# An Algorithm for Page Segmentation

**Conference Paper** · December 2010

CITATIONS
0

READS
90

**2 authors**, including:

Alexey Shigarov
Institute for System Dynamics and Control Theory of Siberian Branch of Russian Academy of Sciences (ISDCT SB RAS)
**31** PUBLICATIONS **36** CITATIONS

SEE PROFILE

# SIMPLE ALGORITHM FOR DOCUMENT LAYOUT ANALYSIS

**A.O. Shigarov, R.K. Fedorov[1]**

**[1] Institute for System Dynamics and Control Theory, Siberian Branch of RAS, 664033, Lermontov st., 134, Irkutsk, Russia, tel. +7-3952-45-31-02, e-mail: shigarov@icc.ru**

In the paper the algorithm for page segmentation (document layout analysis) is proposed. The algorithm provides detecting whitespace gaps located between text blocks on a document page. The algorithm can be used for tasks of document analysis and recognition. In particular, it can be used for column detection in tables and multi-column texts. Its pseudo code is also presented in the paper. The proposed algorithm is sufficient simple for implementation.

## Introduction

The number of documents in the world grows all the time and rapidly. It is particularly discussed in the paper [6]. Document analysis and recognition (DAR) systems are developed for automation of information extraction from document images [7]. Document layout analysis or page segmentation is dividing document into parts being worth (e.g. columns, figures, tables). It is the important task of DAR. Variety of techniques for document layout analysis is discussed in the paper [3].

Conditionally there are two approaches to the problem of page or table segmentation into columns. The first approach is to analyze text layout (text blocks). It requires using complex data structures. For example, such approach, using the Voronoi diagram for page segmentation, is proposed in the paper [5]. The second approach is to use page whitespace analysis. In a page whitespace gaps divide text or table columns, as shown in Fig. 1. The algorithms, used whitespace analysis, are proposed in the papers [1, 2, 8]. In the paper [2] the algorithms [1, 8] are shortly considered. They can be used for detecting whitespace gaps between text blocks. The authors of [2] are pointed out, that algorithms [1, 8] are fairly difficult to implement. A geometric algorithm for whitespace cover is presented in the paper [2]. It provides detecting whitespace gaps on document page. The algorithm [2] is described in terms of the largest empty rectangle problem [4]. Its input is a bounding box containing obstacles (rectangles). The algorithm [2] searches a largest empty rectangle, which contains inside of the

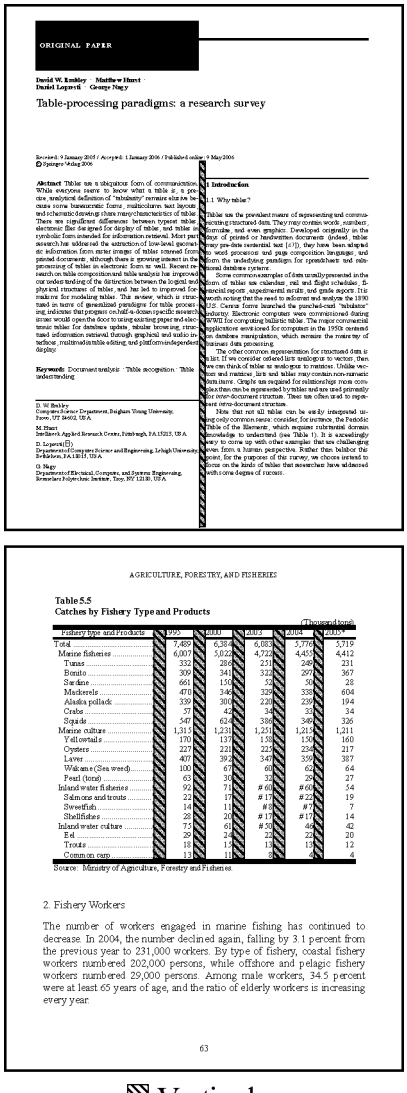


Fig. 1. Using whitespace gaps for dividing columns in multi-column texts and tables.

bounding box between the obstacles. For detecting whitespace gaps on a page it is searched *n*-best solutions (i.e. empty rectangles) in descending order. Furthermore, the algorithm [2] is greedy. After a largest empty rectangle has been found, it becomes the obstacle for searching a next largest empty rectangle. However, it is possible certain of the n-best solutions (detected largest whitespace rectangles) will not be whitespace gaps between columns. For example, a largest whitespace rectangle can be horizontally located on a page between paragraphs or tables. It is the disadvantage of the algorithm [2].

In the paper the original algorithm for detecting whitespace gaps on a document page is proposed. The algorithm allows detecting vertically situated gaps (i.e. visually they are vertically stretched). Also, the algorithm can be used for detecting horizontally situated gaps if interchange X- and Y-axis. The proposed algorithm is simple. Its implementation by Object Pascal includes about 60 lines (statements).

## Problem Formulation

Considered in the paper geometric objects are presented by the rectangular coordinate system. In the system, x-coordinates increase from left to right, and y-coordinates increase from top to bottom. In the paper, following propositions are used. A rectangle (e.g. an obstacle, a bounding box and a gap) $r=(x_l, y_t, x_r, y_b)$ is defined by the coordinates of its sides (or bounds), as following: $x_l = x_l(r)$ is left, $y_t = y_t(r)$ is top, $x_r = x_r(r)$ is right, and $y_b = y_b(r)$ is bottom. Moreover, its sides are parallel to the corresponding coordinate axes. A vertical line $l=(x, y_t, y_b)$ is orthogonal to the X-axis, and is defined by the coordinates, as following: $x = x(l)$ is the x-coordinate, $y_t = y_t(l)$ is the minimal (top) y-coordinate, and $y_b = y_b(l)$ is the maximal (bottom) y-coordinate.

It is assumed that we are given a bounding box *b*, and a set of rectangles (or obstacles) $R=\{r_1,\ldots,r_n\}, n\in\square_0$. Usually this bounding box bounds a document page or a part of the page, and these rectangles are the bounding boxes of text blocks (or words). Moreover, the rectangles of the set *R* are placed within the bounding box *b* without overlapping each other.

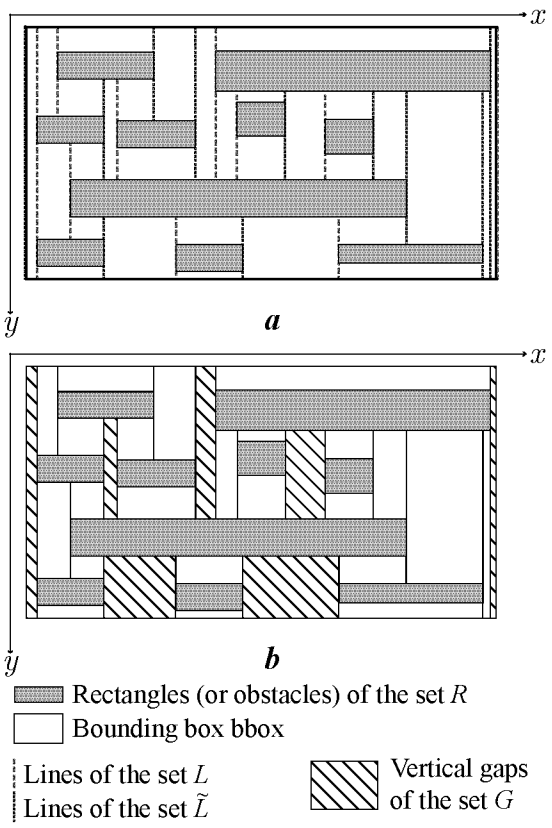


Fig. 2. Detecting whitespace gaps.

Define two sets of rectangles (or obstacles) $R'=\{r_l, r_1,\ldots,r_n, r_r\}$ and $R''=\{r_t, r_1,\ldots,r_n, r_b\}$, where $r_l=(x_l, y_t, x_l, y_b)$, $r_r=(x_r, y_t, x_r, y_b)$, $r_t=(x_l, y_t, x_r, y_t)$, and $r_b=(x_l, y_b, x_r, y_b)$, with $x_l = x_l(b)$, $y_t = y_t(b)$, $x_r = x_r(b)$, and $y_b = y_b(b)$. Also define that a whitespace gap is a rectangle bounding some part of whitespace within the bounding box *b*.

If any two rectangles $r, \tilde{r}$ of the set $R'$ satisfy the following conditions:
1) Their projects on X-axis are not overlapped each other, i.e.

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

2) No other rectangles of the set $R$ exist between the rectangles $r, \tilde{r}$, i.e.

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

Then, it is necessary to divide the rectangles $r, \tilde{r}$ by whitespace gap. The formulated problem is to divide all these rectangles of the set $R'$, satisfying

the conditions (1) and (2), by a minimal number of whitespace gaps.

## Algorithm

Our algorithm consists of two steps. In the first step, following is performed for each rectangle $r : r \in R$:

1) The first line (or rule) is extended from the left bound of the rectangle $r : r \in R$ to up and down until it is stopped at either any other rectangle $\tilde{r} : \tilde{r} \in R$, $\tilde{r} \neq r$ or the bounding box *b*, as shown in Fig. 2, *a*. In this case, each resulting line is added in the set $L$.
2) The second line (or rule) is extended from the right bound of the rectangle $r : r \in R$ by analogy with the first case. In this case, each resulting line is added in the set $\tilde{L}$.

In the second step, couples of the lines $(\tilde{l}, l)$ are formed. In the couple, the first line is either $\tilde{l}$: $\tilde{l} \in \tilde{L}$ or the left bound of the bounding box *b*. The second line is either $l$: $l \in L$ or the right bound of the bounding box *b*. The couple of the lines $(\tilde{l}, l)$ satisfies the following condition: $y_t(l) = y_t(\tilde{l})$ and $y_b(l) = y_b(\tilde{l})$. Moreover, there are no rectangles of the set *R* between the lines of the couple $(\tilde{l}, l)$, i.e.

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

The each couple of the lines $(\tilde{l}, l)$ forms a whitespace gap $g = \left(x_l(\tilde{l}), y_t(l), x_r(l), x_b(l)\right)$, as shown in Fig. 2, *b*. The set of these gaps $G = \{g_1, \ldots, g_m\}$, $m \in \mathbb{N}$ is the output of our algorithm.

In below the algorithm is expressed by the pseudo-code. It is supposed that all sets are presented by lists. The function $\mathrm{SORT}(S, (c_1, c_2))$ sorts elements (e.g. lines, rectangles) in the list $S$ lexicographically by increasing $(c_1, c_2)$-coordinates, i.e. $\forall s, \tilde{s} : s, \tilde{s} \in S$ it is correct that $s < \tilde{s}$, if either $c_1(s) < c_1(\tilde{s})$, or $c_1(s) = c_1(\tilde{s})$ and $c_2(s) < c_2(\tilde{s})$. In the line 01, the rectangle is formed by the top bound of the bounding box. The other rectangle is formed by the bottom bound of the bounding box (in the line 02). The rectangles $r_0$ and $r_{n+1}$ are added to the list *R* (in the line 03). In the line 04, the rectangles of the list *R* are lexicographically sorted by increasing $(y_t, x_l)$-coordinates (top to bottom and left to right). In the lines 05-22, the rectangles of the list *R* are handled (top to bottom and left to right). In the lines 07-11, the rectangles of the list *R* are handled (right to left and bottom to top), while lines from the left and right bounds of the current rectangle are extended to up. In the lines 12-16, the rectangles of the list *R* are handled (top to bottom and left to right), while lines from the left and right bounds of the current rectangle are extended to down. If the step 1 results in the following nonempty sets:

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

then step 2 is started. The line $\tilde{l}$ is formed by the left bound of the bounding box (in the line 23). The other $l$ is formed by the right bound of the bounding box (in the line 24). The line $\tilde{l}$ is added to the list $\tilde{L}$ (in the line 25). The line $l$ is added to the list $L$ (in the line 26). In the lines 27-28, the lines of the lists $L$ and $\tilde{L}$ are lexicographically sorted by increasing $(x, y_t)$-coordinates (left to right and top to bottom). In the lines 29-34, the lines of the list $\tilde{L}$ are handled (left to right and top to bottom). In the lines 30-34, the lines of the list $L$ are handled (left to right and top to bottom).

In below the step 1 of the algorithm is presented by pseudo-code.

```
Input:  b, R = {r_1, ..., r_n}, n ∈ ℕ
Output: G = {g_1, ..., g_m}, m ∈ ℕ
01 create r_0 ← (x_l(b), y_t(b), x_r(b), y_t(b))
02 create r_{n+1} ← (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 = 1,n do
06    y_l, y_r, y_l, y_r ← null
07    for j = i-1,0 do
08       if y_t(r_i) > y_b(r_j) then
09          if y_l = null
            and x_l(r_j) < x_l(r_i) < x_r(r_j)
```

then $\overline{y_l} \leftarrow y_b(r_j)$

10 if $\overline{y_r} =$ null
and $x_l(r_j) < x_r(r_i) < x_r(r_j)$
then $\overline{y_r} \leftarrow y_b(r_j)$

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 $\underline{y_l} \leftarrow y_t(r_j)$

15 if $\underline{y_r} =$ null
and $x_l(r_j) < x_r(r_i) < x_r(r_j)$
then $\underline{y_r} \leftarrow y_t(r_j)$

16 if $\underline{y_l}, \underline{y_r} \neq$ null
then exit for $j$

17 if not $L$ contains $\left(x_l(r_i), \overline{y_l}, \underline{y_l}\right)$
then

18 create $l \leftarrow \left(x_l(r_i), \overline{y_l}, \underline{y_l}\right)$

19 add $l$ to $L$

20 if not $\tilde{L}$ contains $\left(x_r(r_i), \overline{y_r}, \underline{y_r}\right)$
then

21 create $\tilde{l} \leftarrow \left(x_r(r_i), \overline{y_r}, \underline{y_r}\right)$

22 add $\tilde{l}$ to $\tilde{L}$

In below the step 2 of the algorithm is prescented by pseudo-code.

23 create $\tilde{l} \leftarrow \left(x_l(b), y_t(b), y_b(b)\right)$

24 create $l \leftarrow \left(x_r(b), y_t(b), y_b(b)\right)$

25 add $\tilde{l}$ to $\tilde{L}$

26 add $l$ to $L$

27 $\mathrm{SORT}\left(L, (x, y_t)\right)$

28 $\mathrm{SORT}\left(\tilde{L}, (x, y_t)\right)$

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

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

31 if $x(\tilde{l}_i) < x(l_j)$ and $\overline{y}(\tilde{l}_i) = \overline{y}(l_j)$
and $\underline{y}(\tilde{l}_i) = \underline{y}(l_j)$ then

32 create $g \leftarrow \left(x(\tilde{l}_i), \overline{y}(\tilde{l}_i), x(l_j)\underline{y}(l_j)\right)$

33 add $g$ to $G$

## Conclusion

The proposed algorithm can be used for page segmentation. For example, it can be used for column detection on a page with multi-column text. Furthermore, the algorithm can be used for tasks of table segmentation into individual columns. In particular, the algorithm is used in the method for table detection proposed in the paper [9]. The algorithm is sufficient simple for implementation. The computational complexity of the algorithm is $O(n^2)$.

This work was supported by RFBR (projects 09-07-12017-ofi_m and 08-07-00163-a), the grant of the RF President (MK-3592.2010.8), and the Lavrent'ev grants for youth projects of SB RAS.

## References

1. Baird H.S., Jones S.E., Fortune S.J. Image segmentation by shape-directed covers // In Proc. Int. Conf . on Pattern Recognition. USA, 1990. Vol. 1, P. 820–825.
2. Breuel T.M. Two Geometric Algorithms for Layout Analysis // In Proc. of the 5th Int. Workshop on Document Analysis Systems. Lecture Notes in Computer Science. Vol. 2423. P. 188−199.
3. Cattoni R., Coianiz T., Messelodi S., Modena C.M. Geometric layout analysis techniques for document image understanding: a review // Technical report. IRST. Trento, Italy. 1998. 68 P.
4. Chaudhuri J., Nandy S.C., Das S., Largest empty rectangle among a point set // Journal of Algorithms. 2003. Vol. 46, Issue 1, P. 54–78.
5. Kise K., Sato A., Iwata M. Segmentation of page images using the area Voronoi diagram // Computer Vision and Image Understanding. Elsevier Science Inc. 1998. Vol. 70, No. 3. P. 370–382.
6. Lyman P., Varian H.R. How much information? // Technical Report. 2003. Available on the address http://www.sims.berkeley.edu/how-much-info-2003.
7. Marinai S., Fujisawa H. (Eds.). Machine Learning in Document Analysis and Recognition // Series: Studies in Computational Intelligence. 2008. Vol. 90. 434 p.
8. Orlowski M. A new algorithm for the largest empty rectangle problem // Algorithmica. 1990. Vol. 5, No. 1-4. P. 65–73.
9. Shigarov A.O., Bychkov I.V., Ruzhnikov G.M., Khmel'nov A.E. A method for table detection in metafiles // Pattern Recognition and Image Analysis. 2009. Vol. 19, No 4. P. 693–697.