Simple Algorithm for Page Layout Analysis

A. O. Shigarov and R. K. Fedorov

Institute for System Dynamics and Control Theory, Siberian Branch, Russian Academy of Sciences, ul. Lermontova 134, Irkutsk, 664033 Russia

Abstract—An algorithm for page layout analysis (segmentation) is suggested in the paper. It allows whitespace between text blocks to be detected on a document page. The algorithm could be used in document analysis and recognition problems. In particular, it can be used for column recognition in multicolumn text and tables. The suggested algorithm is quite simple for implementation.

INTRODUCTION

The number of documents in the world is growing rapidly; this is discussed in [6] in particular. Document analysis and recognition (DAR) systems are developed for automation of information extraction from document images [7]. Document layout analysis or page segmentation is used for dividing document into specific parts (e.g., columns, figures, tables). This is an important DAR problem. Different methods of document layout analysis are discussed in [3].

Conventionally, there are two approaches to the problem of page or table column segmentation. The first approach is to analyze of text layout (text blocks); this usually requires the use of complex data structures. For example, such an approach with the use of a Voronoi diagram for page segmentation is suggested in [5]. The second approach is to analyze whitespace (free of text) on a page. Gaps separate text and table columns on a page, as is shown in Fig. 1. Algorithms using whitespace analysis are suggested in [1, 2, 8]. Algorithms [1, 8] are briefly considered in [2]. They can be used for detecting whitespace between text blocks. The authors of [2] point out that the algorithms [1, 8] are difficult for implementation. A geometrical layout analysis algorithm, simple for implementation, is presented in [2]; it provides for detecting whitespace gaps in multicolumn text and is described in terms of the largest empty rectangle problem [4]. The algorithm input is a bounding box including obstacles (rectangles). The algorithm [2] finds the largest empty rectangle among obstacles inside the bounding box. To find whitespace gaps on a page, n-better solutions (empty rectangles) in descending order are found on the page. The algorithm [2] is greedy. The found i-largest empty rectangle becomes an obstacle when searching for the next i + 1-largest empty rectangle. However, it is probable that some of n-better solutions (largest rectangles found) are not gaps between columns, but, e.g., horizontal gaps between paragraphs or tables on the page. This is the disadvantage of the algorithm [2].

In the paper, we suggest an unconventional simple algorithm for detecting whitespace gaps on a document page. It allows detecting vertical gaps (visually they are stretched up vertically), as well as horizontal gaps if the X and Y axis are exchanged. The suggested algorithm is simple; its Object Pascal implementation consists of about 60 lines of code (expressions).

1. PROBLEM FORMULATION

The geometrical objects considered in this work are presented in the Cartesian coordinate system, where the x-coordinates increase from left to right and y-coordinates increase from the top down. The following assumptions are used in this work. The rectangle (e.g., an obstacle, bounding box, or gap) $r = (x_l, y_t, x_r, y_b)$ is defined by the coordinates of its sides (boundaries): left $c_l = x_l(r)$, top $y_t = y_t(r)$, right $x_r = x_r(r)$, and bottom $y_b = y_b(r)$; in addition, its sides are parallel to the corresponding coordinate axes. The vertical line $l = (x, y_t, y_b)$ is normal to the X axis and is defined by its coordinates: $x = x(l)$ by the x-coordinate, $y_t = y_t(l)$ by the minimum (top) y-coordinate, and $y_b = y_b(l)$ by the maximum (bottom) y-coordinate.

It is assumed that the bounding rectangle b and the set of obstacles (rectangles) $R = \{r_1, ..., r_n\}$, $n \in \mathbb{N}$ are specified. The rectangle b usually bounds a document page or its part (e.g., a table), while the obstacles are bounding rectangles for text blocks (e.g., words or lines). The obstacles from the set R are totally inside the bounding rectangle b and do not overlap each other.

Let us define two sets of obstacles (rectangles) $R' = \{r_l, r_1, ..., r_n, r_r\}$ and $R'' = \{r_t, r_1, ..., r_n, r_b\}$, where $r_l = (x_l, y_t, x_l, y_b)$, $r_r = (x_r, y_t, x_r, y_b)$, $r_t = (x_l, y_t, x_r, y_t)$, $r_b = (x_l, y_b, x_r, y_b)$, $x_l = x_l(b)$, $y_t = y_t(b)$, $x_r = x_r(b)$, and $y_b = y_b(b)$. Let us also define the gap as a rectangle bounding a certain part of whitespace inside the bounding rectangle b.

If any two rectangles r and $\tilde{r}$ from the set R' satisfy the following conditions,

(1) their projections to the X axis do not intersect, i.e.,

$$x_r(r) < x_l(\tilde{r}), \quad (1)$$

(2) there is no other rectangle of the set R between them, i.e.,

$$\nexists \bar{r}: \bar{r} \in R, \bar{r} \neq r, \tilde{r} \text{ and } \bar{r} \subset w, \text{ where}$$
$$w = (x_r(r), \min\{y_t(r), y_t(\tilde{r})\}, \quad (2)$$
$$x_l(\tilde{r}), \max\{y_b(r), y_b(\tilde{r})\}),$$

then the rectangles r and $\tilde{r}$ should be separated by a gap. The state problem consists in separation of all rectangles of the set R', satisfying conditions (1) and (2), by a minimum number of gaps.

2. ALGORITHM

The algorithm consists of two steps. The following actions are executed for each rectangle at the first step:

(1) The first line (or rule) is extended from the left boundary of the obstacle r: $r \in R$ upward and downward, while it is hampered from top and bottom by another obstacle $\tilde{r}$: $\tilde{r} \in R$, $\tilde{r} \neq r$ or the bounding rectangle b, as is shown in Fig. 2a. In this case, the resulting line is added to the set L.

(2) The second line (or rule) is extended from the right obstacle boundary r: $r \in R$ by analogy with the first case. In this case, each resulting line is added to the set $\tilde{L}$.

Pairs of lines $(\tilde{l}, l)$ are formed at the second step; the first line $\tilde{l}$ in each pair either belongs to the set $\tilde{L}$ or is the left side of the bounding rectangle b, and the second line l belongs to the set L or is the right side of the bounding rectangle b. Each pair of lines $(\tilde{l}, l)$ satisfies the condition

$$y_t(l) = y_t(\tilde{l}) \text{ and } y_b(l) = y_b(\tilde{l}); \quad (3)$$

i.e., their y-coordinates are equal. In addition, there is no obstacle between the lines $\tilde{l}$ and l of each pair, i.e.,

$$\nexists r: r \in R \text{ and } r \subset w, \text{ where}$$
$$w = (x(\tilde{l}), y_t(l), x(l), y_b(l)). \quad (4)$$

Each such a pair of lines $(\tilde{l}, l)$ forms a gap, as is shown in Fig. 2b. The set of such gaps $g = (x_l(\tilde{l}), y_t(l), x_r(l), x_b(l))$ is the algorithm output.

Below the algorithm is represented as a pseudocode. It is assumed that all sets are represented as lists. The function SORT(S, (c_1, c_2)) sorts elements (e.g., lines, rectangles) in the list S lexicographically in (c_1, c_2)-coordinates ascending order; i.e., any two elements s, $\tilde{s}$ of the list S satisfy the condition $s < \tilde{s}$, if either $c_1(s) < c_1(\tilde{s})$ or $c_1(s) = c_1(\tilde{s})$ and $c_2(s) < c_2(\tilde{s})$. An obstacle is formed in line 01 with the help of the upper boundary of a bounding rectangle (in line 02).

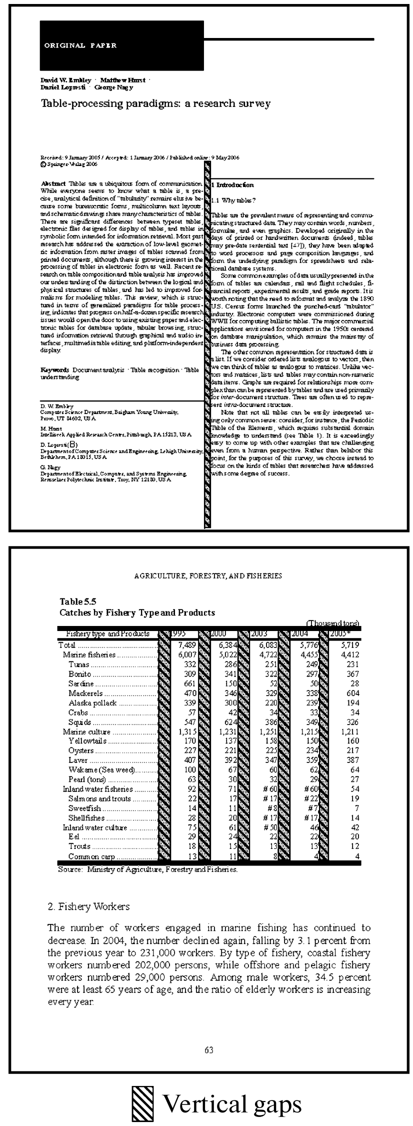

Fig. 1. Whitespace gaps separate columns in multicolumn text and a table.

The obstacles r_0 and r_{n+1} are added to the list R (in line 03). The obstacles in the list R are lexicographically (y_t, x_l)-coordinate ascending ordered from the top down and from left to right (line 04). Obstacles of the list R are sorted out from the top down and from left to right in lines 05–22 and from right to left and from the bottom upward in lines 07–11. In this case, lines are extended upward from the left and right boundaries of the current rectangle. Obstacles of the list R are sorted out from the top down and from left to right in lines 12–16, and lines from the left and right

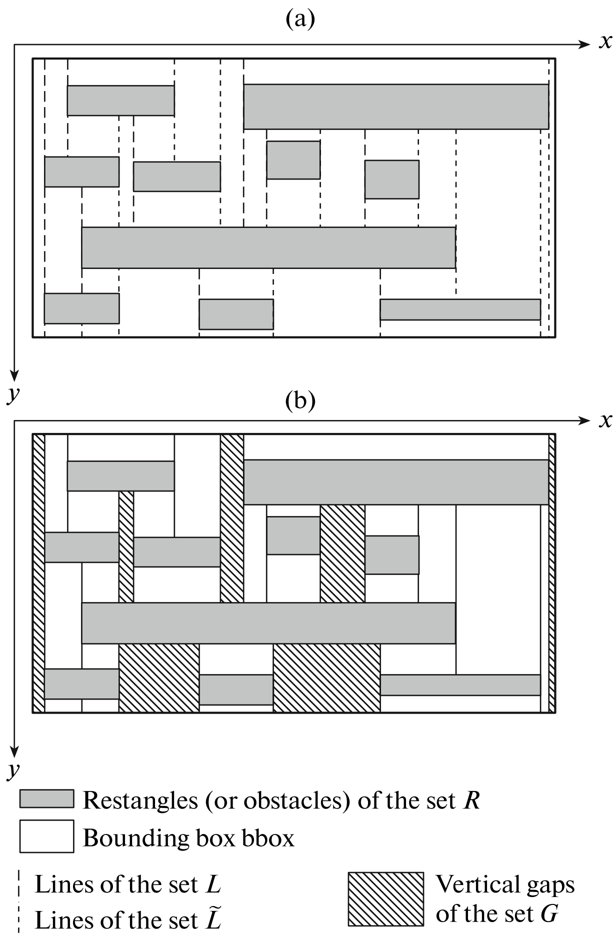

Fig. 2. Detecting whitespace gaps.

boundaries of the current rectangle are extended downward. If the following nonempty sets of lines

$$L = \{l_1, \ldots, l_p\}, \quad \tilde{L} = \{\tilde{l}_1, \ldots, \tilde{l}_s\}, \quad p, s \in \mathbb{N}$$

are obtained after executing step 1, then step 2 is executed. The line $\tilde{l}$ is formed with the help of the left boundary of the bounding rectangle (line 23), and line l, with the help of the right boundary (line 24). The line $\tilde{l}$ is added to the list $\tilde{L}$ (line 25), and the line l, to the list L (line 26). Lines in the lists L and $\tilde{L}$ are lexicographically (x, y_t)-coordinate ascending ordered from left to right and from the top down in lines 27–28. Lines of the list $\tilde{L}$ are sorted out from left to right and from the top down in lines 29–34. Lines of list L are sorted out from left to right and from the top down in lines 30–34. Then, the lines satisfying conditions (3) and (4) are selected and gaps are formed.

The pseudocode of the first algorithm step is presented below.

Input: b, $R = \{r_1, \ldots, r_n\}$, $n \in \mathbb{N}$

Output: $G = \{g_1, \ldots, g_m\}$, $m \in \mathbb{N}$

01 **create** $r_0 \longleftarrow (x_l(b), y_t(b), x_r(b), y_t(b))$
02 **create** $r_{n+1} \longleftarrow (x_l(b), y_b(b), x_r(b), y_b(b))$
03 **add** r_0, r_{n+1} **to** R
04 SORT(R, (y_t, x_l))
05 **for** $i = \overrightarrow{1, n}$ **do**
06 $\overline{y}_l, \overline{y}_r, \underline{y}_l, \underline{y}_r \longleftarrow$ **null**
07 **for** $j = \overrightarrow{i-1, 0}$ **do**
08 **if** $y_t(r_i) > y_b(r_j)$ **then**
09 **if** $\overline{y}_l =$ **null**
and $x_l(r_j) < x_l(r_i) < x_r(r_j)$
then $\overline{y}_l \longleftarrow y_b(r_j)$
10 **if** $\overline{y}_r =$ **null**
and $x_l(r_j) < x_r(r_i) < x_r(r_j)$
then $\overline{y}_r \longleftarrow y_b(r_j)$
11 **if** $\overline{y}_l, \overline{y}_r \neq$ **null**
then exit for j
12 **for** $j = \overrightarrow{i+1, n+1}$ **do**
13 **if** $y_b(r_i) < y_t(r_j)$ **then**
14 **if** $\underline{y}_l =$ **null**
and $x_l(r_j) < x_l(r_i) < x_r(r_j)$
then $\underline{y}_l \longleftarrow y_t(r_j)$
15 **if** $\underline{y}_r =$ **null**
and $x_l(r_j) < x_r(r_i) < x_r(r_j)$
then $\underline{y}_r \longleftarrow y_t(r_j)$
16 **if** $\underline{y}_l, \underline{y}_r \neq$ **null**
then exit for j
17 **if not** L **contains** $(x_l(r_i), \overline{y}_l, \underline{y}_l)$
then
18 **create** $l \longleftarrow (x_l(r_i), \overline{y}_l, \underline{y}_l)$
19 **add** l **to** L
20 **if not** $\tilde{L}$ **contains** $(x_r(r_i), \overline{y}_r, \underline{y}_r)$
then
21 **create** $\tilde{l} \longleftarrow (x_r(r_i), \overline{y}_r, \underline{y}_r)$
22 **add** $\tilde{l}$ **to** $\tilde{L}$

The pseudocode of the second algorithm step is presented below.

23 **create** $\tilde{l} \longleftarrow (x_l(b), y_t(b), y_b(b))$
24 **create** $l \longleftarrow (x_r(b), y_t(b), y_b(b))$
25 **add** $\tilde{l}$ **to** $\tilde{L}$
26 **add** l **to** L

SORT($L(x, y_t)$)

SORT($\tilde{L}, (x, y_t)$)

for $i = \overrightarrow{1, s+1}$ **do**

  for $j = \overrightarrow{1, p+1}$ **do**

    if $x(\tilde{l}_i) < x(l_j)$ **and** $\bar{y}(\tilde{l}_i) = \bar{y}(l_j)$

    and $\underline{y}(\tilde{l}_i) = \underline{y}(l_j)$ **then**

      create $g \longleftarrow (x(\tilde{l}_i), \bar{y}(\tilde{l}_i), x(l_j)\underline{y}(l_j))$

      add g **to** G.

CONCLUSIONS

The suggested algorithm could be used for document page segmentation. For example, it can reveal columns inside multicolumn text and tables. The algorithm is used in the method for detecting tables in a document, suggested in [9]. The algorithm is quite simple; its computational complexity is $O(n^2)$.

ACKNOWLEDGMENTS

This work was supported by RFBR (11-07-00426-a), the grant of the RF President (MK-3592.2010.8), and the Lavrent'ev grants for youth projects of SBRAS.

REFERENCES

1. H. S. Baird, S. E. Jones, and S. J. Fortune, "Image Segmentation by Shape–Directed Covers," in *Proc. Int. Conf. on Pattern Recognition* (Atlantic City, 1990), Vol. 1, pp. 820–825.
2. T. M. Breuel, "Two Geometric Algorithms for Layout Analysis," in *Proc. 5th Int. Workshop on Document Analysis Systems* (Nara, 2008), Vol. 2423, pp. 188–199.
3. R. Cattoni, T. Coianiz, S. Messelodi, and C. M. Modena, "Geometric Layout Analysis Techniques for Document Image Understanding: a Review," Tech. Rep. IRST (Trento, 1998).
4. J. Chaudhuri, S. C. Nandy, and S. Das, "Largest Empty Rectangle among a Point Set," J. Algorithms **46** (1), 54–78 (2003).
5. K. Kise, A. Sato, and M. Iwata, "Segmentation of Page Images Using the Area Voronoi Diagram," Comp. Vision Image Understand. **70**, No. 3, 370–382 (1998).
6. P. Lyman and H. R. Varian, "How much Information?," Tech. Rep. (2003), Available from: http://www.sims.berkeley.edu/how-much-info-2003
7. *Machine Learning in Document Analysis and Recognition,* Ed. by S. Marinai and H. Fujisawa (2008), Vol. 90.
8. M. Orlowski, "A New Algorithm for the Largest Empty Rectangle Problem," Algorithm. **5**, Nos. 1–4, 65–73 (1990).
9. A. O. Shigarov, I. V. Bychkov, G. M. Ruzhnikov, and A. E. Khmel'nov, "A Method for Table Detection in Metafiles," Pattern Recogn. Image Anal. **19**, No. 4, 693–697 (2009).