Two Geometric Algorithms for Layout Analysis

Thomas M. Breuel

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
tbreuel@parc.xerox.com

Abstract. This paper presents geometric algorithms for solving two key problems in layout analysis: finding a cover of the background whitespace of a document in terms of maximal empty rectangles, and finding constrained maximum likelihood matches of geometric text line models in the presence of geometric obstacles. The algorithms are considerably easier to implement than prior methods, they return globally optimal solutions, and they require no heuristics. The paper also introduces an evaluation function that reliably identifies maximal empty rectangles corresponding to column boundaries. Combining this evaluation function with the two geometric algorithms results in an easy-to-implement layout analysis system. Reliability of the system is demonstrated on documents from the UW3 database.

1 Introduction

A wide variety of algorithms for geometric layout analysis of document images have been proposed. Among them are morphology or "smearing" based approaches, projection profiles (recursive X-Y cuts), texture-based analysis, analysis of the background structure, and others (for a review and references, see [6]). While layout analysis is a simpler problem than general image segmentation, it still raises challenging issues in geometric algorithms and image statistics.

This paper presents algorithms for addressing two key problems in geometric layout analysis. The first is an efficient and easy to implement algorithm for analyzing the whitespace or background structure of documents in terms of rectangular covers. Background structure analysis as an approach to document layout analysis has been described by a number of authors [13, 2, 12, 8, 1, 9]. The work by Baird *et al.* [2] analyzes background structure in terms of rectangular covers, a computationally convenient and compact representation of the background. However, past algorithms for computing such rectangular covers have been fairly difficult to implement, requiring a number of geometric data structures and dealing with special cases that arise during the sweep (Baird, personal communication). This has probably limited the widespread adoption of such methods despite the attractive properties that rectangular covers possess. The algorithm presented in this paper requires no geometric data structures to be implemented and no special cases to be considered; it can be expressed in less than 100 lines of Java code. In contrast to previous methods, it also returns solutions in order of decreasing quality.

The second algorithm presented here is a text line finding algorithm that works in the presence of “obstacles”. That is, given a set of regions on the page that are known to be free of text lines (e.g., column separators) and a collection of character bounding boxes, the algorithm will find globally optimal maximum likelihood matches to text lines under a Gaussian error model, subject to the constraint that no text line crosses an obstacle. In contrast, many previous message to line finding (e.g., projection methods, Hough transform methods, etc.) either do not work reliably for multi-column documents or multiple text orientations on the same page, or they require a complete physical layout segmentation into disjoint text regions with uniform text line orientations prior to their application.

Each of the algorithms presented in this paper has useful applications in existing layout analysis systems. Taken together, these two algorithms permit us to take a new approach to document layout segmentation.

Traditional document layout analysis methods will generally first attempt to perform a complete global segmentation of the document into distinct geometric regions corresponding to entities like columns, headings, and paragraphs using features like proximity, texture, or whitespace. Each individual region is then considered separately for tasks like text line finding and OCR. The problem with this approach lies in the fact that obtaining a complete and reliable segmentation of a document into separate regions is quite difficult to achieve in general. Some decisions about which regions to combine may well involve semantic constraints on the output of an OCR system. However, in order to be able to pass the document to the OCR system in the first place, we must already have identified text lines, leading to circular dependencies among the processing steps.

In contrast, if we can perform text line finding in the presence of obstacles, it is not necessary to perform a complete segmentation of the document in order to perform OCR. Rather, all that is needed is the identification of vertical spaces or lines separating text in different columns. That problem turns out to be considerably simpler. It can be accomplished quite reliably using the whitespace analysis algorithm described in this paper using a novel evaluation function.

2 Whitespace Cover

2.1 Problem Definition

We define the maximal white rectangle problem as follows. Assume that we are given a collection of rectangles $C = \{r_0, \ldots, r_n\}$ in the plane, all contained within some given bounding rectangle r_b. In layout analysis, the r_i will usually correspond to the bounding boxes of connected components on the page, and the overall bounding rectangle r_b will represent the whole page. Also, assume that we are given an evaluation function for rectangles $Q : \mathbb{R}^4 \to \mathbb{R}$ satisfying, for any two rectangles r and r' that

$$r \subseteq r' \Rightarrow Q(r) \leq Q(r') \tag{1}$$

In the case described in [8], the Q function is simply the area of the rectangle, which is easily seen to satisfy the condition expressed in Equation 1. The maximal white

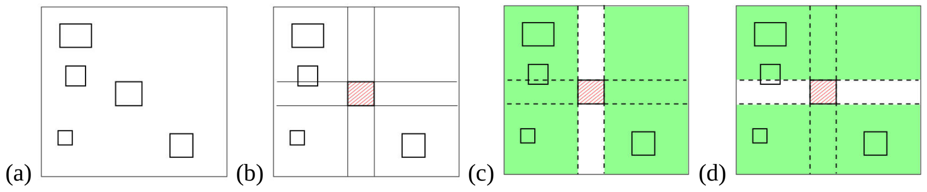

Fig. 1. Figure illustrating the recursion step of the branch-and-bound whitespace cover algorithm. See the text for an explanation.

rectangle problem is to find a rectangle $\hat{r} \subseteq r_b$ that maximizes $Q(T)$ among all the possible rectangles $r \subseteq r_b$, where r overlaps none of the rectangles in C. Or, expressed using mathematical notation:

$$\hat{r} = \hat{r}(C, r_b, Q) = \arg\max_{r \in U} Q(r) \text{ where } U = \{r \subseteq r_b | \forall c \in C : r \cap c = \emptyset\} \quad (2)$$

2.2 Algorithm

As noted above, there are several algorithms for maximal empty rectangle problems, including those from computational geometry (e.g., [11]) and document analysis (e.g., [2]). Unfortunately, such algorithms tend to be fairly complex to implement and have not found widespread use.

The algorithm presented in this paper for the maximum empty rectangle problem can be used with obstacles that are points or rectangles. The key idea is analogous to quicksort or branch-and-bound methods. It is illustrated in Figure 1. Figure 1(a) shows the start of the algorithm: we are given an outer `bound` and a collection of rectangles (`obstacles`). If none of the obstacles are contained within `bound`, then we are done: the `bound` is itself the maximal rectangle given the obstacles. If one or more obstacles are contained within `bound`, we pick one of those rectangles as a "pivot" (Figure 1(b)). A good choice is a rectangle that is centrally located within the `bound`. Given that we know that the maximal rectangle cannot contain any of the `obstacles`, in particular, it cannot contain the pivot. Therefore, there are four possibilities for the solution to the maximal white rectangle problem: to the left and right of the pivot (Figure 1(c)) or above and below the pivot (Figure 1(d)). We compute the obstacles overlapping each of these four subrectangles and evaluate an upper bound on the quality of the maximal empty rectangles that is possible within each subrectangle; because of the monotonicity property (Equation 1), the quality function Q applied to the bounds of the subrectangles itself serves as an upper bound. The subrectangles and their associated obstacles and qualities are inserted into a priority queue and the above steps are repeated until the first obstacle-free rectangle appears at the top of the priority queue; this rectangle is the globally optimal solution to the maximal empty rectangle problem under the quality function Q. This algorithm is given in pseudo-code in Figure 2.

To obtain the n-best solutions, we can keep expanding nodes from the priority queue until we obtain n solutions in order of decreasing quality. However, many of those solutions will overlap substantially. The following greedy variant of the algorithm for

```
def find_whitespace(bound,rectangles):
    queue.enqueue(quality(bound),bound,rectangles)
    while not queue.is_empty():
        (q,r,obstacles) = queue.dequeue_max()
        if obstacles==[]:
            return r
        pivot = pick(obstacles)
        r0 = (pivot.x1,r.y0,r.x1,r.y1)
        r1 = (r.x0,r.y0,pivot.x0,r.y1)
        r2 = (r.x0,pivot.y1,r.x1,r.y1)
        r3 = (r.x0,r.y0,r.x1,pivot.y0)
        subrectangles = [r0,r1,r2,r3]
        for sub_r in subrectangles:
            sub_q = quality(sub_r)
            sub_obstacles =
                [list of u in obstacles if not overlaps(u,sub_r)]
            queue.enqueue(sub_q,sub_r,sub_obstacles)
```

Fig. 2. Pseudo-code for finding the globally optimal whitespace rectangle. A complete Java implementation is about 200 lines of code (statements).

finding the n best solutions addresses this. After we have found the maximal empty rectangle $\hat{r}$, we can add it to the list of obstacles and keep expanding. When we dequeue a search state, we check whether the list of obstacles has changed and, if so, recompute the quality of the node and re-enqueue it. This will result in a greedy cover of the whitespace with maximal rectangles and is considerably faster than restarting the algorithm.

Furthermore, rather than insisting on a cover of completely disjoint rectangles, we can allow for some fractional or absolute overlap among them. A careful implementation of finding such partially overlapping maximal empty rectangles might incorporate the overlap constraint into the computation of the upper bound during the partitioning process. However, the algorithm runs fast enough on real-world problems, and the number of solutions we desire is usually small enough, that it is sufficient merely to generate maximal empty rectangles in order of decreasing quality using the unmodified algorithm, test for overlap of any new solution with all the previously identified ones, and reject any new solution that overlaps too much with a previously found solution.

An application of this algorithm for finding a greedy covering of a document from the UW3 database with maximal empty rectangles is shown in Figure 7. Computation times for commonly occurring parameter settings using a C++ implementation of the algorithm on a 400MHz laptop are under a second. As it is, this algorithm could be used as a drop-in replacement for the whitespace cover algorithm used by [8], and it should be useful to anyone interested in implementing that kind of page segmentation system. However, below, this paper describes an alternative use of the algorithm that uses different evaluation criteria.

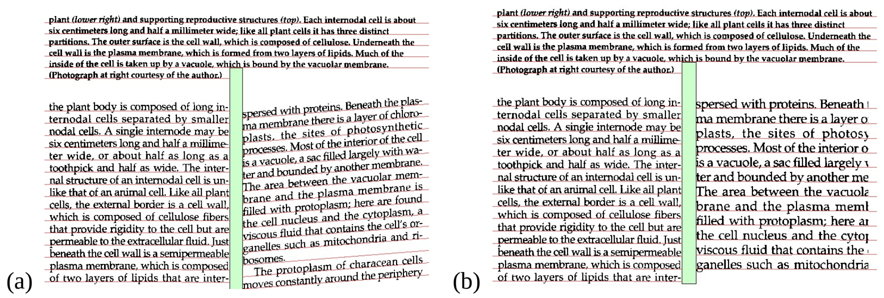

Fig. 3. Application of the constrained line finding algorithm to simulated variants of a page. Gutters (obstacles) were found automatically using the algorithm described in the paper and are shown in green. Text lines were found using the constrained line finder and are shown in faint red. (a) Two neighboring columns have different orientations (this often occurs on the two sides of a spine of a scanned book). (b) Two neighboring columns have different font sizes and, as a result, the baselines do not line up.

3 Constrained Line Finding

3.1 Problem Definition

We will now turn to a second geometric algorithm, one for finding text lines in the presence of obstacles. The “obstacles” will turn out to be the rectangles comprising the whitespace cover found by the algorithm described in the previous section and the evaluation criteria described in the next section. The constrained line finding algorithm is also linked with the algorithm described in the previous section by taking a similar algorithmic approach: branch-and-bound.

The problem that constrained line finding addresses in document analysis is the following. Many documents contain text in multiple columns. Some documents or document images may even contain text at multiple orientations, either because of complex document layouts, or (more commonly) because the two facing pages of a book were scanned at slightly different rotations within the same image. Text lines that are close to each other may therefore still have different line parameters. Some cases are illustrated in Figure 3.

Traditional approaches attempt to cope with such cases by first finding a complete and correct page segmentation and then performing line finding within each text block; that is, they take a hierarchical top-down approach. Unfortunately, finding a complete and correct page segmentation without knowledge of the line structure is difficult. Globally integrated solutions to page layout analysis, like those proposed by Liang et al. [10] avoid this issue, but appear to be complex to implement and so far have not found wide application.

Constrained line finding provides a simpler alternative. A constrained line finder only needs a list of obstacles that lines of text do not cross. These obstacles are generally gutters, and a few graphical elements such as figures or thin vertical lines. Based on the results presented below, finding gutters appears to be a much simpler problem than a complete (even if provisional) layout analysis, and even complex layouts tend

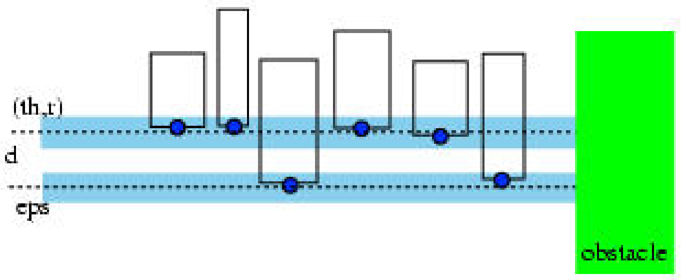

Fig. 4. The text line model used for constrained line finding.

to have a simple gutter structure (see the examples in Figure 7). Those gutters can be identified easily using the whitespace cover method described in the previous section. Furthermore, the constrained line finding method described in this paper can also be used together with orientation independent layout analysis techniques, allowing us to find text lines at arbitrary orientations even in incompletely segmented text.

The approach to constrained text line finding underlying the algorithm in this paper has previously been described for geometric object recognition [3], and applied to text line finding [5]. Let us represent each character on the page by the point at the bottom and center of its bounding box (the alignment point). In the absence of error, for most Roman fonts, each such point rests either on the baseline or on another line parallel to the baseline, the line of descenders. This is illustrated in Figure 4.

For finding “optimal” matches of text line models against the bounding boxes of a page, we use a robust least square model. That is, the contribution of each character to the overall match score of a text line is penalized by the square of the distance of the alignment point from the base line or line of descenders, up to a threshold. This match score corresponds to a maximum likelihood match in the presence of Gaussian error on location and in the presence of a uniform background of noise features, as shown in the literature [7].

Let us assume that lines are parameterized by their distance r from the origin and the orientation θ of their normal. An additional parameter, d, gives the distance of the line of descenders from the baseline. These three parameters (r, θ, d) then determine a text line model. If the alignment points of all connected components on the page are given by $\{p_1, \ldots, p_n\} \subseteq \mathbb{R}^2$, we can express the quality of match (monotonically related to the log likelihood) function as:

$$Q(r, \theta, d) = \sum_i \phi_\epsilon(\text{dist}(l_{r,\theta,d}, p_i)) \tag{3}$$

Here, $\text{dist}(\cdot, \cdot)$ is the Euclidean distance and ϕ is a threshold function

$$\phi_\epsilon(x) = \max(0, 1 - \frac{x^2}{\epsilon^2}) \tag{4}$$

Maximizing $Q(r, \theta, d)$ over all parameters gives us the globally optimal solution to the unconstrained line finding problem. For the constrained line finding problem, we

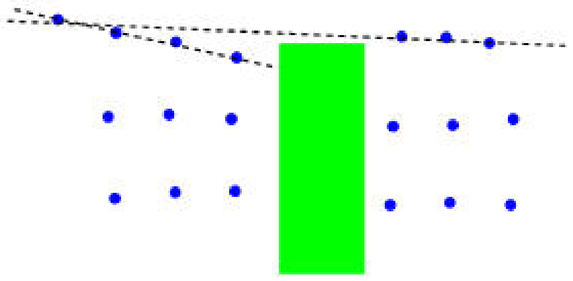

Fig. 5. Illustration of the constrained line finding problem with obstacles. The rectangle is the obstacle and the dots represent points to be matched by a line. Two candidates lines are shown: one dashed line matches four points but is stopped by the obstacle, another dashed line matches five points and narrowly avoids the obstacle.

consider line segments instead of lines and require finding a maximal line segment that does not intersect any of the given obstacles.

3.2 Algorithm

An algorithm for finding globally optimal solutions to the unconstrained text line finding problem has been presented in [5], based on previous work on branch-and-bound methods for geometric matching [4]. We will briefly review the unconstrained method here. The basic idea is to consider rectangular subsets (boxes; cartesian products of line parameter intervals) of the three-dimensional space of text line parameters and compute upper bounds on the value of the quality function achievable over those subsets. Subsets with large upper bounds are subdivided into smaller subsets and reevaluated. Eventually, the rectangular subsets arrived at in this process are small enough to bound the optimal solution to the optimization problem with any desired numerical accuracy. This is an instance of a branch and bound algorithm.

In order to be practical for geometric optimization problems, two difficulties need to be overcome: first, we need to be able to find an upper bound $\hat{Q}$ to the quality function Q over some region, and second, we need to be able to compute that upper bound efficiently. [4] describes the computation of the upper bound $\hat{Q}$ function for a box of line parameters $[\underline{r}, \overline{r}] \times [\underline{\theta}, \overline{\theta}]$. Let us review this approach briefly here. For the moment, to simplify the discussion, consider only the baseline, not the line of descenders. Consider the region L_B swept out by lines with parameters contained in the box of parameters $B = [\underline{r}, \overline{r}] \times [\underline{\theta}, \overline{\theta}]$. We use as our upper bound $\hat{Q}(L_B) = \max_{(r,\theta)\in B} Q(r, \theta)$. Taking advantage of the monotonicity of $\phi_\epsilon(x)$, this bound is easily seen to be

$$\hat{Q}(B) = \sum_i \min_{(r,\theta)\in B} \phi_\epsilon(\text{dist}(l_{r,\theta}, p_i)) \tag{5}$$

$$= \sum_i \phi_\epsilon(\text{dist}(L_B, p_i)) \tag{6}$$

```
def find_constrained_lines(linebox,points,obstacles):
    queue.enqueue(quality(linebox,points),linebox,points,obstacles)
    while not queue.is_empty():
        (q,linebox,points,obstacles) = queue.dequeue_max()
        if accurate_enough(linebox):
            return linebox
        excluded_obstacles =
            [list of obstacle in obstacles
                if linebox.can_not_intersect(obstacle)]
        if excluded_obstacles!=[]:
            ...split linebox at excluded obstacles and enqueue...
        sublineboxes = split(linebox)
        for sub_linebox in sublineboxes:
            sub_points =
                [list of point in points
                    if point.may_match(line)]
            sub_q = quality(sub_linebox,sub_points)
            queue.enqueue(sub_q,sub_linebox,sub_points,obstacles)
```

Fig. 6. Pseudo-code for finding the globally optimal constrained match of a line model against a set of points.

The region L_B is a bow-tie shaped region. It is bounded on four sides by lines given by the extreme values of the line parameter box. The fifth side is bounded by a small circular arc. For the computation of the upper bound $\hat{Q}(B)$, we therefore need to compute the distance of a point p from this region, or at least a lower bound. This computation can be simplified by bounding the circular arc using a fifth line. A lower bound on the distance dist(L_B, p_i) can then be computed using five dot products and a combination of $\min$ and $\max$ operations, as described in more detail in [4]. For the computation of descender lines, we replace dist(L_B, p) by $\min(\text{dist}(L_B, p), \text{dist}(L'_B, p))$, where L'_B is the bow-tie shaped region swept out by the line of descenders in parallel with the baseline (see [5] for more detail).

The second technique that makes implementing geometric matching problems using branch and bound methods simple and efficient is the use of matchlists. That is, for each box B of line parameters, we maintain a list of all and only the alignment points that make non-zero contributions to the quality function Q. We call this list the “matchlist”. When the box B gets subdivided, only alignment points on the matchlist need to be considered.

Up to this point, this section has been a review of prior work on globally optimal line finding. Let us now turn to the question of how we introduce geometric obstacles into this framework to text line finding. When finding text lines with obstacles, we do not allow matches in which a text line model $l_{r,\theta,d}$ intersects an obstacle. This is illustrated in Figure 5. The figure shows two candidate lines (dashed). One line avoids the obstacle and matches points from both sides. Another line matches points on one side of the obstacle better, but cannot “pick up” alignment points on the other side of the obstacle. In fact, in the constrained textline finding problem, solutions are textline segments, not infinite lines.

Perhaps surprisingly, incorporating obstacles into the branch-and-bound textline finding algorithm is simple and does not noticeably increase the complexity of the algorithm on problems usually encountered in practice. The approach is as follows. During

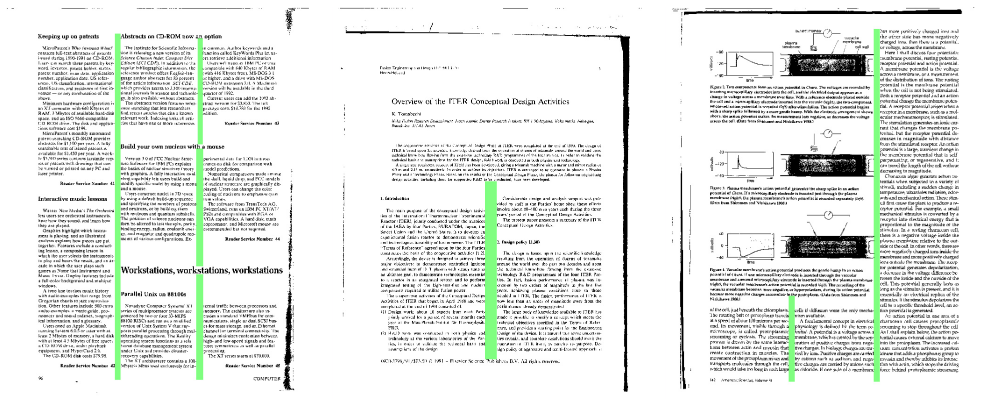

Fig. 7. Examples of the result of whitespace evaluation for the detection of column boundaries in documents with complex layouts (documents A00C, D050, and E002 from the UW3 database). Note that even complex layouts are described by a small collection of column separators.

the branch-and-bound evaluation, we consider successively smaller boxes of line parameters B. When these boxes are large, some of the lines implied by their parameters may intersect an obstacle and some may not. However, as the boxes of parameters get smaller and smaller, at some point, the lines corresponding to these parameter values will either all intersect an obstacle or will all fail to intersect an obstacle. In the case that all lines fail to intersect an obstacle, we simply remove the obstacle from further considerations in subsequent subdivisions of that box of parameters. In the case where all lines intersect an obstacle, we split the set of potentially matching alignment points into two subsets, those to the left of the obstacle and those to the right of the obstacle. We then continue the search with the same box B of line parameters and two separate matchlists, the matchlist for the alignment points to the left of the obstacle, and the matchlist for the alignment points to the right of the obstacle. The algorithm is given in pseudo-code in Figure 6.

This approach to line matching with obstacles uses the matchlists not just as an optimization, but also to structure the search and remove points from further consideration. The line segments that the algorithm finds are implicitly defined by the set of alignment points on a matchlist, the obstacles, and the line. This is a considerably more efficient approach than if we had attempted a search in the space of line segments directly. For finding obstacle-free line segments with baselines, this would have been a search over a five-dimensional parameter space, while the approach based on restricting matchlists requires only a search in the original three-dimensional space of parameters. As a result, using this approach, text line finding with obstacles runs in approximately the same amount of time as text line finding without obstacles.

4 Layout Analysis

So far, this paper has presented two geometric algorithms potentially useful in the implementation of document image analysis systems. The algorithm for the computation of whitespace covers can be used as an easy-to-implement drop-in replacement for the method used in [8]. In that work, rectangles with certain aspect ratios are preferred, and, overall, larger whitespace rectangles are preferred to smaller ones. Their evaluation function is based on statistical measurements on the distribution of whitespace rectangles in real documents, and it is intended to favor those rectangles that are meaningful horizontal or vertical separators.

To test the performance of evaluation functions based on area, aspect ratio, and position on the page, the whitespace coverage algorithm described above was applied to character bounding boxes obtained from document images in the UW3 database. For each document image, a collection of the 200 largest whitespace rectangles with pairwise overlap of less than 80% were extracted. This resulted, as expected, in a collection of whitespace rectangles that almost always completely covered the background, plus additional whitespace rectangles that intruded into text paragraphs. To arrive at a layout analysis, an evaluation function is needed that permits us to select only the rectangles whose union makes up the whitespace that isolates the components of the document layout.

To obtain such an evaluation function, a decision tree was trained to estimate the probability that a given whitespace rectangle is part of the page background. No formal evaluation of the performance was attempted, but the visual inspection showed that a significant fraction of the documents in the UW3 database could not be segmented fully using this approach. As reported in [8], tall whitespace rectangles were usually classified correctly, but for wide whitespace rectangles (those separating paragraphs or sections from one another), a significant number of positive and negative errors occurred. Ittner and Baird's system copes with these issues by computing the wide whitespace rectangles but ignoring spurious wide rectangles until later processing stages (they are not counted as incorrect in the evaluation of their method). Furthermore, visual inspection suggested that there were no rules or evaluation functions based just on the shape of the whitespace rectangles alone that would work reliably in all cases–the UW3 database contained such a diversity of documents that there were inherent ambiguities.

This means that, while evaluation functions based on the shape of whitespace rectangles alone may be useful and reliable for somewhat document collections, for very heterogeneous collections, we probably need another approach. Taken together, these results suggested taking an approach that classifies tall whitespace separately and that takes into account features other than just the shape and position of the whitespace rectangle in its evaluation. Furthermore, several observations suggest that wide whitespace, while sometimes visually salient, is neither necessary nor sufficient for the layout analysis of a document along the vertical axis. For example, paragraph breaks are indicated in many US-style documents by indentation, not additional whitespace, transitions from document headers to body text are most reliably indicated by changes in alignment (centering, left justification, right justification), and some section headings are indicated not by extra spacing but by changes in font size and style.

This then leads to the following four-step process for document layout analysis:

1. Find tall whitespace rectangles and evaluate them as candidates for gutters, column separators, etc.
2. Find text lines that respect the columnar structure of the document.
3. Identify vertical layout structure (titles, headings, paragraphs) based on the relationship (indentation, size, spacing, etc.) and content (font size and style etc.) of adjacent text lines
4. Determine reading order using both geometric and linguistic information.

The key idea for identifying gutters, which we take to mean here tall whitespace rectangles that are a meaningful part of a layout analysis, is to take into account, in addition to the shape and position of the rectangles, their proximity to neighboring text. This constraint is suggested both by document structure, as well as the observation that in a simple maximal white rectangle algorithm, many of the rectangles identified will be bordered only by a few textual components near their corners. Based on considerations of document layouts and readability, we can tentatively derive some rules that we would expect to apply to gutters (in future systems, we intend to base these constraints on statistical properties of pre-segmented document databases):

– gutters must have an aspect ratio of at least 1:3
– gutters must have a width of at least 1.5 times of the mode of the distribution of widths of inter-word spaces
– additionally, we may include prior knowledge on minimum text column widths defined by gutters
– gutters must be adjacent to at least four character-sized connected components on their left or their right side (gutters must separate something, otherwise we are not interested in them)

To test the feasibility of the approach, these rules were encoded into a whitespace evaluation function and the whitespace cover algorithm was applied to finding gutters on pages. To evaluate the performance, the method was applied to the 221 document pages in the "A" and "C" classes of the UW3 database. Among these are 73 pages with multiple columns. The input to the method consisted of word bounding boxes corresponding to the document images. After detection of whitespace rectangles representing the gutters, lines were extracted using the constrained line finding algorithm. The results were then displayed, overlayed with the ground truth, and visually inspected. Inspection showed no segmentation errors on the dataset. That is, no whitespace rectangle returned by the method split any line belonging to the same zone (a line was considered "split" if the whitespace rectangle intersected the baseline of the line), and all lines that were part of separate zones were separated by some whitespace rectangle. Sample segmentations achieved with this method are shown in Figure 7.

5 Discussion and Conclusions

This paper has presented two geometric algorithms. The first algorithm finds globally optimal solutions to the n-maximum empty rectangle problem in the presence of rectangular obstacles, under a wide class of quality functions (including area). The second algorithm finds globally optimal maximum likelihood solutions to the textline finding problem in the presence of obstacles. Both algorithms are easy to implement and practical and have uses in a variety of document analysis problems, as well as other areas of computational geometry.

These algorithms form the basis for an approach to document layout analysis that concentrates on the two arguably most salient and important aspects of layout: gutters (whitespace separating columns of text) and maximal segments of text lines that do not cross gutters. Paragraphs and other layout structure along the vertical dimension can then be found in a subsequent step. Applying this method to the UW3 database suggests very low segmentation error rates (no errors on a 223 page sample). The results also suggest that a description of pages in terms of column separators, text lines, and reading order, is a very compact and stable representation of the physical layout of a page and may be a better goal for the initial stages of layout analysis than traditional hierarchical representations.

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. H. S. Baird, S. E. Jones, and S. J. Fortune. Image segmentation by shape-directed covers. In *Proceedings of the Tenth International Conference on Pattern Recognition, Atlantic City, New Jersey*, pages 820–825, 1990.
3. Thomas M. Breuel. Fast Recognition using Adaptive Subdivisions of Transformation Space. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, pages 445–451, 1992.
4. Thomas M. Breuel. Finding Lines under Bounded Error. *Pattern Recognition*, 29(1):167–178, 1996.
5. 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.
6. 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.
7. William Wells III. Statistical approaches to feature-based object recognition. *International Journal of Computer Vision*, 21(1/2):63–98, 1997.
8. D. Ittner and H. Baird. Language-free layout analysis, 1993.
9. 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.
10. J. Liang, I. T. Philips, and R. M. Haralick. An optimization methodology for document structure extraction on latin character documents. *Pattern Analysis and Machine Intelligence*, pages 719–734, 2001.
11. M. Orlowski. A new algorithm for the largest empty rectangle problem. *Algorithmica*, 5(1), 1990.
12. T. Pavlidis and J. Zhou. Page segementation by white streams. In *1st ICDAR, Saint-Malo*, pages 945–953, 1991.
13. J. P. Trincklin. *Conception d'un systéme d'analyse de documents.* PhD thesis, Thêse de doctorat, Université de Franche-Compté, 1984.