An Algorithm for Finding Maximal Whitespace Rectangles at Arbitrary Orientations for Document Layout Analysis

Thomas M. Breuel
PARC, Inc., 3333 Coyote Hill Rd., Palo Alto, CA 94304, USA

Abstract

The analysis of the background structure (whitespace) of page images has become an important technique for physical document layout analysis. Globally maximal whitespace rectangles have been previously demonstrated to constitute a concise representation of the major layout features of documents. However, previous methods for computing maximal whitespace rectangles were limited to axis-aligned rectangles. This paper presents an algorithm that finds globally maximal whitespace rectangles on page images at arbitrary orientations. The new algorithm eliminates the need for page rotation correction prior to background analysis and can be applied to considerably more complex page layouts than previously possible. The algorithm is resolution independent and takes as input a list of foreground shapes (e.g., character or word bounding boxes or polygons) and a set of parameter ranges; it outputs the N largest non-overlapping maximal whitespace rectangles whose parameters (location, width, height, orientation) fall within the required parameter ranges. Examples of applications of the method to severely skewed documents, as well as the UW3 database, are presented.

1 Introduction

Many methods for document layout analysis based on the structure of the page background have been described in the literature (e.g., [1]; [4] is a review with pointers to additional papers). Morphological approaches use structuring elements to "smear" together foreground page elements and identify large regions of page background. Voronoi-based approaches compute distance transforms or Voronoi diagrams for the elements constituting the page foreground and analyze the boundaries between Voronoi cells[6].

Page background analysis in terms of maximal rectangular whitespace regions has been explored in detail in [1, 2]. Rectangular regions are natural for describing background structure because textual regions are usually approximately rectangular themselves, making the background whitespace representable as the union of a fairly small number of whitespace rectangles. The methods described in [1] use line sweeps to enumerate locally maximal whitespace rectangles, and in a postprocessing step select rectangles satisfying optimality criteria. Those sweep algorithms are efficient, but they tend to be difficult to implement and do not return results in best-to-worst order. A very easy-to-implement algorithm that returns results in best-to-worst order was described in [2]. That paper also demonstrated how optimality criteria can be defined, based on proximity of the whitespace rectangles to character bounding boxes, that result in near perfect identification of column boundaries by whitespace analysis, and how the detection of column boundaries can be extended into a complete physical page layout analysis.

Both line sweep and branch-and-bound algorithms find maximal whitespace rectangles whose sides are aligned with the page coordinate system. When presented with pages at even a small rotation angle, such algorithms fail: rectangular whitespace column separators are potentially very narrow, and an axis-aligned rectangle fails to "fit in" properly even at small rotation angles. In fact, the longer a whitespace column separator is, the narrower it can be and remain visually salient, exacerbating the problem.

This problem can be addressed by correcting for page rotation or skew prior to applying such whitespace analysis techniques. However, determining page rotation very accurately is considerably less reliable in multi-column documents.

Since, based on prior results, rectangular whitespace analysis appears to be a highly reliable method for detecting major layout features in documents whose features are axis aligned, it would seem conceptually easiest to extend the method to work on non-axis aligned whitespace rectangles.

2 Non-Axis Aligned Maximal Whitespace Rectangles

2.1 Geometric Preliminaries

We assume as input to the algorithm an outside bounding box r_b and a set of axis-aligned rectangular obstacles collection of rectangles $C = \{r_0, \ldots, r_n\}$ (the algorithm works without changes for non-axis aligned rectangular obstacles or arbitrary polygonal obstacles). A non-axis aligned whitespace rectangle s can be described in terms of five parameters: its center $s_c = (s_x, s_y)$, its width s_w, its height s_h, and its orientation s_α. Without loss of generality, let us picture the rectangle as being taller and narrow, and refer to its closer-to-vertical axis as its major axis. The direction of the minor axis is then $v_\alpha = (\cos \alpha, \sin \alpha)$, and the direction of the major axis is $v_\alpha^\perp = (-\sin \alpha, \cos \alpha)$.

Now, let us assume that we are given four of these five parameters: the center s_c, the width s_w, and the orientation s_α. Once we have chosen these four parameters, we can easily determine the maximum whitespace rectangle consistent with them: we project all the obstacles that are closer than s_w to the major axis onto the major axis and compute the minimum distance of each projection from the center of s. Let us call $Q(s_x, s_y, s_w, s_\alpha)$ the function that computes the maximum area achievable for a given set of parameters s_x, s_y, s_w, and s_α.

Now consider all the r_i that are close to the major axis than s_w. We wish to compute the minimum distance of all those rectangles from the minor axis; this be the maximum height $\hat{s}_h$ achievable by the rectangle s under the given parameters. Then, $Q(s_x, s_y, s_w, s_\alpha) = s_w\, \hat{s}_h$.

In order to compute the distance of each obstacle from the minor axis, we need to distinguish two cases. If the obstacle straddles the minor axis, its distance is zero. Otherwise, its distance is the minimum of the distances of each of its corners $p_{i,j}$ for $j = 1 \ldots 4$. We can determine both cases by considering the dot products $v_\alpha^\perp \cdot (p_{i,j} - s_c)$. If there are two such dot products that have different signs, then the obstacle straddles the minor axis and its distance is zero. Otherwise, its distance is $\min_j |v_\alpha^\perp \cdot (p_{i,j} - s_c)|$.

This minimum-distance rectangle functions a little bit like the pivot in the axis aligned algorithm. If we iterated through all the possible centers, widths, and orientations, we could determine the non-axis aligned maximal whitespace rectangle this way–very slowly, however.

What we need is a more efficient way of exploring the parameter space of possible solutions. If we could determine for whole ranges of parameters that there is no good solution possible, we could eliminate large regions of the parameter space quickly. Fortunately, interval arithmetic [5] provides us with just that capability.

2.2 Interval Arithmetic

Interval arithmetic replaces the numerical values in arithmetic expressions with intervals and provides rules for computing with those intervals. Let $\circ$ be an arithmetic operator and $u = [\underline{u}, \overline{u}]$ and $v = [\underline{v}, \overline{v}]$ intervals of real numbers. Then we extend $\circ$ to intervals by defining

$$u \circ v = [\min_{x \in u, y \in v} x \circ y, \max_{x \in u, y \in v} x \circ y]$$

It is easily seen that applying this rule consistently in the extension of some function f to interval arithmetic, then computing

$$[\underline{y}, \overline{y}] = f([\underline{x}, \overline{x}])$$

we are guaranteed, for every $x \in [\underline{x}, \overline{x}]$ that $y \in [\underline{y}, \overline{y}]$, where $y = f(x)$, and analogously for functions of multiple arguments. $f([\underline{x}, \overline{x}])$ is called a natural inclusion function for $f(x)$, and while its interval result $[\underline{y}, \overline{y}]$ does not represent tight bounds, if f is sufficiently smooth, then the interval $[\underline{y}, \overline{y}]$ is guaranteed to become narrower as the interval $[\underline{x}, \overline{x}]$ becomes narrower. Interval arithmetic provides us with an efficient means for computing such extended operators, and there are libraries and compilers that provide intervals and operations on them as built-ins. For more information on the subject, the reader is referred to the literature [5].

2.3 Computing Bounds

Using interval arithmetic, we have now the tools in hand to structure the search. Instead of finding the maximum whitespace rectangle for a specific choice of four parameter values, we replace s_x, s_y, s_w, and s_α with intervals. Our overall goal in this section is to extend the function $Q(s_x, s_y, s_w, s_\alpha)$ to a natural inclusion function. Since the evaluation of $Q(s_x, s_y, s_w, s_\alpha)$ involved some conditionals, we cannot simply replace its operators with interval equivalents, but need to analyze its evaluation more carefully.

First, we need to determine the distance of each obstacle from the major axis of s. In the non-interval case, every obstacle whose distance from the major axis was less than s_w limited the value of s_h. However, now that we are using ranges of parameters, comparing distances with s_w can have three answers: definitely less than s_w, definitely greater than s_w, or indeterminate. Only in the case that the obstacle is definitely closer to the major axis of s than any value in the interval s_w do we use its distance from the minor axis to limit the value of s_h. When the obstacle is definitely farther away from the major axis of s than any value in the interval s_w, we remove the obstacle from further consideration. When the comparison is indeterminate, then we retain the obstacle for future consideration but do not use it in restricting the value of s_h further.

The projected distances of the p_j from the center of s, $|v_\alpha^\perp \cdot (p_j - s_c)|$, then also become intervals. An obstacle definitely straddles the minor axis if one of these dot products is definitely less than zero and another one is definitely greater than zero. Otherwise, the projected distances are the minima of the interval distances of the individual corners of the obstacle, as before.

2.4 Search

Using the above methods, we can compute bounds on $Q(s_x, s_y, s_w, s_\alpha)$ over parameter ranges easily and efficiently. What remains is to structure the search in such a way as to be able to quickly exclude large regions of parameter space that cannot contain good solutions. We take a branch-and-bound approach over the four dimensional parameter space, similar to the approach used in [3]. To make the search efficient, we keep track of points that need to be involved in further comparisons using *matchlists* [3]. The overall algorithm is as follows:

- We maintain a priority queue of potential solution.
- Each potential solution consists of intervals for the four parameters s_x, s_y, s_w, s_α, together with a list of potential obstacles. The cross product of the four intervals defines a hyper-rectangle in the four-dimensional parameter space.
- The priority queue is initialized with a single element, consisting of intervals representing the entire range of parameters to be searched and the r_i as candidate obstacles.
- During the search, we remove the top element from the priority queue.
- If the range of parameters is numerically sufficiently accurate, we return it as a solution; the lower bounds of its parameter intervals are guaranteed to correspond to a maximal empty rectangle to within the desired numerical accuracy.
- Otherwise, we split one or more of the intervals of parameter values in half, giving us a rectangular subdivision of the four dimensional region in parameter space we were searching in this step. For each of the parameter subregions and the set of candidate, we carry out the following steps.
- Using the algebraic methods described in the previous section, we remove from set of candidate obstacles those that are definitely further away from the major axis than the interval s_w..
- Among the remaining obstacles, we compute the interval $\hat{s}_h$ of possible minimal distances of those rectangles from the minor axis.
- Finally, we remove all those obstacles whose distance from the minor axis is definitely larger than the $\hat{s}_h$ we just computed. (Alternatively, we retain the $\hat{s}_h$ from the parent region and combine this step with Step 2.4.)
- We enqueue each subregion and its list of remaining obstacles into the priority queue, using its range for $s_w \cdot \hat{s}_h$ as the priority.

This algorithm can be implemented in about 400 lines of C++ code, using small libraries implementing interval arithmetic and simple data structures. Multiple, non-overlapping solutions can be obtained efficiently as described in [3].

Two other practical implementation issues remain to be described. First, we probably do not want to obtain empty rectangles that extend far beyond the page border. We could impose this constraint by choosing carefully the parameter regions that the algorithm explores. However, it is both simpler and faster to create an actual bounding box out of rectangular obstacles placed around the region we wish to search; this will impose the bounding box constraint exactly with no further algorithmic complications.

Second, as observed in previous work on background analysis using axis aligned rectangles [2], documents contain many locally maximal whitespace rectangles that do not correspond to semantically meaningful layout components. What was shown in that paper is that we can reliably distinguish such spurious whitespace rectangles from semantically meaningful ones by taking into account the proximity of a whitespace rectangle to foreground components: semantically meaningful components will almost always separate *something*; that is, they will have a significant density of foreground components close to each of their sides. This additional constraint can be incorporated easily into the current algorithm. Above, we already the distance of each obstacle from the major and minor axis using $|v_\alpha \cdot (p_{i,j} - s_c)|$ and $|v_\alpha^\perp \cdot (p_{i,j} - s_c)|$, where $p_{i,j}$ is one of the corners of the obstacle. If these distances are less than s_w, the obstacle protrudes inside the rectangle s; furthermore, the sign of the dot product tells us which side of the axis the obstacle is on. To require a minimum number of obstacles within a border β of the rectangle, we simply check for the number of obstacles whose distances from the major axis are between s_w and $s_w + \beta$ and whose distances from the minor axis are between $\hat{s}_h$ and $\hat{s}_h + \beta$. Furthermore, we retain these additional obstacles on the obstacle list, so that they will be taken into account by each child of a search node.

3 Experimental Results

To evaluate the performance of the algorithm, it was applied to documents with varying degrees of skew, rotated page images derived from ground truth by bitmap rotation, and to unaltered documents from the UW3 database. All images used in the experiments are letter-sized pages represented at 300dpi (approximately 2550 by 3300 pixels). Visually comparing the performance on images scanned at an angle or derived by bitmap rotation from given page images showed that the algorithm reliably found the rotated whitespace region corresponding to the maximal whitespace rectangle identified by the algorithm described in [2]. Furthermore, detection of whitespace regions by the algorithm at an angle was not significantly slower than detection of upright whitespace regions. Figure 1 shows a representative example of such images.

Based on these preliminary results, a large scale evaluation of the performance of the algorithm was carried out on the UW3 database, which contains a wide variety of document layouts (that database does not have much variability

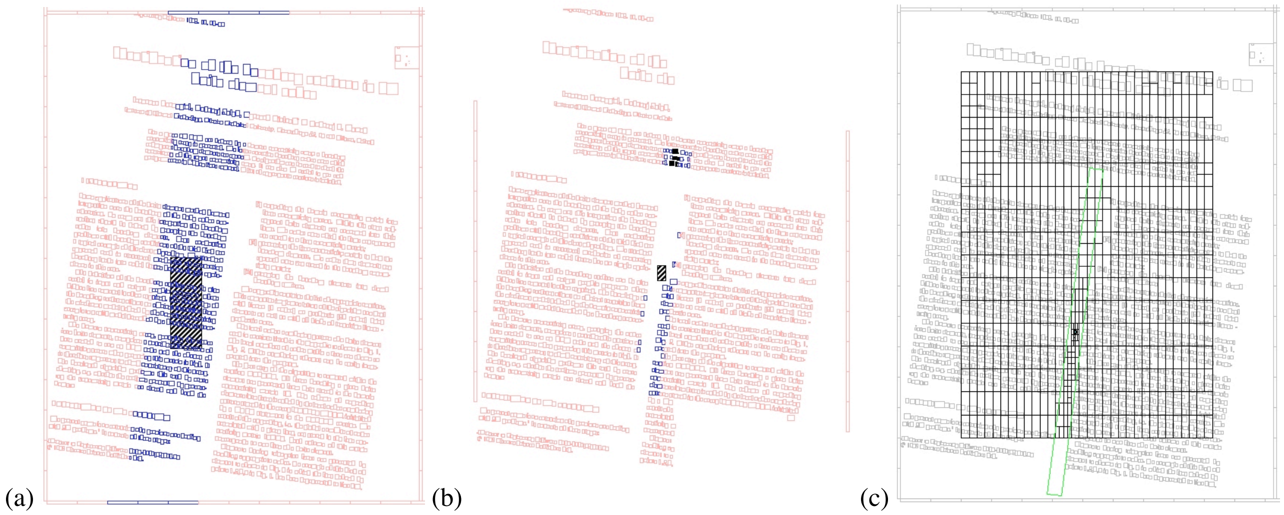

Figure 1. Figure illustrating the progress of the branch-and-bound search. In (a) and (b), the hatched region in the center shows the parameter ranges for s_x and s_y being explored. The lightest rectangles (pink) are shown to provide context and do not participate in the evaluation (i.e., they are not on the obstacle list). The darker, unfilled rectangles (blue) are rectangles on the list of obstacles. The black, filled rectangles are all the obstacles that have resulted in constraints being placed on $\hat{s}_h$ within the single iteration depicted by each of the panels. (a) shows the search early on, (b) shows the search after parameter intervals have become sufficiently small to result in substantial reductions of the list of obstacles. Panel (c) shows the exploration of parameter space in s_x and s_y on the set of obstacles (light gray) and the solution (light gray or green). For clarity, the ranges of the parameters were restricted slightly. Note that in most areas of the page, the search tree is not explored very deeply.

in terms of skew, but, as noted above, that does not affect the running times or error rates significantly). The purpose of those experiments was primarily to determine the performance of the algorithm on complex real-world layouts. Two sets of experiments were conducted. In the first set, the algorithm was presented with the same set of obstacles used in previous experiments [2]; those obstacles were derived from the raw bounding boxes for connected components on the page by removing very small and very large bounding boxes and grouping very closely spaced bounding boxes together. Those are operations that can be carried out reliably even without knowing or correcting for page rotation, as long as the page is not too severely rotated. The running times on those experiments are shown in Figure 2 (b). To test the algorithm on larger inputs and to remove the need for any kind of processing after connected component analysis, in a second set of experiments, the algorithm was applied to the raw set of bounding boxes for all the connected components of on the page. The running times from those experiments are shown in Figure 2 (c).

4 Discussion

This paper has described a simple, practical, and robust algorithm for finding non-axis aligned maximal whitespace rectangles given a list of obstacles and a bounds on the parameters. Experimental results show the algorithm to work efficiently and robustly on real-world documents; the algorithm takes a median of less than a second on preprocessed bounding boxes (up to 1400 obstacles) and less than eight seconds for raw bounding boxes (up to 11000 obstacles) for any page image from the UW3 database. The algorithm can also easily be modified to use, say, convex polygons instead of axis aligned rectangles to describe the foreground structure (making the approach entirely rotation invariant), or find other maximal shapes, like circles, squares, ellipses, rounded rectangles, etc.

Together with prior results [2] demonstrating the use of maximal whitespace rectangles and proximity as the basis of reliable physical document layout analysis, this now gives us a simple two-step process for recovering the major layout components of a document:

- Find the non-axis aligned maximal whitespace rectangles satisfying proximity conditions to foreground components.

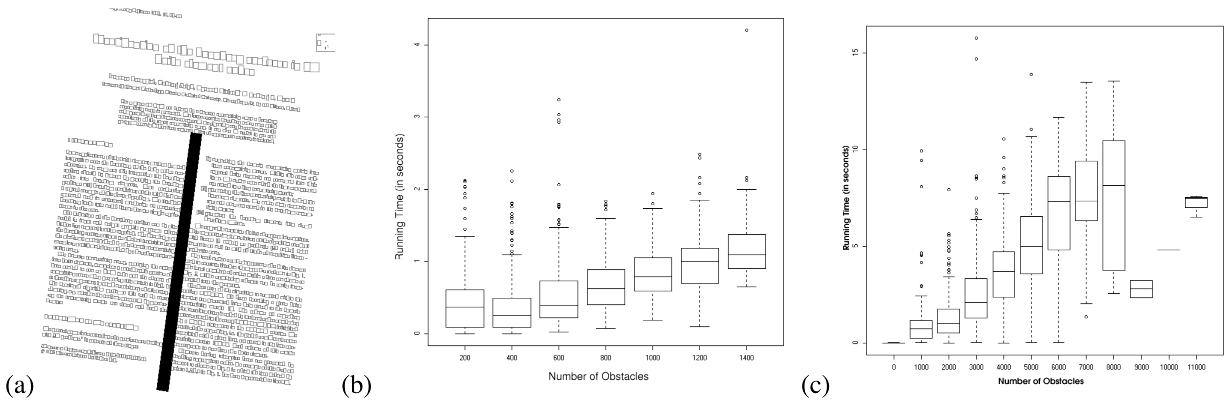

Figure 2. Panel (a) shows the optimal solution (solid black rectangle) for a severely skewed mixed 1/2-column document (obstacles are solid dark gray outlines; document is 2592 by 3300 pixels). Panels (b) and (c) show the running times (as obtained using the "time" command line function on a 1GHz PC running Debian Linux) of the algorithm by the number of obstacles. Panel (b) shows running times when using grouped bounding boxes as obstacles and Panel (c) shows running times when using raw bounding boxes as obstacles. The horizontal axis shows the number of obstacles and the vertical axis the running time in seconds. Running times are shown as standard statistical boxplots (the central line is the median, and the box shows the two quartiles).

• Use those obstacles together with a constrained line finding algorithm [2] for finding the text lines in each column.

This approach obviates the need for a separate document image deskewing step. The advantage of this new two-step approach is that it eliminates a potential source of errors (incorrect skew correction, particularly common in multicolumn documents), and that it allows the system to cope directly with documents in which text is present in multiple orientations (e.g., scanned books, in which facing pages often have slightly different orientations). By removing the need for one software component and processing step, the layout analysis system can also be simplified and slightly speeded up; the speedup is a consequence of the fact that the non-axis aligned maximal whitespace rectangle algorithm has comparable running times to a high quality skew detection algorithm and makes the initial skew detection step redundant–after whitespace analysis and constrained line finding, the page skew is known anyway.

Overall, given this algorithm and the constrained line finding algorithm described in [2], we have a suite of techniques and algorithms at our disposal that are resolution independent and work in terms of geometric objects (as opposed to image processing), that have practical running times for real-world documents, that compute globally optimal solutions with well-defined geometric properties, and that are concise and easy to implement. A more careful evaluation of the quality of the layouts found by these methods as part of an end-to-end document layout analysis system remains to be done; this paper has concentrated on algorithmic issues, presenting these methods as tools that can be used by others in the community, and demonstrating that such methods have performance that make them practical for real-world applications.

References

[1] H. S. Baird. Background structure in document images. In *H. Bunke, P. S. P. Wang, & H. S. Baird (Eds.), Document Image Analysis, World Scientific, Singapore*, pages 17–34, 1994.

[2] T. M. Breuel. Two algorithms for geometric layout analysis. In *Proceedings of the Workshop on Document Analysis Systems, Princeton, NJ, USA*, 2002.

[3] T.M. Breuel. Robust least square baseline finding using a branch and bound algorithm. In *Proceedings of the SPIE - The International Society for Optical Engineering*, page (in press), 2002.

[4] R. Cattoni, T. Coianiz, S. Messelodi, and C. M. Modena. Geometric layout analysis techniques for document image understanding: a review. Technical report, IRST, Trento, Italy, 1998.

[5] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis*. Springer Verlag, Berlin, 2001.

[6] K. Kise, A. Sato, and M. Iwata. Segmentation of page images using the area voronoi diagram. *Computer Vision and Image Understanding*, 70(3):370–82, June 1998.