BACKGROUND STRUCTURE IN DOCUMENT IMAGES

HENRY S. BAIRD

AT&T Bell Laboratories
600 Mountain Avenue, Room 2C-322
Murray Hill, New Jersey 07974-0636 USA

ABSTRACT

A method for analyzing the structure of the white background in document images is described, along with applications to the problem of isolating blocks of machine-printed text. The approach is based on computational-geometry algorithms for off-line enumeration of maximal white rectangles and on-line rectangle unification. These support a fast, simple, and general heuristic for geometric layout segmentation, in which white space is covered greedily by rectangles until all text blocks are isolated. Design of the heuristic can be substantially automated by an analysis of the empirical statistical distribution of properties of covering rectangles: for example, the stopping rule can be chosen by Rosenblatt's perceptron training algorithm. Experimental trials show good behavior on the large and useful class of textual Manhattan layouts. On complex layouts from English-language technical journals of many publishers, the method finds good segmentations in a uniform and nearly parameter-free manner. On a variety of non-Latin texts, some with vertical text lines, the method finds good segmentations without prior knowledge of page and text-line orientation.

1. Introduction

We describe an algorithm for the segmentation of images of machine-printed pages into isolated blocks (or, columns) of text. Our approach is motivated by two conjectures. The first is: *white space is a generic layout delimiter*. That is, publishers and printers in many languages use white space in similar ways to separate blocks of text, since they are constrained by common printing technology [CMS82] and nearly universal conventions of legibility. The second conjecture is: *background is simpler than foreground.* By this we mean that the global structure of white space is easier to analyze automatically than local relations among black components. Note that the first conjecture concerns cultural factors, and the second is more narrowly technical. As stated, they are rather vague: in an effort to make them precise enough to be convincingly tested, we have carried out research into algorithms that automatically analyze the structure of white space.

This research program has led to an algorithm with the following useful characteristics.

1. *Generic and nearly parameter-free.* The method does not require detailed *a priori* models of layout style expressed in a large number of application-specific parameters. It relies instead on generic layout properties and, as we shall show, achieves

good results on a large and useful class of layouts even when prior knowledge of the layout styles consists only of loose bounds on text size.

2. *Relatively insensitive to text-line orientation.* The method performs equally well on writing systems whose text-line orientation is vertical (*e.g.* Chinese, Japanese, and Korean) and horizontal (*e.g.* Latin, Greek, and Hebrew). In fact, it often copes well on layouts with mixed orientations.

3. *Oblivious to page orientation.* The method behaves identically, and so produces identical segmentations, whether the page is rightside up, upside down, rotated ±90°, or even mirror-imaged. Thus it can reliably process streams of images of uncontrolled orientation (*e.g.* as commonly occurs with FAXes). (Of course it also copes with skew and shear angles of ±10° or more from horizontal and vertical: this is a separate technical challenge met by other means.)

Other features of the approach include:

(a) a strong reliance on computational-geometry algorithms with unambiguous semantics, with the result that the heuristics built on them are relatively simple and easy to analyze;

(b) a methodology for semi-automatic design and analysis of the heuristics, guided by empirical distributions of elementary numerical properties of covering rectangles; and

(c) asymptotically and absolutely fast implementations.

An early stage of this research was reported in [BJF90]; in the present paper, we describe a complete implementation of the computational-geometry algorithms, a methodology for design and analysis of the heuristics, and the results of experimental trials.

The task is defined precisely in Section 2, and comparable prior work is discussed in Section 3. The geometric algorithms that undergird the heuristics are described in Section 4. The design and analysis of the heuristics are given in Section 5. Experimental trials are reported in Section 6, and conclusions in Section 7.

2. Segmentation of Pages into Text Blocks

We now describe the problem to be solved and its place within a larger strategy for reading printed pages. We define the *text-block segmentation* problem as follows:

Given a bilevel image of a Manhattan page layout,
find a partition of the image area such that
each text block is isolated and no text line is fragmented.

A *Manhattan* page layout is, informally, one in which all text blocks can be isolated, after skew and shear correction, by a set of horizontal and vertical straightline-segments drawn through white space. Note that this does not restrict the blocks to be rectangles: a block can be non-convex, and even non-simply connected (containing holes), as long as it is delimited by boundaries made up of vertical and horizontal edges.

An *isolated text block* is a set of parallel spaced-apart lines of machine-printed or typewritten text symbols (characters). Two adjacent lines are said to be *spaced apart* by the perpendicular distance between their baselines (in the case of horizontally oriented text lines) or centerlines (in the case of vertically oriented text lines). A text line spacing that is exactly equal to the nominal text size (1 em) of the text above or below it, whichever is larger, is called *set solid*. This is the tightest spacing that we will allow (and the tightest recommended by typeface designers). As a result, within an isolated text block, no two text lines' symbols will substantially overlap when projected parallel to their baselines (or centerlines), although of course we expect and allow parts of a few symbols, such as ascenders and descenders, to overlap (and even touch).

Within our page reader system [Bai92], text-block segmentation is executed after connected components analysis and skew and shear correction [Bai87], and before segmentation of text blocks into text lines [Bai88]. Our method for segmenting text blocks into text lines projects each block parallel to its baseline (or, centerline) and analyzes the resulting one-dimensional profile. For this reason, isolation of text blocks in the sense defined above is essential for good results. This is the essence of our notion of an isolated text block: it is a set of one or more text lines which can be effectively found by projection.

We expect geometric segmentation to provide merely the minimum preconditions for successful symbol recognition: a hierarchical decomposition of the page into blocks of text lines of symbols. For this purpose we consider it critical that symbols be correctly assigned to text lines: that is, text lines should never be merged or fragmented. However, the assignment of text lines to text blocks is not critical. We do not, for example, require that the text lines in an isolated block be ''homogeneous'' in any sense, such as possessing the same typeface, size, or line-spacing. Thus a layout will, in general, have more than one good segmentation, with more or fewer text blocks than may intuitively seem ''best'' when the functional structure of the document is taken into account.

We stress that, in this paper, we restrict our attention to *geometric* (or, *physical*) segmentation, and are not attempting *functional* (or, *logical*) labeling of the parts of the document: *e.g.* identifying titles, paragraphs, footnotes, etc, determining reading order among blocks, and so forth. Our research strategy at present is to defer functional labeling until after symbol recognition.

We will assume that text sizes lie within known bounds [*smin*,*smax*] (*e.g.* [6,24] point), but otherwise they may vary arbitrarily within the page. We focus here for the most part on purely *textual* layouts, and will make no performance claims for layouts that contain line-graphics, half-tone photographs, or dirt. However, we have not systematically excluded these from our trials. Our present preprocessing discards† components that are much too large or too small to be text symbols. Text in tabular layouts, or spaced apart within text lines by more than the normal word-spacing, can be problematic for all published geometric segmentation methods; our method often produces usable results, but we are not ready to make more specific claims.

† This *filtering* rule is: discard a connected component if its bounding box's width and height do not both lie in the range $[0.25\,smin, smax]$.

In spite of these restrictions, the class of layouts that we attack is large and useful: it includes most typewritten documents and professionally-printed magazine, book, and journal pages in which text predominates.

3. Prior Approaches

The layout segmentation problem is surveyed in [Nad84] and [SZ86]. An earlier stage of this research was reported in [BJF90], which critically reviews the relevant literature.

Kise *et al.* [KSBT89] appear to be the first to have segmented layouts by an explicit analysis of white space; their approach is dependent on application-specific layout models. Spitz [Spi90] located ‘‘white streams,’’ in both vertical and horizontal directions, and exploited them as generic layout delimiters. Pavlidis & Zhou [PZ91] have described fast algorithms that follow a similar strategy.

The prior work that has most influenced ours is by Nagy *et al.* (*e.g.* [NKKTV88]), who used X-Y trees – representing alternating X-parallel and Y-parallel linear cuts – in both top-down and bottom-up strategies for segmentation. Our rectangular ‘‘covers’’ (defined in the next section) are best understood as generalizations of their X/Y cuts: (a) a cut must act either horizontally or vertically but not both, while a cover embodies sets of line-segment cuts in both directions; and (b) a cut must immediately fragment some region to be used in an XY tree, while a cover doesn’t have to, in our strategy of incremental unification, since it can combine with others. As a result, our covers can, in principle, segment a larger class of layouts than is possible using XY trees. Another difference is that Nagy *et al.* used application-specific layout models to guide the segmentation process, and so in effect carried out geometric and functional analysis simultaneously.

Most fully-automatic methods described in the literature may be divided into two broad categories: *top-down* and *bottom-up* [SZ86]. A top-down control strategy typically starts by hypothesizing a series of interpretations at a high level (*e.g.* that the page has a header above two blocks), and attempts to verify each by a search of a tree of implied hypotheses at lower levels of detail, finally consulting evidence at the lowest level (*e.g.* pixels or connected components). In difficult applications, top-down methods are often robust but slow. A typical bottom-up strategy starts by merging evidence at the lowest level of detail (*e.g.* forming words from symbols, sometimes by smearing or smoothing) and then rises, merging words into lines, lines into blocks, *etc.*, until the page is completely assembled. In difficult applications, bottom-up methods are often fast but unreliable. Almost all reported methods, of both kinds, require detailed application-specific layout models for good results; one fascinating exception is O’Gorman’s bottom-up method using *k*-nearest-neighbors [O92].

Our approach does not fall strictly into either category: rather, it is characterized by a greedy application of statistical estimation, first globally and then locally, without backtracking. Thus, the earliest decisions are made using the broadest statistical support, and thus with high confidence, in hopes that backtracking can be minimized. Such a *global-to-local* strategy often combines the robustness of top-down methods with the speed of bottom-up methods.

4. Computational Geometry Foundations

The first step in our strategy is to analyze the white background into a set of white rectangles (called *covers*) whose union cover it completely. Some of these covers will be large and of a shape that makes it likely that they lie between text blocks rather than between symbols in the same text line. We make this likelihood explicit by defining a sort order on the covers. The Boolean union of the covers' interiors, applied in this order, gradually cover more and more of the white background; at each stage, connected components within the remaining uncovered parts are candidate text blocks. At some point, we decide to stop the unification process, and report the final segmentation. Much of the computation is carried out by the following two geometric algorithms.

4.1. *Off-line enumeration of maximal white rectangles.*

The input to this algorithm is a set of black rectangles†, which in this application represent the bounding boxes of black connected components, after skew and shear correction and filtering to remove extremely large and small components. The output is the set of all maximal white rectangles implied by the input.

A rectangle is *maximal white* if its interior is white and it cannot be further expanded while staying all white. Clearly a maximal white rectangle touches black or the edge of the image on each of its four sides. The enumeration algorithm is described in [BJF90]: it sorts the black rectangles, sweeps through them left to right, and writes out maximal white rectangles as they are found. Its runtime is $O(n \log n + m)$, where n is the number of input black rectangles and m is the number of output maximal white rectangles. In the worst case m can be $O(n^2)$, but on layouts we observe empirically that it is, on average, $O(n)$ (in fact, typically $\leq 4n$). That the runtime is only slightly superlinear is critical in this application since the algorithm must run to completion, generating all m maximal white rectangles. Concrete runtimes are discussed below in Section 6.

The maximal white rectangles are sorted (in a manner described later), and the resulting sequence is input to the next algorithm.

4.2. *On-line unification of rectangles.*

The input to this algorithm is a sequence of rectangular covers $< c_i >_{i=1,\ldots,k}$. The output is a corresponding sequence $< s_j >_{j=0,\ldots,k}$ of segmentations resulting from the Boolean union of the interiors of the covers seen so far:

$$s_j \;=\; s_0 \,\cup\, \bigcup_{i=1}^{j} interior(c_i)$$

The algorithm is on-line in the sense that each segmentation s_j is output before the next cover c_{j+1} is read (see Figures 3, 4, and 5).

A segmentation is a partition of the plane into covered and uncovered regions. In this application, we consider each uncovered component to be a candidate text block, and

† Throughout this paper, all rectangles will be *Manhattan*: that is, with sides parallel to the X- and Y-axes. The synonyms *rectilinear, isothetic,* and *iso-oriented* also occur in the literature.

we initialize the algorithm a segmentation s_0 in which the infinite region surrounding the page has been covered (this is the region outside the shrink-wrapped bounding box that contains all the black input). Thus s_0 possesses a single candidate block which contains the original page area. After all of the covers have been unified all of the white background will be covered, and thus segmentation s_m's uncovered region represents exactly the union of the interiors of all *n* black input rectangles, and this is true no matter in what order the covers are unified. For all other intermediate segmentations s_j, $0 < j < m$, what candidate blocks its owns depends on the order of unification.

In our implementation, a segmentation is represented as an unordered set of candidate blocks each of which need not, of course, be rectangular or simply connected. Each candidate block is described by one exterior boundary and one or more interior boundaries. Each boundary is represented explicitly, at the full resolution with which the image was digitized, as a set of connected vertices and edges. Edges are conventionally directed so that the interior of the uncovered component is to the left. Each edge is linked to its two vertices, and each vertex to up to four edges (left, right, up, and down). As unification proceeds, all intersection points are made explicit as vertices and a representation of minimum size is maintained: edges lying in the interior of the component are deleted, vertices with exactly two collinear edges are deleted, and coincident edges are combined. After each unification, the data structure permits the number of resulting components (which intermittently rises) to be counted, and the boundaries to be extracted in detail. As a useful side-effect, we also assign the input black rectangles to the candidate blocks that contain them.

Let e_j be the number of edges in segmentation s_j. In the worst case, $e_j = O(j^2)$. However, we have observed empirically that $e_j = O(j)$ on average.

Worst-case runtime to unify cover c_j with segmentation s_{j-1}, giving s_j is $O(e_j(t_j + u_j + v_j))$, where t_j is the number of intersection points detected, u_j is the number of edges created or deleted, and v_j is the number of ‘‘hanging’’ vertices having no neighboring vertex directly above it and no adjacent left or up edge. To discover all intersection points requires $O(e_j)$ time. To insert each new point or edge into their sorted sets takes $O(e_j)$ time at worst. Detecting which edges and vertices are redundant or multiple is done using a sweep-line traverse of the sorted vertex set, for a worst-case time of $O(e_j(1 + v_j))$ time altogether (each hanging vertex requires an $O(e_j)$ time search). In the worst case s_j and t_j are $O(e_j)$, but on layouts they are $O(1)$ on average. Similarly, $v_j = O(e_j)$ worst case, but in practice we observe sublinear growth, close to $O(\sqrt{e_j})$. Thus the expected time to unify cover c_j is $O(e_j^{1.5}) = O(j^{1.5})$.

The expected runtime to compute all the first *k* segmentations appears empirically to be superquadratic but not cubic in *k*. Since, in the application, not all *m* covers are unified (*k* is usually a small fraction of *m*), and through the use of coding speed optimizations, the algorithm does not exhibit unacceptably large runtimes on layouts with fewer than 10,000 symbols. Concrete runtimes are discussed below in Section 6.

Both of these geometric algorithms have: (a) precisely specified semantics; (b) verifiably correct implementations; (c) fixed-point integer arithmetic only; and (d) numerical stability.

The computational-geometry literature contains several algorithms for off-line unification (e.g. [SV85]), but we have found no references to algorithms for on-line unification. There are a number of data structures [PS85] with better worst case asymptotic behavior than ours for at least some of the operations we perform, but they appear to be more complex than ours and do not appear to have been implemented, so it is an open question whether they would be absolutely faster in this context.

5. Design and Analysis of a Heuristic

The segmentation heuristic is: unify covers in a certain order until a certain stopping rule applies. We have experimented with many ordering‡ and stopping rules: we will now describe the most effective of these, and then summarize the methods by which it was discovered.

The *ordering rule* has two elements: a *sort key* and a *trimming rule*. The sort key $K(c)$ of cover c is computed as follows:

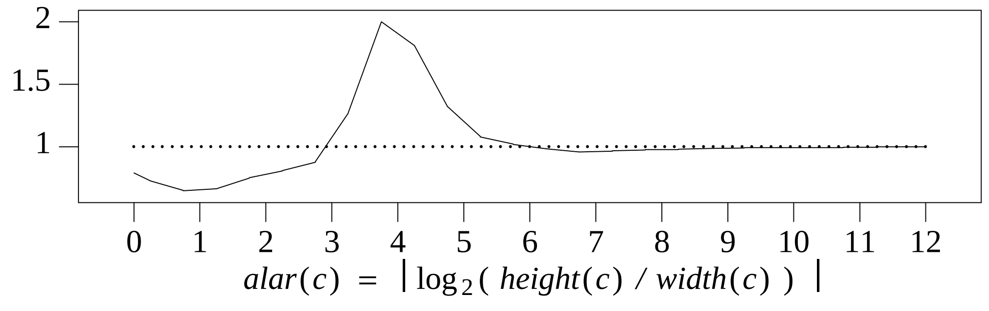
$$K(c) \;=\; \sqrt{\, area(c) \,*\, W(\, alar(c)\,)\,} \quad .$$

The sort is non-increasing on K, implemented using a MAX heap. We express the area $area(c)$ of a cover in a resolution-independent way in typographical units of square points. The square root merely ensures that K is in units of distance (points), which makes the data more convenient to visualize: it has little or no effect on the performance of the algorithm.

For convenience, we have coined the term *alar* for the absolute value of the base-2 logarithm of aspect ratio,

$$alar(c) \;=\; \left|\, \log_2(\, height(c) \,/\, width(c)\,)\, \right|$$

which quantifies the elongation of covers in a dimensionless and orientation-independent way. For 8.5×11 inch images scanned at 400 pixels/inch, $alar(c)$ can take on values in the range [0,12], the value 0 representing squares, and 12 the thinnest possible elongated rectangles (either horizontal or vertical). $W(.)$ is the dimensionless weighting function shown in Figure 1.



‡ We discarded these sort keys as clearly inferior: $area(c)$, $area(c)*(1+alar(c))$, $area(c)*(1+lg(min(hgt(c)/wid(c),16)))$, $area(c)*(1+lg(1+lg(min(hgt(c)/wid(c),16))))$, $sqrt(area(c)*W(alar(c)))$, and $min(hgt(c),wid(c))*(1+alar(c))$.

Figure 1: The dimensionless weighting function $W(alar(c))$.

This function ensures that, among all covers c having the same area, those with $alar(c)$ in the range 3-6 will be unified earlier that the others. In effect, it prefers covers that are ‘‘moderately long and thin’’ (with orientation-independent aspect ratios of 8:1 through 64:1), and penalizes both ‘‘squarish’’ covers and extremely thin ‘‘slivers’’.

The *trimming rule* requires that, before a cover is unified with a candidate block, it is first intersected with the block’s bounding rectangle (not, in general, with its boundary). Often, the trimmed cover is smaller: if its new sort key decreases, then the trimmed cover is pushed back on the heap to be reconsidered later in order. This is intended to avoid early fragmentation of narrow blocks.

Finally, the *stopping rule* is defined as a predicate function of two numerical properties of segmentations. The first is $K(s_j)$ which simply equals the sort key $K(c_j)$ of the last cover unified in making segmentation s_j: the sequence $<K(s_j)>_j$ decreases nonstrictly to 0 as $j \longrightarrow m$. The second is $F(s_j) \ = \ j/m$, the fraction of all covers used so far in making segmentation s_j: the sequence $<F(s_j)>_j$ increases nonstrictly to 1 as $j \longrightarrow m$. The heuristic stopping rule is:

$$\textit{Stop at segmentation } s_j \textit{ when}: \ K(s_j) - 42.43 * F(s_j) \ \leq \ 34.29 \ . \qquad (1)$$

Note that all the rules are defined independently of page orientation and digitizing resolution. Figure 2 illustrates the heuristic’s behavior on a complex layout.

Figure 2: The behavior of the heuristic on a journal article page layout. On the left is the preprocessed (filtered and skew- and shear-corrected) input in the form of black rectangular bounding boxes of the connected components. On the right is segmentation s_{65}, with $K = 28.5$ and $F = 4.7\%$; the uncovered regions are shown in white. All 13 text blocks are correctly isolated and no text lines are fragmented.

Now, in Figures 3-5, we step through a sequence of segmentations of this layout, pointing out significant events.

Figure 3: Three bad ‘‘early’’ segmentations: not all blocks have been isolated yet. On the left is s_0: only the region surrounding the page has been covered, and no covers have yet been unified. In the middle is s_2 (3 blocks, $K = 166.1$, $F = 0.01\%$). On the right is s_{13}: this is the last ‘‘early’’ segmentation (5 blocks, $K = 87.5$, $F = 0.4\%$).

Figure 4: Three good segmentations: all blocks are correctly isolated, and no text line has been fragmented. On the left is s_{25}: this is the first good segmentation (6 blocks, $K = 69.4$, $F = 0.8\%$). Compare s_{65}, the segmentation chosen by the stopping rule, with this and the one in the middle, s_{275}: this is still a good segmentation, although there are more blocks than would be chosen manually (45 blocks, $K = 17.6$, $F = 13.8\%$). On the right is s_{421}: this is the last good segmentation before a text line is fragmented (62 blocks, $K = 13.0$, $F = 32.2\%$).

Figure 5: Three bad ''late'' segmentations, in which at least one text line has been fragmented. On the left is s_{1276}: this is the first bad segmentation (63 blocks, $K = 8.5$, $F = 31.2\%$). Note the first fragmentation: in the third text line in the first paragraph in the leftmost block in the bottom third of the page. In the middle, s_{2198}, with many fragmented text lines (127 blocks, $K = 6.3$, $F = 39.2\%$). On the right is s_{9998} ($K = 0.15$, $F = 99.9\%$), which is close to the last possible segmentation; note that it is approaching the inverse of the black input (Figure 2, left).

5.1. *Designing the Stopping Rule.*

When the sequence of segmentations is graphed in the (K,F) plane, an interesting ''trajectory'' is revealed (Figure 6).

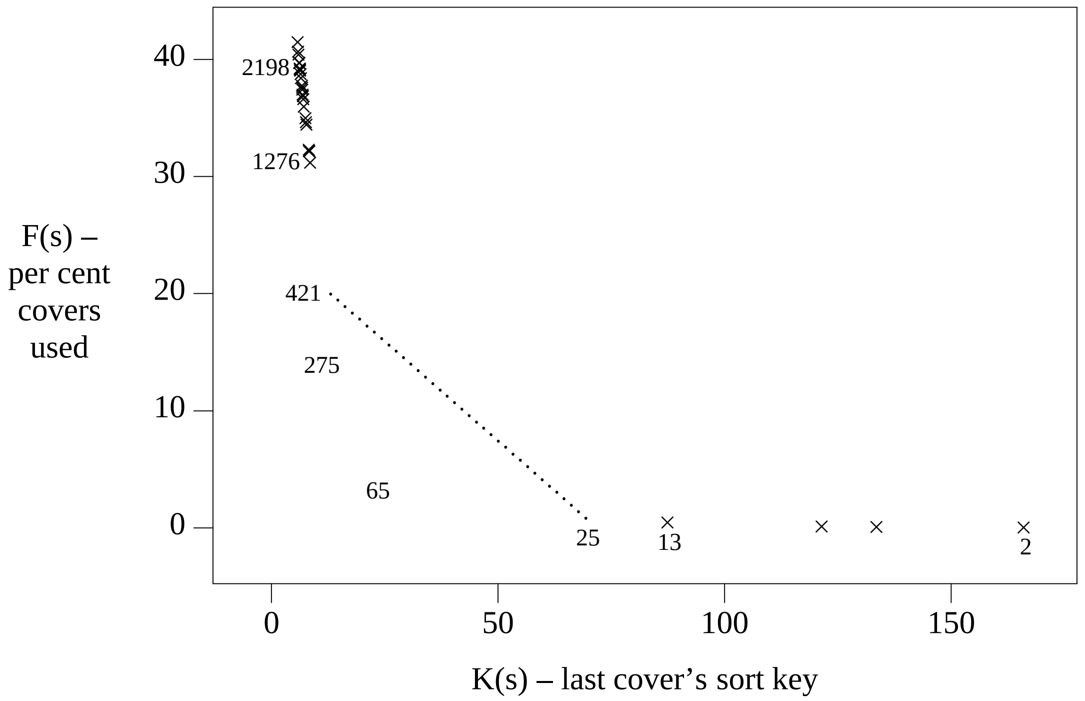


Figure 6: The trajectory described in the (K,F) plane by the segmentations of the layout of Figure 2. Good segmentations are marked , and bad segmentations are marked ×. Several segmentations s_j, chosen from among those in Figures 3-5, are also marked with their index j. The first and last good segmentations are connected by a straight dotted line. For clarity, not all segmentations are shown.

Along this trajectory, K and F change non-strictly monotonically. Draw a dotted line segment between the first and last good segmentations s_{first} and s_{last}: here, between s_{25} and s_{421}. It and the sub-trajectory consisting only of good segmentations form the outline of a convex region. Thus any straight line that crosses this dotted line segment must also cross the trajectory of good segmentations. If we view such a line as a linear inequality discriminant, and stop at the first segmentation along the trajectory that crosses it, then we see that *every linear discriminant that separates s_{first} from s_{last} defines a good stopping rule for the layout.*

This insight leads directly to an effective procedure for choosing the best stopping rule for any given set of layouts: first, find s_{first} and s_{last} for each layout (if they don't exist, then no stopping rule exists for that layout); then, find a linear inequality discriminant that separates as many as possible of the s_{first} points from the s_{last} points. This stopping rule will then operate correctly for the greatest possible number of layouts in the given set.

Finding linear discriminant functions for two-class problems in the plane is a well-studied problem: we use a randomized version of Rosenblatt's perceptron training algorithm [R63]. Figure 7 illustrates the results on a training set of 90 journal layouts.

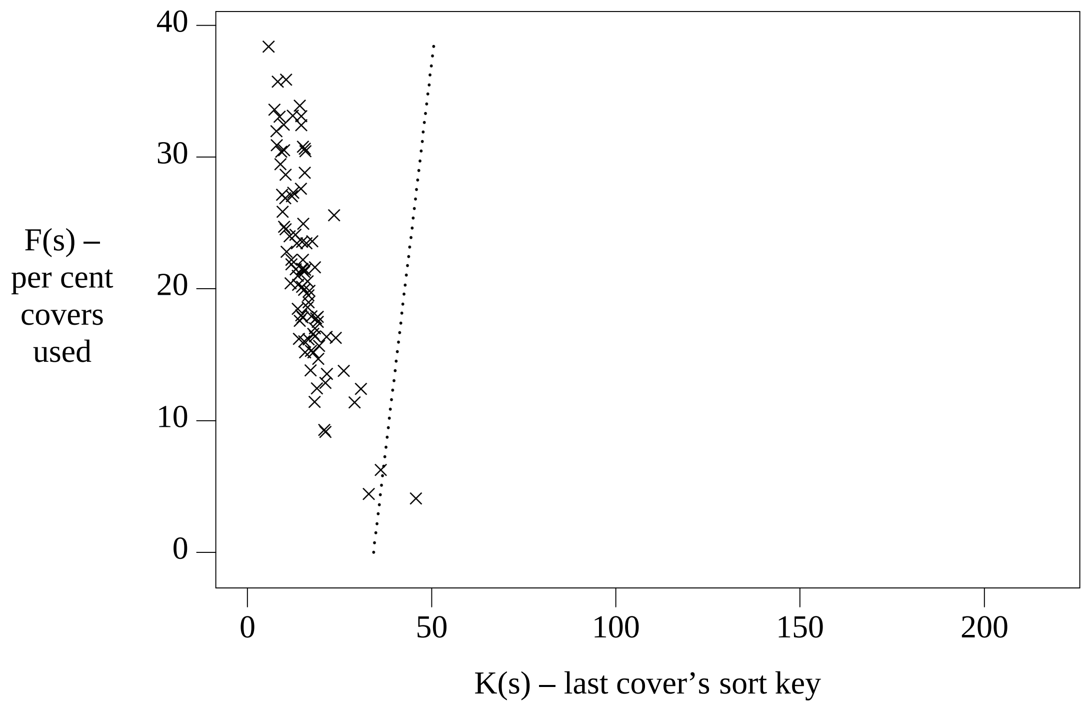


Figure 7: Choosing the stopping rule: the dotted line shows a linear inequality discriminant (corresponding to Equation (1)), selected by a randomized perceptron training algorithm. These data are from 90 journal layouts that are feasible under the ordering rule. First good segmentations are marked and last good segmentations are marked $\times$. The discriminant separates as many as possible (in fact, all but two).

We call the reader's attention to the fact that the two classes are, with few exceptions, well separated. This striking evidence of robustness suggests that the method is in a practical sense generic.

5.2. *Designing the ordering rule.*

We now give a brief narrative of the evolution of the ordering rule. At the outset, it seemed reasonable to try $key(c) = area(c)$, but this turned out to be too crude: column gutters and other important ''long and thin'' covers sorted late, so that the first good segmentations were delayed. So we next tried $key(c) = area(c) * (1 + alar(c))$, and were pleased to see a striking improvement — but now, extremely thin covers sorted early, so that fragmentation of text lines occurred prematurely.

It was clear to us that we needed an ordering rule which emphasized ''moderately'' thin covers only; but how to choose such a rule? We have experimented with an interesting empirical bootstrapping method, as follows. For a training set of layouts, and using $key(c) = area(c) * (1 + alar(c))$, we generated all covers up through the last good segmentation. Here is the empirical density function of $alar(c)$ among these ''early and good'' covers.

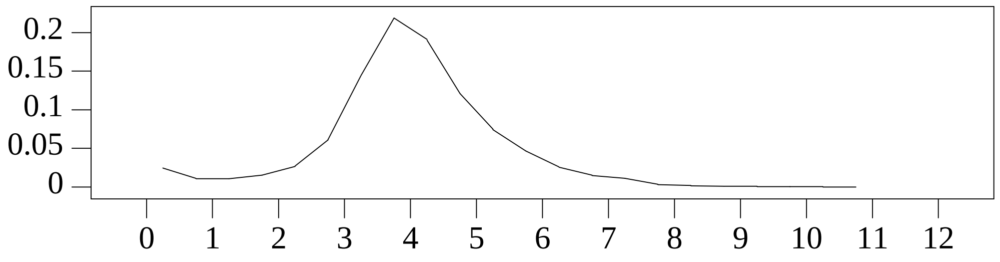


Figure 8: Density of $alar(c)$ among 114143 ''early and good'' covers, ordered by $key(c) = area(c) * (1 + alar(c))$.

Averaged over all the layouts, these included 20% of the total number of covers. We then examined the empirical density function of $alar(c)$ among a comparable set: the first 20% of each layout's covers, but this time ordered using the crude $key(c) = area(c)$.

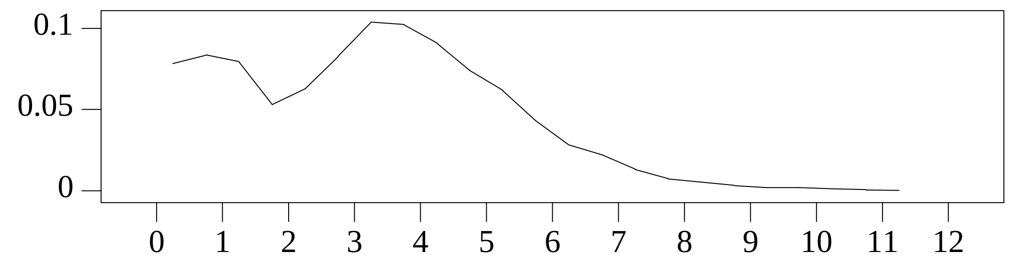


Figure 9: Density of $alar(c)$ among 249673 ‘‘20% by area’’ covers, ordered by $key(c) = area(c)$.

We know from experiment that the first of these $alar(c)$ densities resulted from a better ordering rule than the second. Thus we expect any discrepancies between them to reveal something about which values of $alar(c)$ are more helpful. In an attempt to quantify this, we subtract them, giving the discrepancy function in Figure 10.

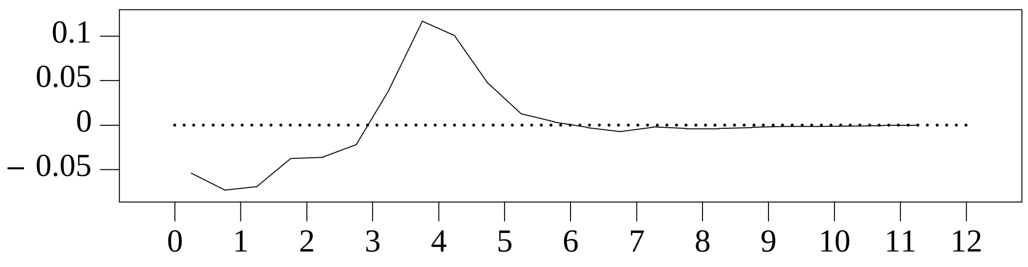


Figure 10: A discrepancy function resulting from the subtraction of the ‘‘20% by area’’ density (Figure 9) from the ‘‘early and good’’ density (Figure 8).

This reveals a systematic difference of the kind that we hoped to make explicit. We convert this into the weighting function $W(alar(c))$ by exponentiating and scaling so that the largest absolute weight is 2.0. The resulting $key(c) = area(c) * W(alar(c))$ has proved, in experimental trials, to be a clear improvement over the others. Subtracting the densities and scaling the result are, admittedly, arbitrary choices: we do not know enough to claim that they will work on other cases.

6. Experimental Trials

We will report on two trials. The first is a systematic trial on 100 pages from archival journals (see Appendix A), representing thirteen publishers and at least 22 distinct layout styles. Of these, 96 were feasible: that is, had at least one good segmentation under the ordering rules. The remaining four layouts failed for the same reason, illustrated in Figure 11.

Figure 11: An example of an infeasible layout under the ordering rule. The two short blocks side-by-side at the bottom are isolated from the others first by horizontal covers; then the trimming rule prevents them from being separated until too late.

Figure 12: Fragmented text lines resulting from an unusually ragged margin. Less troubling is the isolation of vertically aligned labels in indexed lists.

The stopping rule fails on two of the 96 feasible layouts, for an overall success rate of 94%. One of these problems is similar to Figure 11. The other is shown in Figure 12; this layout also exhibits the tendency of the method to break away ‘‘item-list’’ labels that are vertically aligned; this does not usually present a problem to symbol recognition.

We have also run trials on an unsystematic collection of non-Latin layouts, including some with vertical text lines. Since our method is intrinsically independent of page orientation, it tends to work equally well on these; Figures 13 and 14 give examples.

Figure 13: This Korean layout has most, but not all, of the text in vertical text lines. Blocks of both orientations are isolated correctly. Note that one text block is non-convex.

Figure 14: This Chinese layout, in addition to both vertical and horizontal text lines, has a calligraphic (hand printed) section. As a result, some of the ‘‘white streams’’ are not exactly horizontal or vertical. A subtle cut between two narrow blocks of vertically oriented text is missing.

Runtime on the journal pages averaged 1.8 CPU s, with a maximum of 3.1, including the computational geometry algorithms, heuristics, and sorts.

7. Discussion

We have described a method for segmenting into text blocks entire images of pages of text in machine-printed Manhattan layouts. Experimental trials have demonstrated three important practical advantages:

1) the method is generic and nearly parameter-free, requiring no detailed *a priori* layout model (it does need to know loose bounds on text size);

2) it is relatively insensitive to text-line orientation; and

3) it is completely oblivious to page orientation.

In addition, it runs fast, both absolutely and asymptotically. Almost all of the runtime is consumed in unambiguously specified computational geometry algorithms whose implementations can be guaranteed correct. The residual heuristics are simple enough that they can be empirically and semi-automatically designed and analyzed.

The text blocks that are isolated need not be rectangles: they can be non-convex (*e.g.* Figure 13) and even non-simply connected. They must, of course, be delimited by boundaries made up of vertical and horizontal edges, but there is no predetermined limit on the number of edges, and so a wide variety of non-rectangular shapes is allowed. Also, the ‘‘white streams’’ that separate blocks need not be exactly or extensively horizontal and vertical for good results (*e.g.* Figure 14), although it undeniably helps.

The heuristics are probably improvable. For example, it may help to make them more sensitive to text size, whether specified by the user or inferred statistically within blocks on the fly. Also, the present trimming rule is implicated in many failures: it too should perhaps adjust itself to the statistics of the layout. Alternatively, deliberate overcutting followed by a local merging heuristic may clear up some problems. It is likely that the manual effort required for training can be reduced: for example, given page images with ground-truthed zoning, the identification of good and bad segmentations can probably be automated. It would be interesting to carry out a large-scale systematic trial on non-Latin scripts generally, and specifically on pages containing both vertically and horizontally oriented text lines. The reasons for the heuristic’s good performance remain mysterious, but we feel that some aspects may be amenable to formal analysis.

The success of this research program suggests to us that the two conjectures which motivated it — that white space is a generic layout delimiter, and that background is simpler than foreground — often hold true in practice. Their implications deserve to be explored further.

8. Acknowledgements

We are grateful for stimulating discussions with David Ittner, George Nagy, Larry O’Gorman, Larry Spitz, Theo Pavlidis, and Susan Dorward. Special thanks also to Larry Spitz for the Kise reference, and to Larry O’Gorman and Jim Arnold for helping me access the RightPages database for the Journals trial.

9. References

[Bai87] H. S. Baird, ‘‘The Skew Angle of Printed Documents,’’ *Proceedings, SPSE 40th Conf. and Symp. on Hybrid Imaging Systems*, Rochester, New York, pp. 21-24, May 1987.

[Bai88] H. S. Baird, ‘‘Global-to-Local Layout Analysis,’’ *Proceedings of the IAPR 1988 Workshop on Syntactic and Structural Pattern Recognition,* Pont-a-Mousson, France, September 1988.

[Bai92] H. S. Baird, ‘‘Anatomy of a Versatile Page Reader,’’ *Proceedings of the IEEE*, Vol. 80, No. 7, 1992 [in press].

[BJF90] H. S. Baird, S. E. Jones, and S. J. Fortune, ‘‘Image Segmentation by Shape-Directed Covers,’’ *Proc., 10th ICPR*, Atlantic City, NJ, June 16-21, 1990.

[CMS82] *The Chicago Manual of Style*, 13th Edition, The University of Chicago Press, Chicago, Illinois, pp. 696-701, 1982.

[KSBT89] K. Kise, J. Sugiyama, N. Babaguchi and Y. Tezuka, ‘‘Layout Model Based Analysis of Document Structure,’’ *Journal of the Institute of Electronics, Information and Communication Engineers*, Vol, J72-D-11 No. 7, pp 1029-1039, 1989 (in Japanese).

[Nad84] M. Nadler, ‘‘A Survey of Document Segmentation and Coding Techniques,’’ *Computer Vision, Graphics, and Image Processing*, vol. 28, pp. 240-262, 1984.

[NKKTV88] G. Nagy, J. Kanai, M. Krishnamoorthy, M. Thomas, and M. Viswanathan, ‘‘Two Complementary Techniques for Digitized Document Analysis’’ *Proc., ACM Conf. on Document Processing Systems*, Santa Fe, New Mexico, December 5-9, 1988.

[O92] L. O’Gorman, ‘‘The Document Spectrum for Bottom-Up Page Layout Analysis,’’ this proceedings.

[PS85] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag (New York, 1985).

[PZ91] T. Pavlidis and J. Zhou, ‘‘Page Segmentation by White Streams,’’ *Proceedings, 1st Int’l Conf. on Document Analysis and Recognition*, St-Malo, France, pp. 945-953, September 1991.

[R63] F. Rosenblatt, ‘‘A Bibliography of Perceptron Literature,’’ In *Collected Technical Papers*, Vol. 2, *Cognitive Systems Research Program*, Report No. 4, Cornell University, July 1963.

[Spi90] A. L. Spitz, ‘‘Recognition Processing for Multilingual Documents,’’ In R. Furuta (ed.), *Proceedings of the 1990 Int’l Conf. on Electronic Publishing, Document Manipulation and Typography*, Gaithersburg, Maryland, pp. 193-205, September 1990.

[SV85] T. G. Szymanski and C. J. Van Wyk, ‘‘GOALIE: A Space Efficient System for VLSI Artwork Analysis,’’ *IEEE Design and Test of Computers*, Vol. 2 No. 3, pp. 64-72, June 1985.

[SZ86] S. N. Srihari and G. W. Zack, ‘‘Document Image Analysis,’’ *Proceedings, 8th Int’l Conf. Pattern Recognition*, Paris, France, pp. 434-436, October 1986.

Appendix A. The Journals Trial

The ‘‘Journals’’ experiment was conducted on 100 pages of archival technical journals of 13 publishers, representing at least 22 distinct layout styles. Only complex Manhattan layouts were chosen: all have more than one text block, many have variable numbers of blocks at various places on the same page, and a few have nested layouts that cannot be described using XY trees. For each journal, a single issue was chosen (except for *Comm. ACM*, for which two were used, representing two distinct layout styles). The pages contain predominantly text, and were digitized at 300 and 400 pixels/inch using flatbed document scanners. Mathematical equations, tabular data, line-graphics, and half-tone photographs do occur, but are not systematically represented. The images contain no color and no half-tone background under text. For the convenience of researchers who may wish to replicate the trial, we give the bibliographic reference for each page.

For each of the following journal issues, three regular articles and three correspondence items were chosen and their title pages scanned.

IEEE Transactions on Computers, Vol. 40, No. 3, March 1991, pp. 245, 253, 263, 315, 320, 325 (IEEE).
IEEE Transactions on Information Theory, Vol. 37, No. 6, November 1991, pp. 1501, 1519, 1527, 1641, 1645, 1647.
IEEE Transactions on PAMI, Vol. 13, No. 2, February 1991, pp. 99, 114, 133, 185, 192, 202.

For each of the following journal issues, three regular articles’ title pages were scanned.

CVGIP: Graphical Models and Image Processing, Vol. 53, No. 6, Nov. 1991, pp. 501, 511, 522 (Academic Press).
CVGIP: Image Understanding, Vol. 54, No. 1, July 1991, pp. 1, 47, 56 (Academic Press).
Computer Networks and ISDN Systems, Vol. 21, No. 1, March 1991, pp. 17, 53, 69 (North-Holland).
IEEE 1991 International Solid State Circuits Conference, February 1991, pp. 16, 22, 26.
IEEE Computer Graphics and Applications, Vol. 11, No. 2, March 1991, pp. 20, 29, 39.
IEEE Computer, Vol. 24, No. 2, April 1991, pp. 5, 17, 33.
IEEE Design and Test of Computers, March 1991, pp. 6, 50, 58.
IEEE Software, May 1991, pp. 14, 21, 27.
IEEE Spectrum, April 1991, pp. 28, 74, 77.
Information Processing Letters, Vol. 37, No. 3, February 1991, pp. 121, 127, 133 (North-Holland).
Neural Networks, Vol. 4, No. 2, April 1991, pp. 131, 185, 193 (Pergamon Press).
Physics Today, Vol. 44, No. 4, April 1991, pp. 24, 32, 40 (American Institute of Physics).
Proceedings of the IEEE, Vol. 79, No. 3, March 1991, pp. 253, 278, 308.
Systems Integration, Vol. 24, No. 3, March 1991, pp. 22, 28, 45 (Cahner's).

For each of the following journal issues, a single regular article was chosen, and three pages of it were scanned: the first page (usually the title page), one of the last pages (usually with references and/or biographies), and one other page (usually with footnotes or extensive equations).

ACM Computing Reviews, Vol. 33, No. 5, May 1992, pp. 223, 228, 232 (ACM Press).
ACM Computing Surveys, Vol. 23, No. 1, March 1991, pp. 5, 7, 47 (ACM Press).
AT&T Technical Journal, Vol. 70, No. 1, Jan/Feb 1991, pp. 21, 31, 35 (AT&T Corporation).
CVGIP: Image Understanding, Vol. 53, No. 1, January 1991, pp. 1, 2, 13 (Academic Press).
Communications of the ACM, Vol. 33, No. 3, March 1990, pp. 281, 283, 293 (''old style,'' ACM).
Communications of the ACM, Vol. 33, No. 7, July 1990, pp. 21, 25, 26 (''new style,'' ACM).
Computer Networks and ISDN Systems, Vol. 21, No. 1, March 1991, pp. 17, 18, 29 (North-Holland).
Distributed Computing, Vol. 4, No. 2, 1990, pp. 59, 67, 68 (Springer International).
NRC NewsReport, Vol. 42, No. 3, Apr/May 1992, pp. 2, 3, 4 (U.S. National Research Council).
Pattern Recognition Letters, Vol. 7, No. 1, January 1988, pp. 1, 3, 8 (North-Holland).
Pattern Recognition, Vol. 24, No. 1, 1991, pp. 1, 5, 8 (Pergamon Press).
The Bridge, Vol. 21, No. 2, Summer 1991, pp. 4, 7, 9 (U.S. National Academy of Engineering).
The Computer Journal, Vol. 35, No. 1, February 1992, pp. 3, 12, 15 (Cambridge University Press).

To make up an even 100 pages, the following single page was also included:

Scientific American, June, 1989, p. 122.