IJDAR (1999) 2: 67–79

International Journal on IJDAR Document Analysis and Recognition

Efficient and flexible text extraction from document pages

Pietro Parodi, Roberto Fontana

International School for Advanced Studies, Via Beirut 2-4, I-34014 Trieste, Italy; e-mail: parodi@sissa.it

Received April 4, 1999 / Revised June 1, 1999

Abstract. This paper describes a novel method for extracting text from document pages of mixed content. The method works by detecting pieces of text lines in small overlapping columns of width w', shifted with respect to each other by $\epsilon < w'$ image elements (good default values are: $\epsilon = 1\%$ of the image width, $w' = 2\epsilon$) and by merging these pieces in a bottom-up fashion to form complete text lines and blocks of text lines. The algorithm requires about 1.3 s for a 300 dpi image on a PC with a Pentium II CPU, 300 MHz, MotherBoard Intel440LX. The algorithm is largely independent of the layout of the document, the shape of the text regions, and the font size and style. The main assumptions are that the background be uniform and that the text sit approximately horizontally. For a skew of up to about 10 degrees no skew correction mechanism is necessary. The algorithm has been tested on the UW English Document Database I of the University of Washington and its performance has been evaluated by a suitable measure of segmentation accuracy. Also, a detailed analysis of the segmentation accuracy achieved by the algorithm as a function of noise and skew has been carried out.

Key words: Text extraction – Document segmentation – Computational complexity – Segmentation accuracy

1 Introduction

In recent years there has been an increased need for systems able to read a vast amount of documents and to convert hardcopy images into digital format automatically. A key phase of the machine reading of documents is the segmentation of a page into text and non-text regions, so as to provide appropriate input to the optical character recognition (OCR) system.

Despite the many efforts spent on the subject (see [1] for a thorough review) there is still much room for improvement in document segmentation techniques, which are arguably the key-factor for improving the overall performance of an automatic reading system: even very good OCR systems can be almost useless when text-extraction is performed poorly, which is often the case in existing systems for documents with an irregular layout. Furthermore, since the availability of *accurate, flexible* and *fast* segmentation methods is so critical for automatic reading systems, it is necessary to develop objective criteria to evaluate the proposed segmentation techniques.

After a brief review of previous approaches to document segmentation (Sect. 2), this paper describes a novel algorithm based on foreground/background analysis [2,3,4] for extracting text from document pages (Sects. 3, 4). This method is accurate and fast, and it deals successfully with documents with an arbitrary layout, documents where graphical features and text are intertwined, skewed documents and noisy documents. A relevant part of the paper is devoted to a detailed characterization of the behavior of the algorithm: a suitable segmentation accuracy measure is introduced, and segmentation accuracy is evaluated for a standard database and as a function of noise and skew (Sect. 5); time performance (Sect. 6.1) and time complexity (Sect. 4.2) are reported; the role of the different parameters and thresholds of the algorithm (Sects. 4.1, 6.1) and its weaknesses and breaking points (Sect. 6.2) are discussed. The relation of this work with other works based on foreground/background analysis is also discussed (Sect. 6.3).

2 Previous work

Techniques for text extraction and document segmentation are traditionally subdivided into three main categories: bottom-up, top-down and hybrid techniques.

Bottom-up techniques [5,6,7,8] progressively merge evidence at increasing scales to form, e.g., words from characters, lines from words, columns from text lines. They are usually more flexible than top-down methods, but they may suffer from the accumulation of mistakes when going from the small-scale details up to the large-scale features.

Correspondence to: P. Parodi

Top-down techniques [9,10,11,12] start by detecting the large-scale features of the image (e.g., columns) and proceed by successive splitting until they reach the smallest-scale features (i.e., individual characters, or text lines). For the procedure to be effective, a priori knowledge about the structure of the page is necessary. These techniques are therefore particularly useful when the layout is constrained, such as is often the case when considering pages from scientific journals.

Most methods do not fit into one of these two categories and are therefore called *hybrid*. Among these we cite methods based on texture analysis [13,14,15] and methods based on background analysis [2,3,4,27].

In methods based on *texture analysis* the problem of reconstructing the document layout is seen as a problem of texture segmentation. The document page is subdivided into small regions each of which is classified as belonging to one of a few categories (text, drawing, image, etc) according to an analysis of its texture. Once each region in the image has been tentatively classified, a globally consistent segmentation is carried out by the usual techniques of machine vision. Examples of methods using texture analysis are those based on Gabor filtering and mask convolution [13], fractal signature [15], and wavelet multiscale analysis [14]. All these methods are quite general and flexible but they are also computationally demanding.

In methods based on *background analysis*, the white spaces in the image are the basis for image segmentation. Our method ideally belongs to this category, and we therefore spend a few more words to discuss them. A more detailed comparison of our work with these works based on background analysis will however be carried out only after describing our proposed algorithm, in Sect. 6.3.

The idea of using the background as a basis for image segmentation has been used first by Baird et al. [2]. Their method works by covering the background with a set of maximal white rectangles, and by identifying the different text regions as the connected non-white components. The main disadvantages of this method are that (1) it assumes a Manhattan layout, that is, text regions must be bounded by rectangles; (2) it assumes that text regions are surrounded by vertical and horizontal streams of white space. Reported time performances for a journal document page are about 10 seconds on a DEC VX 11/8550; the method requires pre-processing in which connected components are extracted, and this implies an additional time of about 10 seconds for the same architecture.

In Pavlidis & Zhou [3], segmentation is performed by subdividing the document page into horizontal stripes of small height and by detecting the white vertical gaps in each stripe. The intervals between white vertical gaps are then merged with those of pairwise adjacent horizontal stripes to form text columns. Thanks to the use of thin horizontal stripes, a moderate amount of skew does not affect the goodness of segmentation. A limitation of this method is that it is limited to documents where text regions are surrounded by straight streams of white space, and is therefore unsuitable for documents where, e.g., text is cropped around pictures, as in Fig. 3. Also, columns separated by vertical lines instead of white spaces are likely to cause trouble. Furthermore, a run-length smearing is used as a pre-processing before computing the vertical projection profile in the horizontal stripes, and the smearing value must be tuned according to the font size, which in turn is unknown, and is therefore chosen, in practice, according to the resolution (10 pxl at 300 dpi). Even if the smearing value is appropriate for the main font size, the class of documents to which the algorithm can be safely applied is restricted to those documents where all fonts are approximately of the same size.

Also based on background analysis and on covering the background with a set of white tiles are the works by Antonacopoulos & Ritchings [4] and Antonacopoulos [27]. Unlike [2], they do not require the layout to be Manhattan-like, nor do they require as in [3] the text regions to be surrounded by *straight* streams of white spaces. The main disadvantages of the method developed in these works are that (1) it assumes that text regions are surrounded by (not necessarily straight) streams of white spaces: it is therefore not suitable to be applied to documents like that in Fig. 3, where text merges into graphical features (specifically, in this image this is due to noise in the scanning stage); (2) it implicitly assumes that only one font is present: the identification of white streams is in fact preceded by a vertical smearing, and the smearing value is estimated from the image data by estimating in turn the typical value of the base line skip. This works well with images such as those shown in the paper, where only one font is present and the base line skip is very regular, but it is likely to cause trouble when different font sizes are present.

A more thorough survey on papers on document segmentation can be found in [1]. The recent paper by Jain and Yu [8] also contains a brief up-to-date survey of existing methods for page segmentation put in tabular form.

3 Rationale of the approach

If the document consists of a single column of text, a simple recipe for extracting text is: compute the horizontal projection profile of the page (Fig. 1) i.e., count the number of black dots for each scanline (then group scanlines whose projection value is greater than zero into clusters. Each of these clusters is, under ideal conditions, a line of text.

At least two factors prevent this approach from being directly applicable to more general situations: first, even a slight skew may spoil the projection profile: this requires skew-correction mechanisms, or excessive care in scanning the document; second, text can appear in more columns, and it can appear mixed with graphic elements.

However, the naive approach outlined above for detecting text lines in a one-column, text-only page suggests this strategy for dealing with pages with a more complex layout:

1. subdivide the document into *overlapping* small columns of height equal to the document height, and

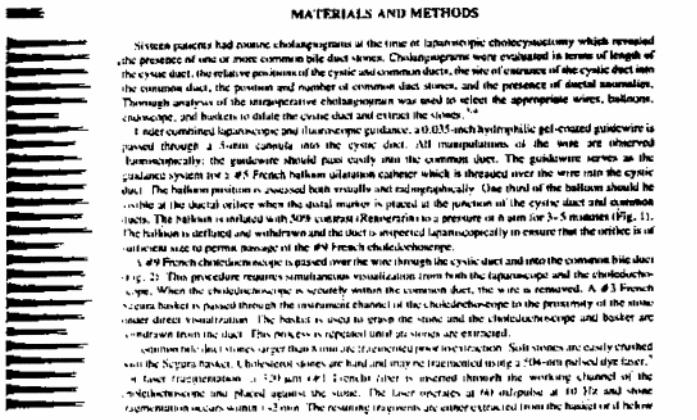

Fig. 1. A single-column, text-only document page, and on the left the horizontal projection profile. The picture is borrowed from [26]

on each of them compute the projection profile to find text lines. The strips of non-white scanlines extracted at this stage are called *line elements*;

2. combine the line elements extracted in Step 1 to find the text lines.

Using small columns drastically reduces problems related to skew, and allows us to deal with documents with arbitrary layouts. Specifically, it is easy to realize that:

- if the columns are of width $w^{'}$ and the shift between two successive columns is ϵ, all text of at least width $w = w^{'} + 2\epsilon$ is going to be extracted, regardless of the layout of the page and of the presence of graphic elements;
- the shift between successive columns, ϵ, is directly related to the precision by which text is reconstructed: the beginning and the end of each text line will be recovered with an error at most equal to ϵ.

The minimum text width w and the segmentation precision ϵ are worst-case estimates. If, for example, a text line is surrounded on each side by a white space of sufficient width on both sides, then – thanks to a module which performs white spaces extraction inside each line element – the beginning and end of the text line are determined with a precision of 1 pixel and text width can be as low as once or twice the font size.

4 The algorithm

This section describes the basic version of the algorithm and examines its sequential and parallel complexity.

Input. The input is a binarized image, scanned at a certain resolution ρ. The size of the image is $I_{\mathrm{L}} \times I_{\mathrm{W}}$, where I_{L} is the image length and I_{W} is the image width. The document may be skewed, up to about 10 degrees. Notice that skew due to careless scanning is usually no more than 2 or 3 degrees.

1. Column subdivision. Subdivide the document into columns C_i of width equal to $w^{'} = w - 2\epsilon$, and height

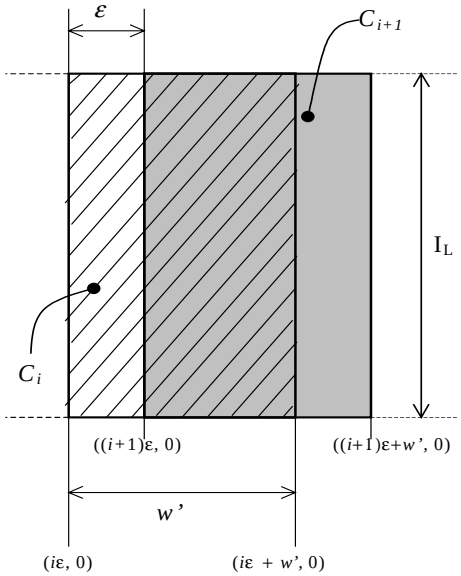

Fig. 2. An illustration of the column subdivision process. Two successive columns (C_i and C_{i+1}) are shown. Column C_i is white with black oblique lines, and C_i is gray. If $w^{'} > \epsilon$, there is a region of overlapping between the two columns (gray region with black oblique lines). The 2D coordinates of the relevant points are indicated

I_{L} equal to the height of the document. The column C_i is chosen so as to scan the area defined by the rectangle $[i\epsilon, i\epsilon + w^{'}] \times [0, I_{\mathrm{L}}]$. This means that column $C_{i+1}$is shifted to the right by ϵ image elements with respect to C_i, and that the two columns partially overlap if $w^{'} > \epsilon$ (see Fig. 2). By construction, the number of columns is $n_{\mathrm{COL}} \sim I_{\mathrm{W}}/\epsilon$.

2. Extraction of line elements. For each column C_i ($i = 1, \ldots n_{\mathrm{COL}}$), compute the horizontal projection profile, that is, count the number of black pixels for each row of image elements contained in the column. If the density of black pixels in a row (number of black pixels divided by the total number of pixels) is less than a certain threshold δ_{BP}, we call it 'white'. In general, δ_{BP} depends on the amount of noise, which can be estimated automatically. For clarity, the dependence of δ_{BP} on noise is discussed separately (see Sect. 5.2). For standard, not-too-noisy images, a good default value for δ_{BP} is 0.

We extract the chains of pairwise adjacent non-white rows in each column, denoted as *line elements*. These will be the atomic units from which the text lines will be constructed in a bottom-up fashion. Figures 3B and 4B show examples of line elements extraction.

3. Histogram analysis. Compute the histogram of occurrence for each height of the lines found in Step 2. Find all the modes, that is, the maxima of the histogram. The peaks of the modes corresponding to text have an approximately fixed width. This is because the height of a text line is comprised, under ideal conditions, between two extreme cases: that in which a line has neither ascendants nor descendants, e.g., '11', and that in which a line has both ascendants and descendants, e.g., 'pbpbpb'. The ratio between the two cases is about two, although

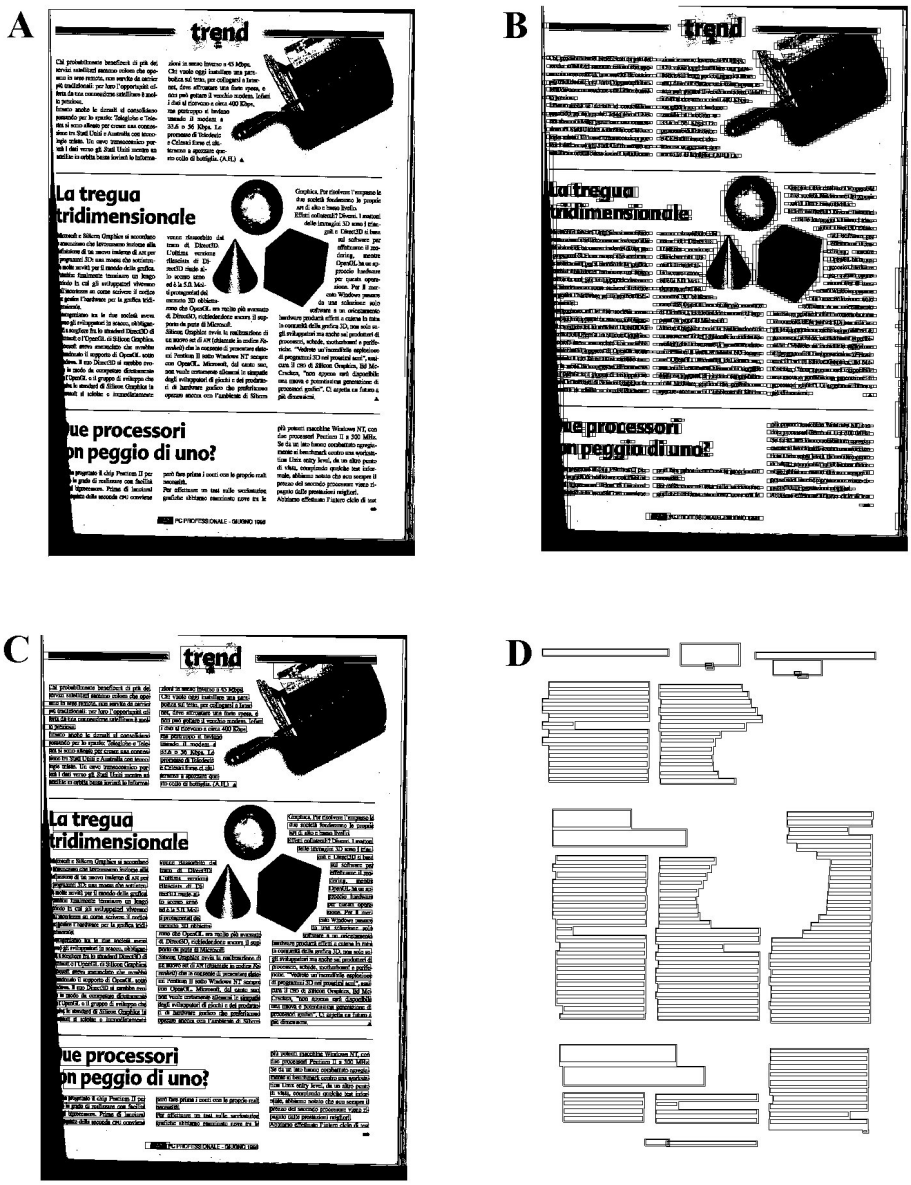

Fig. 3. A A document page scanned at 300 dpi. The layout is irregular and there is text cropped around pictures, and text merging into the black stripe on the left. Several fonts are present. **B** The line elements extracted from each column, as in Step 2 of the algorithm. **C** Text lines extracted from the document, after Step 4 of the algorithm. **D** Text lines after gathering into blocks

the exact ratio depends on the font style; we therefore expect a width of about log2 if we draw the histogram in logarithmic scale. Experimentally, a width of about $\delta_{HIST} = 2log2$ was found to be adequate. Therefore, we do not need to extract the width of each peak: we can assume that its width is fixed and equal to δ_{HIST}.

4. Extraction of text lines. The input to this step is a set of line elements $\lambda_1, \ldots \lambda_n$. The problem is to join these line elements to form text lines. The procedure is as follows:

(a) Associate one or more modes of the histogram extracted in Step 3 with each line element λ_i. More than one mode can be associated with the same line element, since two different peaks of the histogram may overlap.
(b) Text lines are lists of connected line elements. Before building the lists, however, it is important to perform *white spaces extraction*. White spaces inside each line element can be easily detected by a vertical projection profile inside each line element. Vertical white lines (defined similarly to white lines in Step 2, with the same value of δ_{BP}) are grouped into white strips and if a white strip of width greater than δ_{WHITE} times the average width of the character ($\delta_{WHITE} = 2$ in the current implementation) is found, one of the following actions is taken:

- if the white strip is at the beginning or at the end of the line element, the boundaries of the line element are redefined;
- if the white strip is inside the line element, the line element λ_i is split into two line elements $\lambda_i^{'}$, $\lambda_i^{''}$, each of them with its new boundaries.

The output of this procedure is a set of new line elements $\lambda_1^{'}, \ldots \lambda_m^{'}$, which are at most twice as the original line elements.

(c) Construct a graph whose elements are the line elements after white spaces extraction, $\lambda_1^{'}, \ldots \lambda_m^{'}$, and such that there is a *link* between $\lambda_i^{'}$ and $\lambda_j^{'}$ iff

(i) $\lambda_i^{'}$ and $\lambda_j^{'}$ belong to at least one common mode of the histogram;

(ii) $\lambda_i^{'}$ and $\lambda_j^{'}$ belong to adjacent, successive columns;

(iii) $\lambda_i^{'}$ and $\lambda_j^{'}$ partially overlap.

If a line element should be connected to two different line elements belonging to the successive column, no link is added at all. Therefore, the degree of each node is at most two.

(d) Find the connected components of the graph constructed in Step (c). Since the maximum degree of the nodes of the graph is two, the connected components are simply lists of line elements.

(e) *Skew detection.* Suppose that the line elements $\lambda_1, \ldots \lambda_r$ all belong to the same list L_k. Let $(x_1, y_1), \ldots (x_r, y_r)$ be the centers of the line elements in the text line. The skew of the individual text line can be estimated by computing the best-fit line through $(x_1, y_1), \ldots (x_r, y_r)$. This line can be written as $y = a_k x + b_k$, where a_k is related to the tilt angle θ_k of the line by the relation $\tan\theta_k = a_k$. The skew angle θ^* can be estimated as the most frequent tilt angle θ_k, which in turn can be determined by drawing the histogram of the θ_k's, whose range goes from $\pi/2$ to $\pi/2$ with intervals of width $\delta\theta$. $\delta\theta$ is chosen according to the desired precision on the skew. In our implementation, $\delta\theta = 0.1°$. If θ^* is less than $\delta\theta$, the algorithm proceeds as if there were no skew at all. Otherwise, a change of coordinates is performed. The new coordinates are chosen so that the image is seen as unskewed in the new system: $x^{'} = x\cos\theta^* + y\sin\theta^*$, $y^{'} = -x\sin\theta^* + y\cos\theta^*$. All operations – including rectangular hull extraction, classification, blocks formation – will be performed, from this point on, according to the $(x^{'}, y^{'})$ system of coordinates. To unify notation we will use $(x^{'}, y^{'})$ even when $\mid \theta^* \mid < \delta\theta$, in which case it is to be understood that $x^{'} = x$, $y^{'} = y$.

(f) For each list $L_k = \{\lambda_1, \ldots \lambda_r\}$, find the rectangular hull $[x^{'}_{\min}, x^{'}_{\max}] \times [y^{'}_{\min}, y^{'}_{\max}]$ bounding all line elements $\lambda_1, \ldots \lambda_r$.

(g) A simple test is performed on each of these lists to discard those which cannot possibly be text lines. This is done using a very loose and conservative criterion, which is based on the observation that knowing the height h and length l of the rectangular hull, we can predict the number $n_c \sim l/h$ of characters in a single line of text. A rough measurement related to the number of characters n_c is the number of black-to-white transitions in the middle line ($y^{'} = (y^{'}_{\min} + y^{'}_{\max})/2$) of the rectangular hull of the list. If the number of transitions is not between $l/(h\delta_{B/W})$ and $\delta_{B/W} l/h$, the list is classified as non-text.

Furthermore, it is required that the tilt angle θ_k of the list under consideration be consistent with the skew computed in (e). Observe that a line which is unskewed might appear slightly skewed because of the relative position of ascendants and descendants, e.g., the first line elements might have descendants but no ascendants and the last line might have ascendants but no descendant. It is easy, however, to find an upper bound for this spurious skew: in ideal conditions, the spurious skew will never be more than the tilt angle θ_s of a line which starts from the centroid of a line element with descendants but no ascendants and ends in a line element with ascendants but no descendants, $\theta_s = \frac{h}{2(n_k-1)\epsilon}$ where h is the font size and n_k is the number of line elements belonging to the list). This is confirmed experimentally by computing the skew of individual text lines for an unskewed document page produced with a text editor and directly converted to an image file without passing through a scanner so that no skew is introduced in the scanning process: this computation immediately shows that θ_s is a good estimate for the upper bound of the spurious skew.

Therefore, all lists of line elements whose tilt angle θ_k is such that $\mid \theta_k - \theta^* \mid > \delta_{TILT}\theta_s$ (the factor δ_{TILT} is set to 3 in the current implementation and has been introduced to yield a more conservative test).

(h) *Merging text lines which overlap.* It is possible that for some reason two neighbor line elements fail to connect in Step 4c although they are indeed part of the same text line. To deal with cases like this, we perform post-processing that merges text lines which overlap and are of comparable height. For each linked list of elements, L_i, we compute the average height of the line elements, h_i, and the standard deviation σ_i. We then merge two overlapping text lines iff $h_j - \delta_{\mathrm{MERGE}}(\sigma_i + \sigma_j) \leq h_i \leq h_j + \delta_{\mathrm{MERGE}}(\sigma_i + \sigma_j)$ where δ_{MERGE} has been set to 2 in the current implementation.

Figures 3C and 4C show the state of the algorithm after Steps 4f and 4g have been performed.

So far, text lines have been extracted individually. Blocks of text lines are now extracted, so as to provide at least a partial reading order to the OCR system.

5. Blocks formation. We create blocks by constructing a directed graph whose nodes are the text lines extracted in Step 4, and an arc connects text line l_1 to text line l_2 iff:

(i) l_1 and l_2 have compatible height. This is decided by the following conservative criterion. Remember that for each line element one or more modes of the histogram are associated with it. One or more modes of the histogram are therefore associated also with the text line formed by a given list of line elements by simply considering the union of all sets of associated modes. l_1 and l_2 are considered to be of compatible height if the set of modes associated with l_1 and that associated with l_2 share at least one element.

(ii) The lowest point of l_1 is above the highest point of l_2, and the base line skip between l_1 and l_2 is compatible with the assumption on the least among the common font sizes. In this case 'compatible' means that the base line skip must not be more than $\delta_{\mathrm{B.L.SKIP}}$

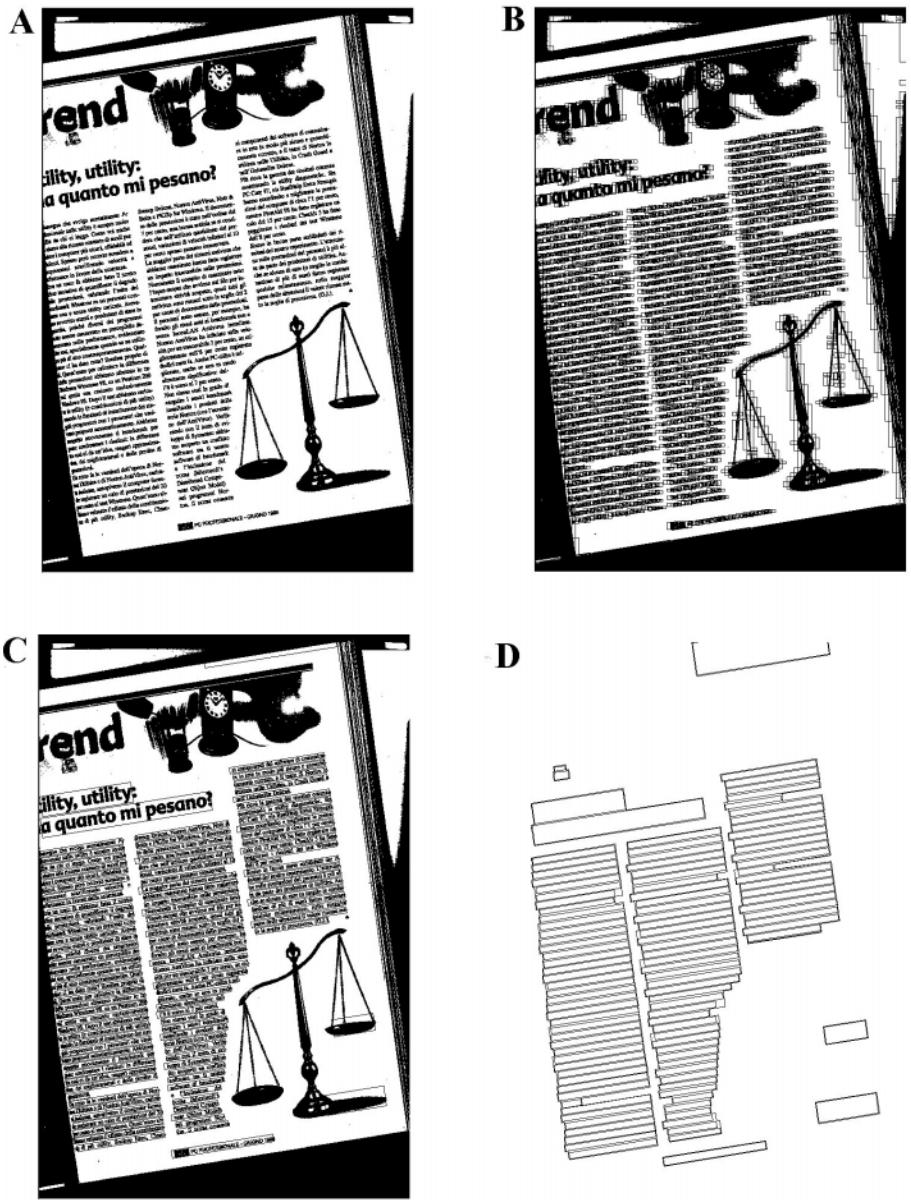

Fig. 4. A A document page scanned at 300 dpi. The skew is 10.8 degrees. There is text cropped around pictures, and several fonts are present. **B** The line elements extracted from each column, as in Step 2 of the algorithm. **C** Text lines extracted from the document, after Step 4 of the algorithm. **D** Text lines after gathering into blocks

times the font size ($\delta_{\text{B.L.SKIP}} = 2$ in the current implementation).

(iii) l_1 and l_2 overlap vertically: that is, $\delta_{\text{SPAN}} = 80\%$ of the x-span of either of them is contained into the x-span of the other.

Blocks are defined as the weakly connected components of this graph. Figures 2D and 3D show examples of blocks extraction. Apart from the geometrical representation, a block is just a set of text lines which must be read in a certain order. A unique ordering rule, which simply expresses the idea that text must be read from top to bottom and from left to right, is the following:

(i) Pick the text line with the lowest y-coordinate (we assume that y grows from top to bottom). If there is any ambiguity, pick the line with the lowest x-coordinate. This line is surely among the topmost lines of a block.

(ii) Assign a number to any node reachable from the topmost line, equal to the distance from the top line. We assume a unit cost for traversing a single arc of the graph. By construction, the number assigned to any node is unique.

(iii) If a node has been visited but any of its incident arcs has not been traversed, traverse it assigning a proper number to the nodes it connects: if we traverse it backward, we subtract 1, otherwise we add 1.

(iv) More than one node of the graph may be possibly assigned the same number n, indicating the distance from the top. In this case, we number all the nodes with the same n as (n, i) where i is assigned according to the x-coordinate of the beginning of the line. To simplify the graphical representation, lines with the same value of n are merged and only the rectangular hull enclosing them all is shown.

At the end of this procedure the directed graph can therefore be translated into an ordered list of text lines.

Output. The output is a set of blocks, each possibly containing a single line. Examples are provided in Figs. 3D and 4D. Blocks can be viewed as ordered lists of text lines and are the input to the OCR system.

4.1 List of parameters and thresholds

Table 1 lists the thresholds and parameters used throughout the algorithm.

4.2 Complexity analysis

We mention here some results on the sequential and parallel complexity of our algorithm, without getting into the details of how of their computation. The interested reader can find these details in [16]. The sequential complexity of the segmentation algorithm is $O((I_l I_w w + n_\lambda^2 I_w + n_\lambda w I_w)/\epsilon)$ where n_λ is the maximum number of text lines in a column. Since $I_{\mathrm{L}} = O(\rho)$ and $I_w = O(\rho)$ where ρ is the scanning resolution, we can otherwise express the overall complexity as $O((w\rho^2 + n_\lambda^2\rho + w n_\lambda \rho)/\epsilon)$.

The parallel complexity is $O(\log \rho)$ steps with $O((w\rho^2 + n_\lambda^2\rho + w n_\lambda \rho)/\epsilon)$ processors. The model of parallel computation for which this result was obtained is the widely used PRAM: parallel random access machine with shared memory, with exclusive-read, exclusive-write access priority. It is the weakest, and therefore most useful, of the several PRAM models which are used in literature [17].

The parallel version of our algorithm can easily be made optimal (a parallel algorithm is said to be *optimal* if the parallel time multiplied by the number of processors is equal to the time required by the sequential version of the algorithm) by applying Brent's principle of shifts of processors, thus obtaining $O(\log \rho)$ steps with $O((w\rho^2 + n_\lambda^2\rho + w n_\lambda \rho)/(\epsilon \log \rho))$ processors. The meaning of this result, for the practical case when the number of processors is P (possibly a very small number) is that the power of parallelism can be exploited fully. We can program the P processors so that the algorithm takes time $O((w\rho^2 + n_\lambda^2\rho + w n_\lambda \rho)/(\epsilon P))$, thus obtaining a speed-up factor equal to P.

These theoretical results on the complexity of our algorithm have been confirmed by experiments on actual time performance, where the dependence of execution time vs. resolution and all the relevant parameters of the algorithm (ϵ, w, n_λ) have been tested. The interested reader can find these results in [16].

5 Experimental results

The algorithm presented in this paper has been tested, among other sets of images, on the images of the UW English Document Database I (vol. 1, "Binary images") of the University of Washington [18], henceforth indicated as UW-I. This section analyzes the performance of our algorithm on this database based on an appropriate measure of segmentation accuracy (Sect. 5.1). Sections 5.2 and 5.3 show how the performance of our algorithm depends on the amount of noise and skew respectively.

5.1 Measuring segmentation accuracy

Segmentation accuracy is measured by a method inspired by the algorithm described in [19,20], but adapted to our type of output, which consists mainly of text lines. Therefore our method is textline-based, as opposed to text-based [21] or region-based [19,20]. The method works by (1) constructing the ground-truth lines associated with the document, and (2) comparing the ground-truth lines with the actual lines as put out by the text-extraction algorithm. Segmentation accuracy is estimated by computing the parameters listed in Table 2. These parameters are computed after the text-line extraction step.

For the UW-I database, the mean values of the parameters shown in Table 2 are shown in the *first row* of Table 3. The mean was performed on a sample of 100 binary images randomly selected from the database. The sample was drawn from the set of *binary* images scanned from the original journals and from photocopied journal pages (and *not* from the set of LaTeX files).

5.2 Accuracy vs. noise

The algorithm relies on the construction of line elements, which depends critically on the value of the threshold δ_{BP}. The value of δ_{BP}, which gives the threshold value for 'white' in the *horizontal* projection profile for a column (Step 2) and in the *vertical* projection profile of a line element (Step 4b) must depend on the amount of speckle noise, intended in the widest possible sense, as that kind of noise on account of which a white pixel has a finite probability P_{noise} to be mistaken as black. Speckle noise is usually small enough in documents scanned directly from the source or even in photocopied documents that a default value $\delta_{\mathrm{BP}} = 0$?can be used. However, to deal with more general cases we have set $\delta_{\mathrm{BP}} = P_{\mathrm{noise}}$. A heuristic but effective way of estimating P_{noise} automatically is that of randomly selecting a big number (1000 in our case) square regions of $I_w/100 \times I_w/100$ pixels, compute the percentage of black pixels for each of them and set P_{noise} equal to the smallest percentage found, which is likely to correspond to a region which does not contain anything except for speckle noise. Estimation of speckle noise carried out as above usually gives the correct percentage of noise with an error of up to about 2% of P_{noise}. For specially troublesome images the estimation can be carried out by selecting manually a region containing nothing except noise.

Table 3 (*second row*) shows the value of the segmentation accuracy parameters for a sample of 100 images from the UW-I database corrupted with 8% speckle noise. The main effect of noise is to increase the number of lines which are split.

Table 1. List of parameters and thresholds used in the algorithm, with a brief definition, their default values and some comments especially regarding the way they are chosen

Parameter	Brief description	Default value	Comments
ρ	resolution of scanned image	300 dpi	Chosen by the user when scanning the document, or given a priori if document is downloaded from a database
I_{L}	image length		See above
I_{W}	image width		See above
ϵ	column shift	1% I_{W}	Chosen by the user according to the implementation needs (e.g., segmentation precision, time complexity) and to the characteristics of the document (e.g., type of document, noise in document...).
$w^{'}$	column width	4% I_{W}	See above.
$\delta\theta$	Binwidth of skew histogram	0.1°	See above
δ_{BP}	Maximum density of black pixels for 'white' scanlines	0% or depending on noise level (Sect. 5.2)	See above
δ_{HIST}	width of peaks corresponding to a fixed font in the histogram of Step 3	$2\log 2$	Largely document – and implementation – independent. Tuned according to standard typographical usage. No need for further tuning except for very specific applications.
δ_{WHITE}	controlling extraction of white strips in Step 4b	2	See above
$\delta_{\mathrm{B/W}}$	controlling number of black-to-white transitions in the first classification test of Step 4g	2	See above
δ_{TILT}	controlling maximum relative skew of an individual text line in the second classification test of Step 4g	3	See above
δ_{MERGE}	Controlling merging of text lines in Step 4h	2	See above
$\delta_{\mathrm{B.L.SKIP}}$	Controlling block formation in Step 5ii	2	See above
δ_{SPAN}	Controlling block formation in Step 5iii	80%	See above

Table 2. The most relevant parameters for estimating segmentation accuracy, and the way they are computed

p_{miss}	Percentage of missed lines	Number of ground-truth lines (GTLs) not detected divided by the total number of GTLs. If for a given GTL only k words out of $k^{'}$ have been properly detected, the line counts as $(k^{'}-k)/k^{'}$.
p_{spur}	Percentage of spurious lines	Number of lines that do not correspond to any GTL, divided by the number of GTLs.
p_{split}	Percentage of split lines	Number of GTLs that have been merged (either horizontally or vertically) divided by the number of GTLs.
p_{merge}	Percentage of merged lines	Number of GTLs that have been split (either horizontally or vertically) divided by the number of GTLs.

Figure 5A shows the output of the segmentation algorithm for a document from the UW-I database corrupted with 10.1% of speckle noise. Text has been extracted after noise has been automatically estimated. As a typical example of the behavior of accuracy vs. noise, Fig. 6 shows $p_{\mathrm{miss}}, p_{\mathrm{spur}}, p_{\mathrm{split}}, p_{\mathrm{merge}}$ vs. P_{noise} for the same document, after the addition of varying amounts of noise. Notice that the behavior of the four parameters is at times irregular. This is first of all because noise addition is a random process, and some segmentation defects (e.g., the merging of two text lines) may or may not emerge depending on whether, e.g., a given pixel is corrupted in a specific realization of the noise-adding process. Furthermore, there are discretization effects (see for example Fig. 7, where random processes do not play a role), which are however less relevant.

5.3 Accuracy vs. skew

In Step 4e of the algorithm described in Sect. 4 skew is estimated with a precision of $\delta\theta = 0.1$ degrees, which is the binwidth of the skew histogram. In all cases which have been tested the estimated skew was in agreement with the skew directly measured from the image within the experimental precision of $\delta\theta = 0.1$ degrees. Successive processing on the document is always made in the new coordinate system of the unskewed document. This allows us to deal efficiently with skew up to about 10%. The algorithm breaks down when the skew angle is so

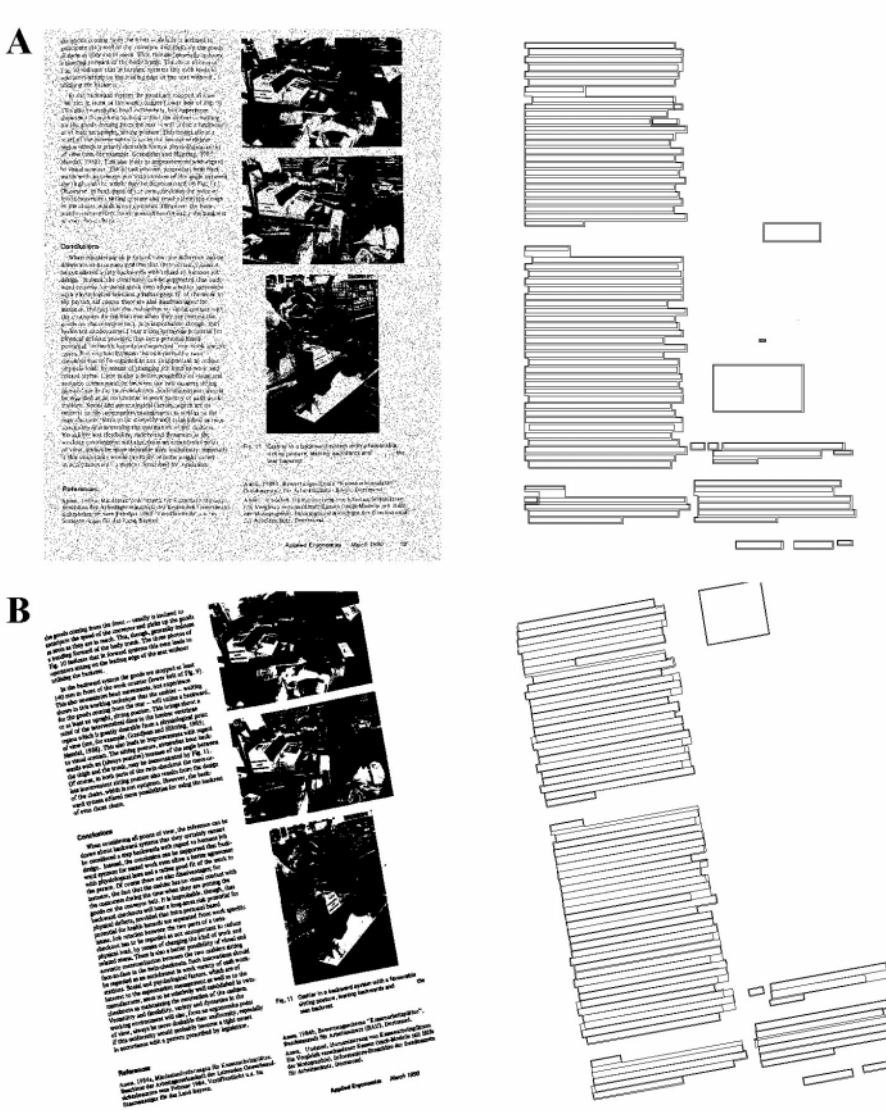

Fig. 5. Text extraction from a 300 dpi image drawn from the UW-I database (image K00obin), after **A** the document has been corrupted by speckle noise $P_{\text{noise}} = 10.1\%$, and **b** the document is skewed by 10 degrees. In **B** characters look thicker and some of the small white regions in the image have disappeared, on account of discretization effects after rotation and scale-reduction

high that the white vertical spaces between line elements disappear from the projection profile (see Sect. 6.2).

The behavior of the segmentation accuracy parameters for a sample of 100 images from the UW-I database is shown in Table 3 (third row), after rotating them by 10 degrees. Skew increases the number of missed lines, although it remains within acceptable values.

Figure 5B shows the output of the segmentation algorithm for a document from the UW-I database after rotation of 10 degrees. As a typical example of the behavior of accuracy vs. skew, Fig. 7 shows p_{miss}, p_{spur}, p_{split}, p_{merge} vs. skew for the same document. Notice that while the percentage of spurious lines does not exhibit a clear tendency to increase, the other parameters have a sudden increase at about 18 degrees: for that value, line elements start merging vertically in the projection profile. The decrease of p_{merge} for the last experimental point is due to the concurrent raise of the other parameters. The irregularities in the behavior of p_{spur} vs. skew depend on discretization effects.

Table 3. Mean values and standard error (standard deviation divided by the square root of the sample size) of the segmentation accuracy parameters for a sample set of 100 images from the UW-I database under different conditions (normal images, noisy images, skewed images)

	p_{miss}	p_{spur}	p_{split}	p_{merge}
Sample of images from the UW-I database	1.2 ± 0.2	3.0 ± 0.4	2.0 ± 0.4	0.3 ± 0.1
Same sample, each image with 8% speckle noise	1.3 ± 0.2	4.0 ± 0.7	7.0 ± 1.3	0.8 ± 0.1
Same sample, each image with 10 degrees of skew	2.2 ± 0.6	5.2 ± 1.5	2.0 ± 0.4	0.3 ± 0.1

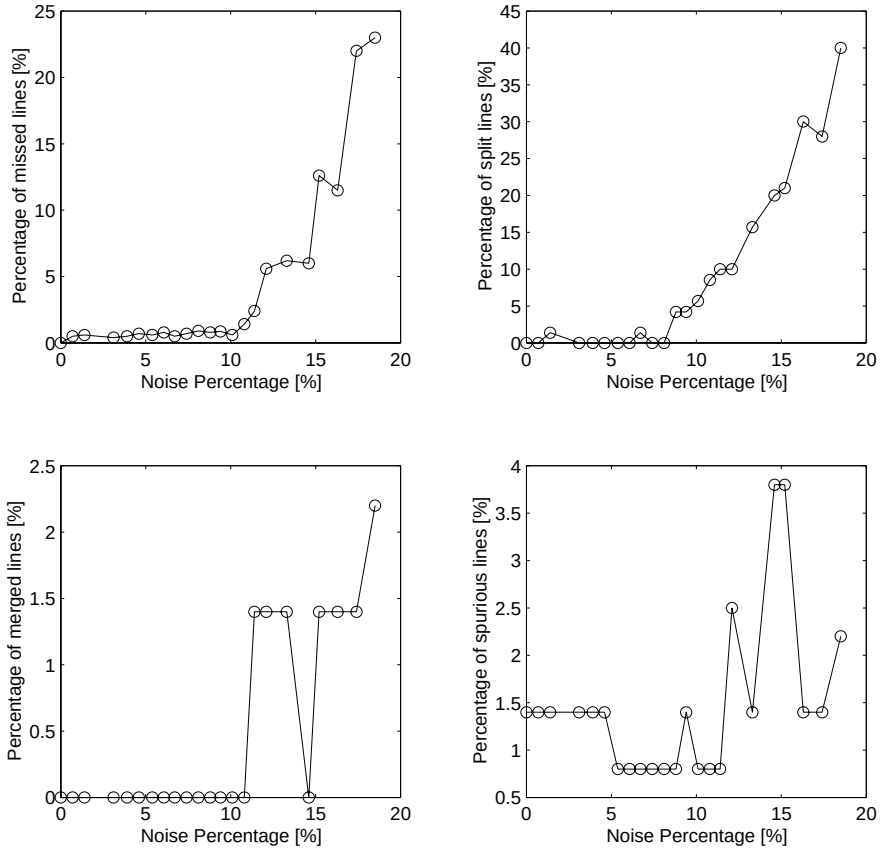

Fig. 6. Segmentation accuracy parameters as a function of speckle noise for the document shown in Fig. 4 (image K00obin of the UW-I database)

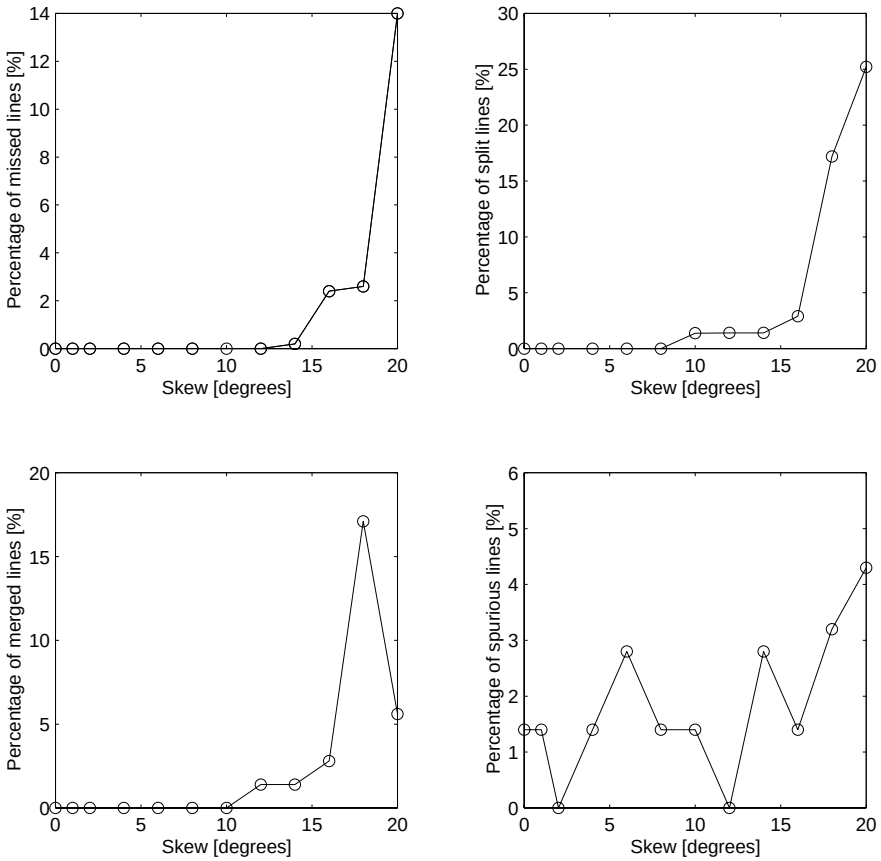

Fig. 7. Segmentation accuracy parameters as a function of skew for the document shown in Fig. 4 (image K00obin of the UW-I database)

6 Discussion

In this section we discuss strengths (Sect. 6.1) and weaknesses (Sect. 6.2) of the text-extraction algorithm presented here, and we compare it to closely related works (Sect. 6.3).

6.1 Strengths of the proposed methods

Efficiency. The algorithm is very fast. On the whole set of images scanned at 300 dpi of the UW-I database, the average running time is $1.33 \pm 0.06s$ per image on a PC with a Pentium II CPU (300 MHz), MotherBoard Intel 440LX, which is far less than the time taken by I/O operations, which demand an additional 7.2s as an average.

Flexibility. The algorithm is general-purpose, in the sense that it does not assume a specific layout of the page, nor does it assume that the page can be decomposed into rectangular blocks, or that the text is disjoint from graphics and pictures. The strongest requirement made on the layout is that all text be oriented in the same direction.

Local vs. global thresholds. Most methods for page segmentation rely on the use of appropriate thresholds. Just to name a few examples:

- In top-down methods using recursive X-Y cuts, columns are identified by detecting the white space between columns. If this space is greater than a certain threshold δ (proportional to resolution) a separation is created. Inside columns, paragraphs, lines and so on are separated according to new thresholds.
- Some methods (especially bottom-up methods) use run-length smearing as a pre-processing stage to connected component analysis. This reduces the computational time and smoothes the image. Two black run lengths are joined if the white space between them is less than a certain threshold δ', dependent on resolution.

Generally speaking, in bottom-up methods there are several levels at which clustering of features takes place, and each level has its own thresholds; similarly, in top-down methods there are several levels of subdivision and each level has its own thresholds.

The resort to thresholds is to a certain extent inevitable, because any modelization of text must be based on a certain number of parameters: font size, intercharacter distance, interword distance, base line skip, height of ascendants and descendants, column separation, and more. Most of these parameters, however, are roughly proportional to the font size. One problem is that these parameters are local, and not a property of the whole page. Many authors *compute* some of the parameters to obtain, for example, estimates for run-length smearing, but these estimates usually refer to the whole page, which is not appropriate when regions of text with different fonts are present on the page.

The algorithm described in this paper uses several thresholds. All of them, however, are derived from standard *typographical usage* and are fairly robust. What is more important, the thresholds have been determined *locally*, as a function of the (locally) estimated font size (see [22] for another case where local thresholds are used).

Our system also introduces two parameters, ϵ and w, which are related respectively to the segmentation precision and to the minimum width of text which is guaranteed – under ideal conditions – to be detected. These two parameters can be decided by the user according to the implementation needs. The algorithm, however, is not very sensitive to the choice of these two parameters, and $\epsilon = 1\%$, $w = 4\%$ of the image width are good default values for a wide range of documents, from dense magazine pages to postcards.

Robustness against skew. The algorithm is robust against skew: it can safely handle images with up to about 10 degrees of skew (see Sect. 6.2 for more detailed considerations). Notice that skew due to careless scanning or photocopying is usually no more than 2–3 degrees. This robustness against skew is due to the fact that the document is subdivided into small overlapping columns. In each individual column, skew does not have a strong impact. Skew is then estimated at the moment of the formation of text lines. All the subsequent operations are performed on the new system of coordinates according to which the document is unskewed. Notice that the presence of skew does not require pre-processing: skewed and unskewed documents are treated in the same manner. Time performance is not affected by skew [16].

6.2 Weaknesses and breaking points

In this section we describe the weaknesses of our algorithm, and the points where the algorithm breaks:

- The algorithm only detects black text on white background. Adaptation to the case of non-white but homogeneous background should be easy. However, when background is not homogeneous, some text will certainly be lost.
- The algorithm assumes that all text is oriented in the same direction, which is by default horizontal. If the main body of the text is horizontal (apart from skew), the method will not find vertical or oblique text. This makes the algorithm not suitable to deal with documents containing both vertical and horizontal text, such as is typically the case with, e.g., Japanese documents. This is a disadvantage of our method with respect to some methods developed in the past [23,24] which can deal with this problem.
- For the algorithm to work properly, inter-line gap must be greater than zero. When there is no vertical space between two text lines (which sometimes happens when specific typographical effects are pursued, as in commercial ads) the projection profile will fail to discriminate the line elements, undermining text lines reconstruction.

– The algorithm is not guaranteed to work properly when skew exceeds a certain threshold. For standard journal papers, this threshold is usually above 10 degrees (see the example in Fig. 4). The precise location of the breaking point depends on the specific document: for a homogeneous region of text, the algorithm will start having problems when $\tan(\theta^*) \geq ILG/w'$, where θ^* is the skew angle, ILG is the inter-line gap and w' is the column width. After that, the algorithm will start behaving as if there were no inter-line gap at all (see discussion above): the projection profile will start merging line elements vertically, undermining text lines reconstruction.
– Under ordinary conditions (standard journal papers), the algorithm will start giving unreliable results (about 5% of missed lines) when the probability of mistaking a white dot for a black one is 10–15% (speckle noise).

6.3 Comparison to related approaches

In this section we compare our method to the other methods based on background analysis, that is, methods which use white streams as a basis for image segmentation. First of all, observe that a distinction should be made: whereas in [2] and in [4,27] the white streams are themselves the basic units for the successive steps of the segmentation procedure, and they are aggregated so as to encircle the regions of text, in [3] and in this work the non-white intervals (column intervals and line elements, respectively) rather than the white streams are used as the basic units which are merged to give text regions. In the latter case we have therefore *foreground aggregation* rather than *background aggregation* [25].

The work most related to ours is certainly [3]. The key-idea in both [3] and our work is that the problems caused by the complexity of the layout and by the skew can be circumvented if one subdivides the document into small stripes (which are horizontal in [3], vertical in our work) and then assembles the information gathered in the different stripes. Despite this formal analogy, however, there are crucial differences between the two methods. First, the purpose of analyzing thin stripes of the document is different: [3] aims at building up the text columns, we aim at building up the text lines directly. Second, in our case the stripes overlap, which allows us to control segmentation precision. In [3], text columns are found with a segmentation precision which coincides with the height of the horizontal stripes. It is possible to increase precision by decreasing the height of the stripes, but this strategy cannot be pushed too far: each stripe must be high enough to be "representative" of the desired level of detail, and not to be exposed to wild variations due to details in the image at a smaller scale than that we are interested in. In our case, the overlapping between columns allows the decoupling of the two problems, precision and representativeness.

These are the differences which are specific to the comparison between our work and [3]. As regards other differences, notice that, unlike [3] and [4,27] – which use run-length smearing with a smearing value computed for the whole page – in our work all thresholds are computed locally and this allows us to deal with genuinely multi-font pages. Finally, an advantage of our work is that it does not require that text be surrounded by streams of white space, either straight [2,3] or arbitrary [4,27]. However, observe that although not required by our algorithm, this is still a desirable property: when a text region is surrounded by a white stream of sufficient width (one or two times the size of a character), the reconstruction precision becomes exactly 1 pixel and not ϵ as in the general case.

Acknowledgements. The authors would like to thank Giulia Piccioli, Apostolos Antonacopoulos and Giorgio Musso for fruitful discussion. We would also like to thank Fernando Nuflo and Riccardo Brancaleon for their technical help. Part of this work was carried out while Pietro Parodi was at the Department of Computer Science of the University of Toronto, partially funded by the IBM Center of Advanced Studies in Toronto.

References

1. L. O'Gormam, R. Kasturi.: Document Image Analysis. IEEE Computer Society, 1995
2. W.S. Baird, S. E. Jones, S. J. Fortune.: Image segmentation by shape-directed covers. In: Proc. of ICPR, pp. 820–825, 1990
3. T. Pavlidis, J. Zhou.: Page segmentation and classification. CVGIP 54:484-496, 1992
4. A. Antonacopoulos, R. T. Ritchings.: Flexible page segmentation using the background. In: Proc. of ICPR, 1994
5. L.A. Fletcher, R. Kasturi.: A robust algorithm for text string separation from mixed text/graphics images. IEEE Transact PAMI 10(6):910–918, 1988
6. J.L. Fisher, S.C. Hinds, D.P. D'Amato.: A rule-based system for document image segmentation. In: Proc. of ICPR, pp. 567–572, 1990
7. L. O'Gorman.: The document spectrum for page layout analysis. IEEE Transact PAMI 15(1l):1162–1173, 1993
8. A. K. Jain, B. Yu.: Document representation and its application to page decomposition. IEEE Transact Pattern Analysis and Machine Intelligence 20(3), 1998
9. K.Y. Wong, R.G. Casey, F.M. Wahl.: Document analysis system. I.B.M. J Res Dev 26(6):647–656, 1982
10. D. Wang, S.N. Srihari.: Classification of newspaper image blocks using texture analysis. CVGIP 47:327–352, 1989
11. F.M. Wahl, K.Y.Wong, R.G. Casey.: Block segmentation and text extraction in mixed text/image documents. CVGIP 20:375–390, 1982
12. G. Nagy, S.C.Seth.: Hierarchical representation of optical scanned documents. In: Proc. of ICPR, pp. 347–349, 1984
13. A.K. Jain, S. Bhattacharjee.: Text segmentation using Gabor filters for automatic document processing. Machine Vision and Applications 5:169–184, 1992
14. K.Etemad, R. Chellappa, D. Doermann.: Page segmentation using wavelet packets and decision integration. In: Proc. of ICPR, pp. 345–349, 1994

15. Y.Y. Tang, H. Ma, X. Mao, D. Liu, C.Y. Suen.: A new approach to document analysis based on modified fractal signature. In: Proc. of ICDAR, Montreal, Canada, 1995
16. P. Parodi, G. Piccioli.: An efficient and flexible method for text extraction in document pages. Technical Report RBCV-TR-96-51, University of Toronto, Canada, 1996
17. J. JêJê.: An introduction to parallel algorithms. Reading, MA:Addison-Wesley, 1992
18. S. Chen, M. Y. Jaisimha, J. Ha, R. M. Haralick.: Reference manual for the UW English Document Database I, 1993
19. B. A. Yanikoglu, L. Vincent.: Ground-truthing and benchmarking document page segmentation. In: Proc. of ICDAR, Montreal, Canada, 1995
20. B. A. Yanikoglu, L. Vincent.: Pink Panther: A complete environment for ground-truthing and benchmarking document page segmentation. Manuscript, 1996
21. J. Kanai, S. V. Rice, T. Nartker, G. Nagy.: Performance metrics for document understanding systems. In: Proc.ICDAR, Tsukuba, Japan, 1993
22. D. Sylwester, S. Seth.: A trainable, single-pass algorithm for column segmentation. In: Proc. of ICDAR, Montreal, Canada 1995
23. N. Amamoto, S. Torigoe, Y. Hirogaki.: Block segmentation and text area extraction of vertically/horizontally written document. In: Proc. of ICDAR, Tsukuba, Japan, 1993, pp. 336–340
24. Y. Ishitani.: Document layout analysis based on emergent computation. In: Proc. of ICDAR 97, pp. 45–50
25. A. Antonacopoulos.: Personal communication.
26. D.S. Le, G.R. Thoma, H. Wechsler.: Automated page orientation and skew angle detection for binary document images. Pattern Recognition 27(10):1325–1344, 1994
27. A.Antonacopoulos.: Page segmentation using the description of the Bbackground. Computer Vision and Image Understanding, Special Issue on Document Image Understanding and Retrieval 70(3), 1998, pp. 350–369