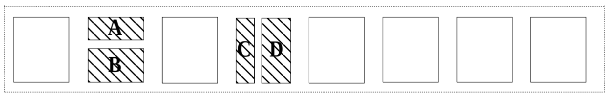


Figure 13. A horizontal textline (enclosed in dashed lines) incorporates irregular components A, B, C and D.

Similar adjustments can be made within paragraphs. During the textline construction stage, some textlines may not be fully connected because some conservative parameter values are being used there. After the paragraph formation stage, those properly aligned but disconnected pieces can join together to form a longer line (Figure 14a). However, there is a danger of joining textlines that are really part of different columns. As a remedy, recursive X-Y cuts can be used to undo the ill result (Figure 14b).

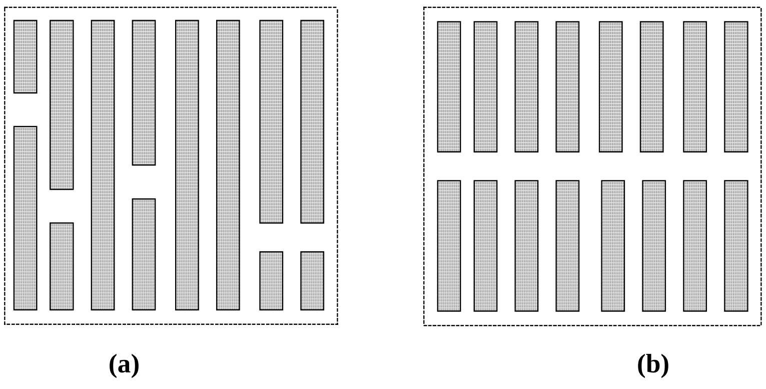


Figure 14. (a) Broken pieces falling within the same paragraph can be connected to make complete pieces. (b) Textlines belonging to different columns can be separated by means of recursive X-Y cuts.

The solution path suggested in this section thus evolves as a self-adjustment process. It starts to construct solutions that satisfy some easy constraints and then gradually improves them by referring to better knowledge gained during the process. The goal of this process is to adjust solutions so that they satisfy more difficult constraints.

The constraint conditions considered in this section can be met by many Chinese documents, but certainly not by all of them. Some special documents (business cards, for example) may require that one or several of the conditions be removed or weakened, and that a few others be added. However, one nice thing about this approach is that it can easily adjust the solution path according to changes in the constraints. The reason for this easy adjustment is that constraint conditions can be naturally classified into categories. (We categorized them into three sets). Thus, when one category of constraints remains unchanged, all the corresponding operations remain unchanged. The idea of modularity thus naturally grows in this solution method.

For testing the document layout methodology, we have prepared a set of samples collected from magazine and newspapers. There are approximately 1400 of them. Many samples consist of a single article, while others have more than one article. We classify these samples according to their layout structures into seven categories: (1) horizontal headlines with L-shaped contents, (2) horizontal headlines with rectangle-shaped contents, (3) vertical headlines with L-shaped contents, (4) vertical headlines and rectangle-shaped contents, (5) mixture of texts and pictures, (6) mixture of texts and tables, and (7) mixture of Chinese and English characters (within the same textlines).

Each time when we make ready a version of layout-analysis program for testing, we randomly select a few samples from each of the seven categories, plus some “tough” samples that are always selected for the testing purpose. In the last testing we made, 50 samples were chosen and two problems were detected out of the test results. Both of the problems have to do with the assumptions being taken by the layout-analysis program being tested.

The first problem occurs at two parallel textlines, each consisting of two characters only. The spacing between the two textlines is actually smaller than the spacing within each textline. This fact obviously violates our spacing consumption for textlines. Since this kind of problem occurs rarely in regular documents, we decide to do nothing about it. The second problem has to do with the listed items that contain only two characters. Since our program assumes that a textline has to contain at least three characters, those items were not segmented as textlines. To solve this problem, care must be taken to deal with listed objects. This is a subject we would rather not go any further in this article.

4. Retrieving Information from Machine-Transcribed Contents

There are thousands of distinct commonly used characters in the Chinese language. This means that for each of the characters to be recognized by a machine, it has to be matched against at least thousands of template images. Due to the time constraint, which can be crucial in many applications, the trade-off between the time spent by the recognizer and the accuracy attained by it must be considered. For this reason, the thousands of characters involved can lead to significant loss in recognition accuracy.

The usual measure for accuracy calculates the percentage of the rank-first candidates that hit the targets. If more candidates are included in the calculation, however, the hit rate can clime to nearly 1. For example, the hit rate achieved by the first five candidates generated by our recognizer can reach way above 99% for most applications even though the rate achieved by the rank-first candidates varies from application to application. Since machine-transcribed contents are used for the searching task only, the first five candidates can be more useful than the rank-first candidates. The idea is as follows.

In the data prepared for the searching task, we store the rank-first candidate generated by the recognizer for each of the characters extracted from original documents. Assuming now that the user intends to look for a keyword string, say $K_1K_2...K_n$ in the original documents, our search engine will search for the information in the prepared data, instead. Since this data is likely to contain errors, the string $K_1K_2...K_n$ may appear as $V_1V_2...V_n$ in the prepared data, where V_i is either K_i or a character misidentified by the recognizer for K_i, for $i = 1,2,...,n.$ To ensure that users get what they want, the search engine should look for all the possible variants $V_1V_2...V_n$ in the prepared data.

To do so, the search engine looks up a pre-stored table that contains for each character entry those characters that can be misidentified for it. The misidentified characters for entry E are those that are expected to appear in the short-listed candidates for E. Therefore, they can be prepared in the following way. For each training sample of E, we collect the first N short-listed candidates generated by the recognizer. We then calculate the accumulation rate for the candidates and sort them accordingly to obtain the first M candidates, where M is a number close to N.

The success rate of the search engine is thus related to the hit rate of the table that lists all the misidentified characters. Normally, the hit rate can reach 99.5% when no more than 5 misidentified characters are stored for each entry.

Thus, if the keyword is a bi-gram (two-character string), the engine can find all the occurrences of the keyword in the original images, with a probability rate higher than 99%. If the keyword consists of more than two characters, we can relax a little bit by requiring the engine to find all the strings $U_1U_2...U_n$, where at least n-1 entries in $U_1U_2...U_n$ match with the corresponding entries in $V_1V_2...V_n$. Then the engine can find all the occurrences of the keyword in the original images, with a probability rate higher than 99.9%.

If textual contents are stored in the same order as they appear in the original documents, searching for a keyword and all its variants through these contents would require M×L×P times of character comparison, where M is the number of misidentified characters, L is the length of keyword string, and P is size of the textual contents. For the later reference, this search method will be called linear search. The search time can be greatly reduced if we use a pre-stored file to record for each character entry all the positions in which it occurs in the textual contents.

The way to use the pre-stored file is the following. When a search engine starts to find $V_1V_2...V_n$, a variant of the keyword string, we start to construct a matrix with n rows and P columns, where n is the length of the character string and P is the size of textual contents. First, all the entries in the matrix are filled with 0's. Now, if the pre-stored file has an occurrence of V_i at the m-th position in the textual contents, then we put 1 at position (i, m-i+1) in the matrix.

Thus, suppose that $V_1V_2...V_n$ actually appears in the text, and that its first character appears at the 37th position in the text. Then we put 1 at (1, 37) in the matrix. Its second character must then appear at the 38th position, so we put 1 at (2, 37) in the same matrix, and so on. Finally, all the entries in column 37 in this matrix are 1's.

Having done the above, we start to examine all the columns in the matrix. If the keyword is a bi-gram, we pick up all the columns that have two 1's as entries. If the keyword has more than two characters, then we pick up all the columns that have either all 1's or only one 0. Finally, we output the indices of all these columns. They indicate the starting positions of all the occurrences of $V_1V_2...V_n$ in the textual contents.

The method described above is essentially a fast-matching method, and will be referred to as “fast search” method. The new aspect of this method is that it has to guarantee the matching of all possible variants of the given keyword. The computational cost is then M times higher than that which would be required for exact matching (i.e., matching the keyword only), where M is the maximum number of misidentified characters listed for each character entry.

Table II. Time Consumptions by Two Methods for Bi-gram Search

Type of Search		Exact	Error-Tolerant
Search Speed	Linear Search	0.626	0.595
	Fast Search	0.12	0.067
	Linear Search / Fast Search	5.217	8.881

Table III. Time Consumptions by Two Methods for Tri-gram Search

Type of Search		Exact	Error-Tolerant
Search Speed	Linear Search	0.633	0.613
	Fast Search	0.093	0.047
	Linear Search / Fast Search	6.806	13.043

In the above, Table II and Table III list out the time consumptions by both search methods, where “tri-gram” stands for three-character string, “exact search” refers to the search of keyword itself, and “error-tolerant search” refers to the search of all possible variants of the keyword. The search speed is measured in terms of second per 1000 pages, where one page consists of 500 characters on average.

We note that there can be many ways of implementing the fast search method. It is possible to obtain higher speed at the cost of huge amount of memory space. The speeds listed in these two tables are derived from an implementation scheme that balances the utilization of both memory space and computation time.

5. Summary

In the previous sections, we have described the special problems that are encountered in building a system that supports information retrieval from Chinese document images. Two types of solutions have been considered: capturing visual and textual contents from paper documents and retrieving information from prepared data. In the introductory section, we discussed how all these solutions could be put together to obtain a system that requires minimal human interference. In the remainder of the article, we focused on some special technical problems and their solutions.

Context sensitivity lies at the heart of the problems considered here. For the binarization problem, the global threshold method establishes a foundation and sets the condition for window-based binarization. For the latter to work properly, another process is employed to set the window sizes correctly. In the document layout problem, the solution path is not so straightforward. Contextual information is part of the solution being sought. The solution scheme thus evolves into a self-improvement process. While context sensitivity makes the two problems difficult to solve, it actually makes the task of information retrieval easy. There is an advantage in the fact that a keyword used in the searching process normally consists of at least two characters. Inaccurate and piecemeal information contained in machine-transcribed contents can be synthesized into much more reliable and useful information, when used for the purpose of information retrieval.

REFERENCES

1. N. Otsu, A threshold selection method from gray-scaled histogram, *IEEE Trans. Systems, Man, and Cybernetics*, Vol. 8, pp. 62-66, 1978.
2. G. Johannsen and J. Bille, A threshold selection method using information measures, *Proc. Sixth Intern. Conf. Pattern Recognition*, pp. 140-143, Munich, Germany, 1982.

3. J. M. White and G. D. Rohrer, "Image thresholding for optical character recognition and other applications requiring character image extraction," *IBM J. Res. Develop.* Vol. 27, No. 4, pp. 400-411, 1983.
4. W. Tsai, Moment-preserving thresholding: a new approach, *Computer Vision, Graphics, and Image Processing*, Vol. 29, pp. 377-393, 1985.
5. J. N. Kapur, P. K. Saboo, and A. K. C. Wong, A new method for gray-level picture thresholding using the entropy of the histogram, *Computer Vision, Graphics, and Image Processing*, Vol. 29, pp. 273-285, 1985.
6. J. Kittler and J. Illingworth, On threshold selection using clustering criteria, *IEEE Trans. Systems, Man, and Cybernetics*, Vol. 15, pp. 652-655, 1985.
7. Y. Liu and S. N. Srihari, "Document image binarization based on texture features," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 19, pp. 540-544, 1997.
8. E. Giuliano, O. Paitra, and L. Stringa, Electronic character reading system, *U.S. Patent* 4,047,15, 1977.
9. W. L. Hwang and F. Chang, Character Extraction from Documents Using Wavelet Maxima, *Image and Vision Computing*, Vol. 16, pp. 307-315, 1998.
10. F. Chang, K. H. Liang, T. M. Tan, and W. L. Hwang, Binarization of document images using Hadamard multiresolution analysis, *Proceedings Intern. Confern. Document Analysis and Recognition*, Banglore, 1999.
11. G. Nagy, S. C. Seth, and S. D. Stoddard, Document analysis with an expert system, *Proceedings Pattern Recognition in Practice II*, Amsterdam, 1985.
12. H. S. Baird, S. E. Jones, and S. J. Fortune, Image segmentation by shape-directed covers, *Proceedings 10th ICPR*, Atlantic City, pp. 820-825, 1990.
13. F. M. Wahl, K. Y. Wong, and R. G. Casey, Block segmentation and text extraction in mixed text/image documents, *Comput. Vision Graphics Image Process*, vol. 20, pp. 375-390, 1982.
14. T. Pavlidis and J. Zhou, Page segmentation and classification, *CVGIP: graphical models and image processing*, vol. 54, pp. 484-496, 1992.
15. A. Dengel, Initial learning of document structure, *Proceedings Intern. Confern. Document Analysis and Recognition*, Tsukuba, 1993.
16. D. Wang and S. N. Srihari, Classification of newspaper image blocks using texture analysis, *Comput. Vision Graphics Image Process*, vol. 47, pp. 327-352, 1989.
17. A. K. Jain, B. Yu, Document representation and its application to page decomposition, *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 20, pp. 294-308, 1998.
18. L. O'Gorman and R. Kasturi eds., *Document Image Analysis*, Los Alamitos, IEEE CS Press, 1995.

19. H. Fujisawa and Y. Nakano, A top-down approach for the analysis of documents, *Proceedings 10th Intern. Conf. Pattern Recognition*, pp. 113-122, Atlantic City, 1990.
20. J. Fisher, S. Hinds, and K. D'Amato, A rule-based system for document image segmentation, *Proceedings 10th Intern. Conf. Pattern Recognition*, pp. 567-572, Atlantic City, 1990.
21. H. Baird, Anatomy of a versatile page reader, *Proceedings IEEE*, vol. 80, pp. 1059-1065, 1992.
22. F. Chang, et al., A Document Analysis and Recognition System, *Inter. Conf. Document Analysis and Recognition*, Ulm, Germany, 1997.
23. F. Chang and T. R. Chou, Problems and Solutions for a Content Capture and Information Retrieval System, *4th IAPR International Workshop on Document Analysis Systems*, 2000, Rio de Janeiro, Brazil.