Decomposing Rectilinear Figures into Rectangles

Ritu Chadha and Donald C. S. Allison

TR 88-17

DECOMPOSING RECTILINEAR FIGURES INTO RECTANGLES

Ritu Chadha and Donald C. S. Allison

Department of Computer Science

Virginia Polytechnic Institute and State University

Blacksburg, VA 24061.

Abstract

We discuss the problem of decomposing rectilinear regions, with or without holes, into a minimum number of rectangles. There are two different problems considered here : decomposing a figure into non-overlapping parts, called *partitioning* , and decomposing a figure into possibly overlapping parts, called *covering*. A method is outlined and proved for solving the above two problems, and algorithms for the solutions of these problems are presented. The partitioning problem can be solved in time $O(n^{5/2})$, where n is the number of vertices of the figure, whereas the covering problem is exponential in its time complexity.

Introduction

We discuss the problem of decomposing a rectilinear figure into a minimum number of rectangles. A *rectilinear figure* is a polygon in which each edge is either parallel or perpendicular to a given direction. Decomposing a figure into rectangles may be done either by allowing the rectangles to overlap, or by requiring that the rectangles of the decomposition be disjoint. A decomposition of a figure into overlapping rectangles is called a *cover* and a decomposition of a figure into non-overlapping rectangles is called a *partition* of the figure. Our goal in this work is to establish a method for finding rectangle covers and partitions for a given rectilinear figure which consist of the smallest possible number of rectangles.

A topic of significant recent interest is the decomposition of polygons into simpler components such as convex polygons, star-shaped polygons, spiral polygons, convex quadrilaterals, and rectangles. The motivation for such studies can be ascribed to the existence of a large number of algorithms which can be applied to these simpler components, but not to general polygons. Thus certain problems can be solved by applying efficient specialized algorithms to the component parts of the decomposed polygon. Another application, which is concerned with decomposing figures into rectangles, is the use of such a decomposition for the efficient creation of masks for photolithography using pattern generators which can only print rectangles. The processing time is obviously proportional to the number of rectangles and thus minimum decompositions of polygons are needed. The decomposition of polygons into simpler components is also of use in pattern recognition schemes, since an arbitrary polygonal shape is recognized more easily once its component parts have been identified.

According to Keil [4], partitioning problems have received more attention than covering problems in the past. In general, minimally partitioning a polygon can be done in less time than minimally covering it. It has been generally observed that NP-hard algorithms arise more frequently for covering than for partitioning problems, although a few partitioning problems have been found to be NP-hard, as described by O'Rourke and Supowit [9].

A number of polygon covering problems have been studied in the past. In 1979, Masek [8] proved that the problem of finding the minimum rectangle cover for a rectilinear polygon is NP-complete. Thus many researchers concentrated their attention on covering problems for more restrictive classes of polygons, such as horizontally or vertically convex polygons. Others studied decompositions of general polygons into subsets of different shapes, such as convex, star-shaped, or spiral subsets.

Partitioning problems more often deal with the decomposition of general polygons into a minimum number of simpler components. Lipski et. al. [7] presented a method for finding a partition of a rectilinear figure into a minimum number of rectangles, where the rectilinear figure is originally specified as a union of m rectangles. The time taken by the algorithm they developed is $O(m^3)$. Keil [3] developed polynomial time algorithms for decomposing a polygon into a minimum number of simpler components such as convex polygons, spiral polygons, star-shaped polygons, and monotone polygons. Johnson [2] gave a brief overview of the advances made in these areas in his quarterly column in the Journal of Algorithms.

This paper presents an algorithm for deriving a minimum rectangle partition and a minimum rectangle cover for a rectilinear figure with or without holes. The approach here differs from that adopted by Lipski et. al. [6, 7] in the way in which the rectilinear figure to be partitioned/covered is described. Lipski et. al. assumed that the figure to be partitioned is given as the union of a number m of rectangles, whereas in this work we describe a given rectilinear figure by specifying the x and y coordinates of its vertices relative to a pair of perpendicular axes. The time complexity of their partitioning algorithm, which was $O(m^3)$, is improved here to $O(n^{5/2})$, where n is the number of vertices of the figure.

Theoretical discussion

This section discusses the methods used to derive a minimum rectangle partition and a minimum rectangle cover for a given rectilinear figure.

Minimum rectangle partition

Definitions, notation and terminology : A **partition line** is a line drawn in the interior of a figure F, parallel to at least one of the sides of F.

A **created vertex** is an extremity of a partition line which is not a vertex of the original figure.

Two edges of a figure are said to be **collinear** if they can be connected by a straight line parallel to both the edges and internal to the figure.

A vertex of a rectilinear polygon is said to be **concave** if the angle at the vertex (measured inside the polygon) is 270 degrees. A vertex which is not concave is called **convex**.

A partition line is said to be a **concave** line if it joins two concave vertices.

If F is a rectilinear polygon, we let n, h, x, v, r denote the number of vertices, holes, convex vertices, concave vertices, and created vertices respectively of F.

Lemma I.: $x - v = 4 - 4h$. (1)

Proof : We know that for the external contour of the figure, the number of convex vertices exceeds the number of concave vertices by 4, while the converse is true for the contour of any hole. Hence, summing (x - v) over the external contour and the contours of all holes, we get $x - v = 4 - 4h$. ■

Corollary I.

$x = n/2 - 2h + 2$ (2)

$v = n/2 + 2h - 2$ (3)

Proof : Solving (1) and the identity $x + v = n$ simultaneously, we get (2) and (3). ■

Theorem I. Let P be the number of rectangles in any partition of a figure F, where F is a figure without collinear edges. Then

$P \geq n/2 + h - 1$ (4).

Proof : Let F be partitioned into P rectangles, and let r denote the number of created vertices in F. Every concave vertex must be the extremity of at least one partition line; the other extremity of this line must be a created vertex (since F has no collinear edges). Hence

$r \geq v$ (5).

Also every vertex of F is a vertex of at least one rectangle of the given partition, and every created vertex of F is a vertex of at least two adjacent rectangles of the given partition; thus the total number of vertices of all the rectangles in the partition $= 4P \geq n + 2r \geq 2n + 4h - 4$, using (5) and (3). Dividing by 4, the result follows. ∎

Theorem II. Let F be a figure without collinear edges. Then there exists a partition of F into P rectangles with $P = n/2 + h - 1$.

Proof : By construction. For every concave vertex, draw a horizontal partition line to a vertical edge. The convex and concave vertices of the figure are the vertices of one rectangle of the partition, and the created vertices are the vertices of exactly two adjacent rectangles of the partition. Thus $4P = n + 2r$; however, by our construction, $r = v$, therefore $4P = n + 2v = 2n + 4h - 4$ using (3), i.e. $P = n/2 + h - 1$. ∎

Corollary II. The minimum number of rectangles into which a rectilinear polygon without collinear edges can be partitioned is $n/2 + h - 1$. (6).

Proof : From theorems I and II. ∎

We now turn to the problem of partitioning a figure F with collinear edges into a minimum number of rectangles.

Define a function M of two variables n and h as follows :

$$M(n,h) = n/2 + h - 1,$$

where

n = number of vertices of the figure

h = number of holes of the figure.

Consider any concave line joining concave vertices a and b in F. Drawing a partition line from a to b has the following effect :

Case (i): Both vertices a and b belong to the external contour of F or to the contour of the same hole in F : The figure F is partitioned into two smaller figures, F_1 and F_2. If F_1 has n_1 vertices and h_1 holes, then F_2 has $n-n_1$ vertices and $h - h_1$ holes. Hence the sum of the values of the function M for F_1 and F_2 is :

$$M(n_1,h_1) + M(n-n_1,h-h_1) = n/2 + h - 2 = M(n,h) - 1.$$

Case (ii): a belongs to the external contour of F and b belongs to the contour of a hole, or a and b belong to the contours of different holes : The figure F is transformed into a new figure F′ which has $h - 1$ holes and n vertices. Hence the value of the function M for figure F′ is $M(n,h-1) = = n/2 + h - 2 = M(n,h) - 1.$

Now, we transform the figure F into one or more figures without collinear edges by drawing concave lines. From the above analysis, the sum of the values of the function M for the new figures obtained by drawing concave lines decreases by 1 for each concave line drawn; hence if we draw a total of 'C' concave lines, the sum of the values of the function M for the resulting figure(s) will be

$$M(n,h) - C \qquad (7)$$

Each resulting figure now has a minimum partition into $n'/2 + h' - 1 = M(n',h')$ rectangles, where n′ and h′ are the number of vertices and holes, respectively, of the figure (by Corollary II). Hence, summing the number of rectangles in the minimum partitions of each figure into

which F has been divided (which equals M(n', h')), we get M(n,h) - C (by (7)) = n/2 + h - 1 - C as the minimum number of rectangles into which F can be partitioned.

Thus to minimize the number of rectangles in the partition, we need to maximize the value of C, i.e. the number of concave lines used to partition the figure into a number of figures without collinear edges.

Consider a figure F with two intersecting concave lines l_1 and l_2 (a pair of lines is said to **intersect** if they have at least one point in common). Suppose F is partitioned using line l_1. Then l_2 is no longer a concave line for the new figure (see Figure 1 for an example illustrating this). This shows that in order to use a maximum number of concave lines to partition figure F, we must find a set of concave lines of maximum cardinality such that no two lines in this set intersect each other. We have proved

Theorem III. The minimum number of rectangles into which a figure F can be partitioned is n/2 + h - 1 - C where

n = number of vertices of F

h = number of holes of F

C = maximum cardinality of a set S of concave lines, no two of which intersect each other.

Proof : See the preceding discussion. ■

From Corollary II and Theorem III, we have obtained the minimum number of rectangles into which a given figure can be partitioned. The method of construction of this partition follows from the proof of the preceding theorems.

Minimum rectangle cover

We now turn to the problem of finding the minimum number of rectangles, which may overlap, which completely cover a given rectilinear figure F.

Definitions and terminology

A **basic rectangle** of a figure F is a rectangle B which satisfies the following conditions:

(i) $B \subseteq F$

(ii) There is no rectangle P such that

$B \subset P$.

A **rectangle cover** (or simply **cover**) of a figure F is a set of rectangles (possibly overlapping) whose union is the figure F.

A **minimum rectangle cover** (or **minimum cover**) of a figure F is a rectangle cover of F with minimum cardinality.

A **segment** is a line of positive length.

A rectangle R is represented by its lower left and upper right-hand corners; e.g. if a rectangle has its lower left corner lying at (x_1,y_1) and its right upper corner at (x_2,y_2), then we write $R = (x_1,y_1 ; x_2,y_2)$.

Lemma

For any rectangle P contained in a figure F, there exists a basic rectangle $B \subseteq F$ such that $P \subseteq B$.

Proof : Let d be the minimum distance from the right edge of P to an edge of the contour of F lying to the right of P. P can be stretched by an amount 'd' to the right; similarly we can stretch P to the left, upwards, and downwards, to obtain a new rectangle B. Clearly B is a basic rectangle containing P. ∎

Theorem I.

The set S of all basic rectangles is a rectangle cover for F.

Proof : Let a be any point in F. There exists some rectangle P which contains a. By the lemma, there exists a basic rectangle P′ such that $P \subset P'$. Hence $a \in P$. Thus every point in F is covered by some basic rectangle and therefore S covers F. ■

Theorem II. There exists a minimum cover for F consisting solely of basic rectangles.

Proof : Let S be any minimum cover of F. By the lemma, for every rectangle $P \in S$, we can find a basic rectangle such that $P \subset P'$; hence the union of these basic rectangles covers the figure F and forms a minimum cover of F. ■

Hence we will take the set of all basic rectangles of F (which covers F by Theorem I) and eliminate as many rectangles as possible from this union to get a minimum basic rectangle cover for F. Our strategy is to break up the figure into smaller rectangular subregions as follows : for every concave vertex v, draw a horizontal line internal to the figure, joining v to the boundary of the figure; similarly draw a vertical line internal to the figure and joining v to the boundary of the figure. This divides the figure F into a number of subrectangles. We call the resulting partition the **subdivision** of F, and the rectangles of this partition are called **subrectangles** . We now establish some properties of basic rectangles and subdivisions.

Property 1. The left and right edges of a basic rectangle each have a segment in common with some vertical boundary of the figure, and the upper and lower edges of a basic rectangle each have a segment in common with some horizontal boundary of the figure.

Proof : The proof follows from the proof of the lemma. ■

Property 2. Let R be a rectangle lying in the figure, such that each side of R has a segment in common with some boundary of F. Then R is a basic rectangle of F. (Note : this is the converse of Property 1).

Proof : This is obvious from the fact that it is not possible to extend R in any direction, i.e. no rectangle R′ such that $R \subset R'$ can be found in the figure F. ■

Property 3. When two rectangles $R = (x_1,y_1 ; x_2,y_2)$ and $S = (X_1,Y_1 ; X_2,Y_2)$ intersect non-trivially (i.e. they have a point in common which does not lie on the boundary of either of the rectangles), then the following equations hold :

(i) $Y_2 > y_1$

(ii) $y_2 > Y_1$

(iii) $X_2 > x_1$

(iv) $x_2 > X_1$.

Proof : (i) If $Y_2 \leq y_1$, then the entire rectangle S lies below R, and hence they cannot intersect non-trivially. (ii) - (iv) are proved similarly. ■

Property 4. If P is a rectangle of the subdivision of F and B is a basic rectangle, then either

$B \cap P = \emptyset$ or $P \subseteq B$.

Proof : Let $P = (x_1,y_1;x_2,y_2)$ be a rectangle of the subdivision of F, and let $B = (X_1,Y_1;X_2,Y_2)$ be a basic rectangle of F. If $B \cap P = \emptyset$, there is nothing to prove; hence assume that $B \cap P \neq \emptyset$. This means that the conditions (i) to (iv) listed under Property 3 are satisfied. We need to prove that $P \subseteq B$.

Suppose not. Then there exists a point $r = (a,b)$ such that r belongs to P and r does not belong to B. This means that r is either above B, below B, to the right of B, or to the left of B (note that r could simultaneously be above and to the left of B, or below and to the left of B, etc.). Without loss of generality, assume that r is below B (a symmetrical argument follows if r is above, to the left, or to the right of B). If r is below B, then

$$b < Y_1$$

and by (ii), $Y_1 < y_2$

i.e. $b < Y_1 < y_2$

Now, by Property 2, there exists a horizontal boundary of F with y-coordinate Y_1, which has a segment in common with the base of rectangle B. Let H be the boundary nearest to rectangle P with this property. H cannot intersect the rectangle P in a segment (because other-

wise a part of the boundary of F would be passing through P, which means that a portion of the rectangle P would lie outside the figure F, which is a contradiction.). Let $H = (c,d ; Y_1)$. Since H does not intersect the rectangle P (except possibly at a point), it must lie either to the right or to the left of rectangle P. Assume that it lies to the left of rectangle P (see Figure 2).

Then the vertex (d,Y_1) of the figure F must be concave (since the rectangle B lies in F and the lower right corner of B lies to the right of (d,Y_1)). Hence by our method for obtaining a subdivision of F, we must draw a horizontal line internal to the figure, joining the concave vertex (d,Y_1) to a boundary of F; this line will cut through rectangle P, along the lower edge of rectangle B. This leads us to a contradiction, since no line of the subdivision of F can pass through the rectangle P (since P itself is a rectangle of the subdivision of F).

Hence our assumption must be wrong, i.e. there is no point r such that $r \in P$ and $r \notin B$.

Therefore, we have

$$P \subseteq B$$

Thus either

$$P \cap B = \emptyset$$

or $P \subseteq B$. ■

Theorem III. Consider the subdivision of a figure F. For any two points p_1 and p_2 belonging to the same subrectangle,

$$p_1 \in B \iff p_2 \in B$$

for all basic rectangles B of F.

Proof : Let p_1 and p_2 be two points belonging to the same subrectangle P of F, and let p_1 be contained in some basic rectangle B of F. Then $p_1 \in B \cap P$; hence $B \cap P \neq \emptyset$. Therefore by Property 4, we have

$$P \subseteq B.$$

Thus p_2, which belongs to P, also belongs to B; and by symmetry, it is obvious that the converse also holds. ■

In other words, this theorem states that all points in the same subrectangle of the subdivision of F are contained in the same basic rectangle(s) of F. Now in order to obtain a rectangle cover S for F consisting of basic rectangles, every point in F must be covered by at least one basic rectangle in S. However, we know by Theorem III that if even one point of a subrectangle is covered by a basic rectangle, then every point of the subrectangle is covered by the same basic rectangle. Thus we can focus our attention on obtaining a minimum set of basic rectangles such that every subrectangle is covered by at least one basic rectangle from S. This problem can be solved in the same way as the classical covering problem for Boolean functions (see [5]). The method is described below.

Choosing basic rectangles for the minimum cover : Construct a two-dimensional chart A with the same number of rows as the number of basic rectangles of F; each row corresponds to a basic rectangle of F. The number of columns in the chart equals the number of subrectangles of the subdivision of F and each column corresponds to a subrectangle of F. Entries are made in the chart as follows : the entry A[i,j] is set to '1' if and only if the basic rectangle corresponding to row i covers the subrectangle corresponding to column j, and is set to '0' otherwise.

We now process this chart in order to determine which of the basic rectangles of F should be chosen to form a minimum rectangle cover for F. To begin with, we note that if a column j contains only one '1', in row i say, then the basic rectangle B corresponding to row i must be chosen to be part of the minimum cover for F, since it is the only basic rectangle which covers the subrectangle corresponding to column j. Thus row i is removed from the chart, along with every column in which there is a '1' in the i^{th} row, because all these subrectangles are covered by the basic rectangle corresponding to row i. This basic rectangle is chosen to be a rectangle of the minimum cover for F.

We now remove **dominated rows** and **dominating columns** from the chart. A row i is said to be **dominated** by a row i' if and only if row i' has 1's in all the columns in which row i has 1's; in this case row i' dominates row i; the definition for columns is similar. Dominated rows are removed because the basic rectangle corresponding to a row i' dominating a row i covers all the subrectangles covered by row i. Similarly, dominating columns are removed since all the basic rectangles which cover the subrectangle corresponding to a column j also cover the subrectangle corresponding to any column j' dominating column j.

We repeat the above steps as necessary to reduce the chart completely. However, it may happen that the repeated application of these steps does not reduce the chart; in this case, the chart is said to be **cyclic** (see Figure 3 for an example of a cyclic chart). To reduce a cyclic chart, we choose a column j of the chart at random, preferably one with the minimum number m of 1's in it. We now obtain m smaller charts from the cyclic chart, where the i^{th} chart is a copy of the cyclic chart with the row corresponding to the i^{th} '1' in column j removed, along with all the columns in which this row had a '1'. These m charts are now processed by the steps described earlier. One of these charts will ultimately result in the choice of the smallest number of basic rectangles among all m charts; these basic rectangles are then added to the previously chosen basic rectangles for the minimum rectangle cover.

Implementation

The algorithm used to implement the decomposition of a rectilinear figure into a minimum number of overlapping/non-overlapping rectangles is described in this section.

Input and output

The input to this program consists of

(i) The number of vertices in the figure F to be partitioned, and

(ii) The x and y coordinates of the vertices of the figure F.

The output of the program is a decomposition of the figure F into a minimum number of overlapping and non-overlapping rectangles.

Preliminary data sorting

Data input and sorting: The data is read from the input data file and placed in a list in which each element is a vertex (x,y) of the given figure. Assume that the figure F has n vertices. We now sort these n vertices first by x coordinate and then by y coordinate so that the following statements hold :

If F has vertices $v_1, v_2, ..., v_n$, $v_i = (x_i, y_i)$, $1 \leq i \leq n$, then for any $v_i = (x_i, y_i)$ and $v_j = (x_j, y_j)$ with $i < j$, the following statements hold : either

(i) $y_i < y_j$

or

(ii) $y_i = y_j$ and $x_i < x_j$.

The resulting list contains the vertices of the figure ordered from top to bottom, and from left to right in each row of vertices.

Reading in the n vertices is done in linear time, i.e. O(n). The sorting can be done using any conventional sorting technique, such as heapsort, which takes time O(n log n); hence the overall time complexity of this step is O(n log n).

Construction of the edges of the figure : Given the vertices $\{v_1, v_2, ..., v_n\}$ of the figure sorted as just described, the edges of the figure are derived. Define the horizontal neighbor of a vertex v to be the other end of the horizontal edge through v , and the vertical neighbor of a vertex v to be the other end of the vertical edge through v. Initially, none of the vertices have horizontal or vertical neighbors assigned to them. The purpose of this step is to assign vertical and horizontal neighbors to each vertex.

Start examining the vertices of the figure one by one, in the order in which the vertices are found in the list Hvertices. If the vertex v_i being examined has no horizontal neighbor, then its horizontal neighbor is necessarily the next vertex in the ordered list, i.e. v_{i+1} ; the horizontal neighbor of v_i is thus initialized to v_{i+1}, and the horizontal neighbor of v_{i+1} is initialized to v_i. Also, v_i is marked as the left end and v_{i+1} is marked as the right end of the edge. The edge running from v_i to v_{i+1} is placed in a list of horizontal edges called HElist.

If $v_i = (x_i, y_i)$, has no vertical neighbor, then the vertices v_{i+1}, v_{i+2} , ... are scanned until a vertex $v_j = (x_j, y_j)$ $(j \geq i + 1)$ is found such that $x_i = x_j$. This vertex will then be the vertical neighbor of v_i. The vertices v_i and v_j are thus assigned their vertical neighbors (v_i is the vertical neighbor of v_j and vice-versa) and the edge connecting v_i and v_j is placed in a list of vertical edges called VElist. v_i is marked 'down' (because it is the lower end of the vertical edge) and v_j is marked 'up' (because it is the upper end of the vertical edge). In this manner we obtain a list of horizontal edges (HElist) and a list of vertical edges (VElist); also each vertex is assigned a horizontal neighbor and a vertical neighbor, and is marked left/right and up/down according as the vertex is the left or right end of a horizontal edge and the upper or lower end of a vertical edge.

Finding the horizontal neighbor of a vertex can be done in constant time, since the horizontal neighbor of a vertex v_i is either v_{i-1} or v_{i+1}; hence all the horizontal neighbors of the n vertices can be found in O(n) time. Finding the vertical neighbor of a vertex, however, can take $O(n^2)$ time, since we may need to scan all the vertices v_{i+1}, v_{i+2}, ... , v_n to find the vertical neighbor of a vertex v_i; therefore finding the vertical neighbors of all the vertices of the figure can take time upto $O(n^2)$, which is the overall time complexity of this step.

Derivation of the contour of the figure : The contour of the figure is really a union of contours consisting of the external contour of the figure along with the contours of the holes in the figure. To derive the external contour of the figure, we need to create a list (named EXT) whose elements are the vertices of the external contour of the figure in the order in which the vertices are visited while moving around the external contour of the figure in some fixed direction ; for the purpose of consistency we choose the counterclockwise direction. This is done as follows.

Let the first element of the list EXT be the leftmost vertex of the bottom row of vertices in the figure ; this vertex is nothing but the first element of the list of vertices Hvertices. Mark this vertex 'visited' and move right to get the next element of the contour, which is also the second element of the list Hvertices. Mark this vertex 'visited'. The strategy used hereafter is to visit the unvisited neighbor (if any) of the last vertex of EXT, mark it 'visited', and repeat this procedure until the last vertex of EXT has no unvisited neighbors.

The contours of the holes of the figure are derived as above; the first vertex of the contour of a hole of the figure is obtained by scanning the list 'Hvertices' for the first unvisited vertex (if any). The contour of the figure is completely derived when all the vertices of the list Hvertices have been marked 'visited'.

To derive the contour of the figure and of the holes of the figure, we need time $n(h + 1)$. This is because the list Hvertices (or a part of the list Hvertices) has to be scanned once for the derivation of each contour. There are h contours of holes and one external contour; hence the number of scans of list Hvertices is $n(h + 1)$, or $O(nh)$.

Finding concave vertices : To determine which vertices of the figure are concave, we examine the vertices in the contour of the figure and mark the ones which are concave. Now, every vertex v has a horizontal and a vertical edge incident on it; to determine whether v is convex or concave, we need to determine on which side of these edges the figure lies. This is done by traveling along the contour of the figure in a counterclockwise direction and recording the direction of travel at any particular time. The directions of travel along the horizontal and vertical edges incident on a vertex indicate whether the vertex is convex or concave; e.g. for the directions of travel around the external contour shown in Figure 4, the figure must lie on the side shown shaded. This result can be extended for the contours of holes. Hence to determine whether a vertex belonging to the external contour of the figure is concave, we implement this method : travel along the external contour of the figure (the vertices of the external contour are contained in the list EXT) and keep track of the direction of travel along the vertical and horizontal edges incident on the vertex. The direction of travel is known since each vertex is marked 'left/right' and 'up/down'. The order in which these edges are traversed

is important, since travelling South and then East means that the vertex is convex, whereas travelling East and then South means that the vertex is concave. After every vertex in the external contour has been labelled as convex or concave, we repeat the same procedure for the vertices of all the holes in the figure until all the vertices of the figure F have been labelled as convex or concave.

To carry out this step, each vertex has to be visited once; hence the time taken is O(n).

Sorting the vertices by column. A new list, named Vvertices, is now created. This list holds the vertices of the figure F sorted as follows : if F has vertices $v_1, v_2, \ldots, v_n$; $v_i = (x_i,y_i)$, $1 \leq i \leq n$, then for any $v_i = (x_i,y_i)$, and $v_j = (x_j,y_j)$, with $i < j$, the following statements hold : either

(i) $x_i < x_j$

or

(ii) $x_i = x_j$ and $y_i < y_j$.

To sort the vertices we may use any conventional sorting technique, such as heapsort; the upper bound on the sorting time required is then O(n log n).

Finding a minimum rectangle partition

Finding internal lines joining pairs of collinear edges : The pairs of collinear edges in the figure F (if any) are found as follows. Horizontal collinear edges are found first. Recall that the vertices in the list 'Hvertices' are ordered row-wise, starting from the bottommost row and going from left to right in each row. Thus two edges can be collinear only if there exist four consecutive vertices v_i , v_{i+1} , v_{i+2} , v_{i+3} in Hvertices with the same y coordinate. These four vertices are the extremities of two edges which are collinear. We must now check whether the line joining these two edges is internal to the figure F or not. The line joining these two edges is the segment joining v_{i+1} and v_{i+2} . This segment is internal to the figure F (i.e. is entirely contained within the figure F) if and only if the following two conditions hold :

(i) v_{i+1} and v_{i+2} are concave vertices

(ii) No vertical edge of the figure intersects the segment joining v_{i+1} and v_{i+2} at any point other than v_{i+1} and v_{i+2}.

Condition (i) is checked by verifying that v_{i+1} and v_{i+2} are labelled 'concave' vertices. If (i) is true, then (ii) is checked by scanning the list VElist of vertical edges. A vertical edge running from a point (a,b) to a point (a,c) with $b < c$ will intersect the segment from v_{i+1} to v_{i+2} at a point other than v_{i+1} or v_{i+2} if and only if

$$x_{i+1} < a < x_{i+2}$$

and $b < y_{i+1} < c$ (recall that $y_{i+1} = y_{i+2}$).

The scan of the vertical edges is interrupted if a vertical edge satisfying the above conditions is found; if the scan is completed without finding such an edge, the segment joining v_{i+1} and v_{i+2} is added to a list called 'HorizCol', consisting of horizontal lines connecting collinear edges.

To find internal lines joining pairs of vertical collinear edges, a procedure similar to the one described above is used; the list Vvertices is used instead of the list Hvertices.

When a pair of horizontal or vertical edges is found to be collinear, the lists HElist or VElist have to be scanned in order to verify that the line joining these edges is internal to the figure; thus the time taken for each pair of collinear edges is O(n) (since the size of the lists HElist and VElist is n/2). Since the number of collinear edges is O(n), the overall time complexity of this step is $O(n^2)$.

Finding a maximum set of partitioning lines: We now have to find a maximum subset of HorizCol ∪ VertCol consisting of lines which do not touch each other. To solve this problem, we make use of graph theory. Consider an undirected graph G in which each vertex represents a line in HorizCol ∪ VertCol, and let an edge e join vertices a and b if and only if the lines

which vertices a and b represent touch each other. Then the problem of finding the maximum number of lines in HorizCol U VertCol which do not touch each other translates into the problem of finding a maximum set of independent vertices in the graph G (a set S of vertices in a graph is said to be independent if and only if there is no edge joining any two vertices in the set S).

Note that the graph G is bipartite (since no two horizontal edges can touch each other and similarly no two vertical edges can touch each other). The method used to find a maximum independent set for G is explained in [1, 7]. After a maximum set has been found, the list HorizCol is updated to contain all the horizontal lines corresponding to this set and similarly VertCol is updated to contain all the vertical lines in the set.

From [1], the time required here is $O(n^{5/2})$.

Completion of the partition: The figure F along with the set of lines contained in the lists HorizCol and VertCol now represent a partially partitioned figure with no collinear edges. Thus we complete the partition of the figure by drawing a horizontal line internal to the figure from every remaining concave vertex to the boundary of the figure. This is done as follows : suppose $v_i = (x_i, y_i)$ is labeled as a concave vertex. We first scan the lists HorizCol and VertCol to check if v_i is an extremity of any line segment in these lists. If it is not, then an internal horizontal line must be drawn from v_i to the boundary of the figure. Now, v_i is either the left end or the right end of a horizontal edge of the figure (recall that every vertex has been marked as a 'left' end or a 'right' end). If v_i is a left end, then since v_i is a concave vertex, the figure F must lie to the left of v_i. Thus an internal horizontal line starting at v_i must be drawn towards the left until the first vertical edge is encountered. This is accomplished by scanning the list of vertical edges VElist and the list of vertical partition lines Vertcol for a line satisfying the following specifications : Let the line joining the points (a,b) and (a,c) with $b < c$ be written as (a ; b, c) and let the line joining the points (a,c) and (b,c) be written as (a, b; c). Then the required line satisfies

$$a = \max \{x_k \mid (x_k ; B, C) \in M \}$$

where $M = \{ (x_k ; B, C) \in VertCol \cup VElist \mid B \leq y_i \leq C , x_k < x_i \}$.

The line (a ; b, c) satisfying these conditions is the nearest vertical edge to the left of v_i. Thus the new partition line is $(a, x_i; y_i)$ which connects the concave vertex v_i to the nearest vertical edge. This line is added to the list HorizCol.

If v is the right end of a horizontal edge, a procedure similar to the one outlined above is followed to obtain a line joining v to the nearest vertical edge/partition line to its right.

The above process is repeated for every concave vertex which is not already an extremity of a line in HorizCol or VertCol.

We now have a complete set of lines (contained in the lists HorizCol and VertCol) which partition the figure into a minimum number of rectangles.

The lists HorizCol and VertCol have to be scanned for each concave vertex. The time required for this is bounded above by n^2 (since the sizes of both lists is O(n) and the number of concave vertices is less than or equal to n). After this, if the concave vertex being examined is not already an extremity of an edge in HorizCol or VertCol, the list of vertical edges VElist is scanned; the time taken for this is O(n) for each vertex, therefore the time taken for all the concave vertices is $O(n^2)$.

Finding a minimum rectangle cover

Separating left and right edges We construct two lists RElist (Right Edge list) and LElist (Left Edge list) containing the right and left vertical edges of the figure F respectively. This is done by using the list EXT and the lists HOLE[1] to HOLE[h] which contain the vertices of the external contour of the figure and of the h holes of the figure (respectively) in the order in which they would be traversed while traveling around F in a counterclockwise direction.

First consider the external contour of F, the vertices of which are contained in the list EXT. If a vertical edge V of F is a right edge, then its lower extremity will be visited before its upper

extremity during a counterclockwise traversal of the figure; and conversely, if V is a left edge of F, then its upper extremity will be visited before its lower extremity during a counterclockwise traversal of the figure. Hence we travel around the external contour of the figure (by scanning the list EXT); we start at the leftmost vertex of the lowest row of vertices and travel counterclockwise. The direction of travel from one vertex to another is alternately horizontal and vertical. When the direction of travel is vertical, we are moving from one extremity of a vertical edge to the other; therefore to determine whether that edge is a left edge or a right edge, we need only know whether the direction of travel is upward or downward. This information is stored in each vertex (since each vertex is marked 'up' or 'down' according to whether it is the upper or the lower end of a vertical edge). Hence in one scan of the list EXT, we can derive a list of right and left edges of the external contour.

A similar method is used for determining the left and right edges of holes, with the difference that if a vertical edge of a hole is traversed in a downward direction during a counterclockwise traversal of the contour (i.e. its upper extremity is visited before its lower extremity) then the edge is a right edge, otherwise it is a left edge.

The construction of the two lists REIist and LEIist is done by scanning every vertex of the figure once; hence the time required is O(n).

Creating the subdivision of the figure : To create the subdivision of the figure, we need to :
(i) Join every concave vertex by an internal horizontal line to the boundary of the figure, and
(ii) Join every concave vertex by an internal vertical line to the boundary of the figure.
This is done as follows :

(i) The list HCOMB will contain the horizontal lines of the subdivision. The list Hvertices of the vertices of the figure is scanned for concave vertices. As soon as a concave vertex is found, the list HCOMB is scanned to see whether a horizontal line has already been drawn joining that vertex to a boundary of F. If not, we see whether the concave vertex is a left or a right extremity of a horizontal boundary of the figure (this can easily be done since each vertex is marked 'left' or 'right' according to whether it is the left or the right extremity of a horizontal

edge). If it is a left extremity, then the horizontal line of the subdivision which is to be drawn must extend to the right of the vertex v. Thus we scan the list of vertical edges VElist for the first vertical edge to the right of vertex v which intersects the horizontal line drawn through v. The line joining v to this vertical edge is then added to HCOMB. A similar procedure is followed if v is the right extremity of a horizontal edge of the figure.

The above process is repeated for every concave vertex of the figure until every concave vertex has been joined to the boundary of the figure by a horizontal line internal to the figure.

(ii) The list VCOMB will contain the vertical lines of the subdivision. The list Vvertices of the vertices of the figure is scanned for concave vertices. As soon as a concave vertex is found, the list VCOMB is scanned to see whether a vertical line has already been drawn joining that vertex to a boundary of F. If not, we see whether the concave vertex is an upper or a lower extremity of a vertical boundary of the figure (this can easily be done since each vertex is marked 'up' or 'down' according to whether it is the upper or the lower extremity of a vertical edge). If it is an upper extremity, then the vertical line of the subdivision which is to be drawn must extend above the vertex v. Thus we scan the list of horizontal edges HElist for the first horizontal edge above vertex v which intersects the vertical line drawn through v. The line joining v to this horizontal edge is then added to VCOMB. A similar procedure is followed if v is the lower extremity of a vertical edge of the figure.

The above process is repeated for every concave vertex of the figure until every concave vertex has been joined to the boundary of the figure by a vertical line internal to the figure.

After lists HCOMB and VCOMB have been created, the horizontal edges of the figure are added to the list HCOMB, and the vertical edges of the figure are added to the list VCOMB. Before the vertical edges of the figure are added to the list VCOMB, we first make a copy of the list VCOMB and store it in a list named LVCOMB, and add the left edges of the figure to LVCOMB. Thus the list HCOMB now contains the horizontal edges of the figure, along with the horizontal lines drawn joining each concave vertex to the boundary of the figure; the list VCOMB contains the vertical edges of the figure, along with the vertical lines drawn joining

each concave vertex to the boundary of the figure; and the list LVCOMB contains the left edges of the figure along with the vertical lines drawn joining each concave vertex to the boundary of the figure.

We now sort these lists as follows. The list HCOMB is sorted so that for any pair of edges HCOMB[i] = $(x_1, x_2; y_1)$ and HCOMB[j] = $(X_1, X_2; Y_1)$ with $i < j$, we have either

(i) $y_1 < Y_1$

or (ii) $y_1 = Y_1$ and $x_1 < X_1$.

The list VCOMB is sorted so that for any pair of edges VCOMB[i] = $(x_1; y_1, y_2)$ and VCOMB[j] = $(X_1; Y_1, Y_2)$ with $i < j$, we have either

(i) $x_1 < X_1$

or (ii) $x_1 = X_1$ and $y_1 < Y_1$.

The list LVCOMB is sorted in the same way as the list VCOMB.

Scanning the lists HCOMB and VCOMB takes time O(n) for each, since the lengths of these lists cannot exceed n. Since these scans are performed at most once for every concave vertex, the overall time complexity is bounded above by n^2. Sorting the lists HCOMB, LVCOMB, and VCOMB takes time O(n log n); hence the time complexity of this step is $O(n^2)$.

Constructing the basic rectangles of the figure : We now construct a list of basic rectangles of the figure F. Recall from Property 1 that the left side of a basic rectangle has a segment in common with one or more left edges of the figure. Hence we use the following strategy for constructing basic rectangles. We scan the list LElist of left edges one by one. Let L be a left edge of the figure. We extend L in both directions (upwards and downwards) as follows :

To extend L downwards, scan the list of horizontal edges of the figure (HElist) for the nearest horizontal edge below L which satisfies the following condition : the line H intersects the ver-

tical line through L (i.e. the line L, extended downward if necessary) at a point other than its right extremity.

Similarly, to extend L upwards, scan the list of horizontal edges for the nearest horizontal edge H′ above L satisfying the following condition : the line H′ intersects the vertical line through L (i.e. the line L, extended upward if necessary) at a point other than its right extremity. The reason for thus extending L as far as possible in both directions is to obtain the longest possible line L′ which has a segment in common with a left edge of the figure F and lies inside the figure.

We now begin to scan the list of right edges (RElist) to find the nearest right edge R which lies to the right of the extended edge L′, and which does not lie entirely above or entirely below L′. Once such an edge is found, it is clear that the rectangle with L′ as its left side and with its right side along R is a basic rectangle.

Now, we delete from L′ the projection of R onto L′. If there is no remainder or if the remainder does not contain a portion of L, we are done. Suppose we have one or more portions of L′ left over after the deletion, each containing a portion of L. These leftover edges may form one or more basic rectangles each. Thus we repeat the same procedure as that described above for finding right edges for these leftover portions by looking for edges in RElist which lie to the right of the previously chosen right edge R. We continue like this until there is no portion of L′ left over containing a portion of L.

To extend a line L, we need to scan the list HElist of horizontal edges; since the size of HElist is n/2, this takes O(n) time. After extending L, the list RElist is scanned once, which also takes time O(n) (since the size of RElist is less than n/2). The number of lines such as L is O(n); therefore the time complexity here is $O(n^2)$.

Constructing the subrectangles of the figure : In the previous step, the lines of the subdivision of the figure were placed in the lists HCOMB and VCOMB, along with the edges of the figure; and the list LVCOMB contains the left edges of the figure and the lines of the subdivision. From these lists, we derive a list called SUBRECT which will contain the coordinates of the

lower left and upper right corners of the subrectangles of the subdivision. Every line in the list LVCOMB will be the left edge of one or more subrectangles of F. Thus for every line in LVCOMB, we look for the corresponding edges in HCOMB and VCOMB which will complete the subrectangle. This is done as follows. For each edge in LVCOMB, we search for the first horizontal line H in HCOMB which satisfies the following three conditions :

(i) The line H intersects L at a point p

(ii) p is not the lower extremity of L

(iii) p is not the right extremity of H.

The top edge of the subrectangle being constructed will lie along H, and the lower edge of the subrectangle will be the horizontal line originating from the lower end of L. We now have to find the vertical line which will mark the right edge of the subrectangle. To do this, search the list VCOMB for the nearest vertical line L' lying to the right of L which does not lie entirely above or entirely below L. The subrectangle is now completely formed; its lower left corner is the lower extremity of L, and its upper right corner lies at the intersection of the lines through H and L'. These coordinates are inserted in the list SUBRECT.

Now, if H intersects L at a point p where p is not the upper extremity of L, then there is a portion of L left over which can be used to form a new subrectangle. Call this leftover portion L_1 We proceed in exactly the same manner as before to form a subrectangle with L_1 as its left side (or as a portion of its left side). When there is no 'leftover' from the line L, carry out the same procedure for the next line in list LVCOMB. This process is repeated until every line in the list LVCOMB has been used to form one or more subrectangles. The list SUBRECT now contains the coordinates of the lower left and upper right corners of every subrectangle of the subdivision of the figure F.

From [6], the number of subrectangles of a figure is $O(n^4)$. For every subrectangle formed, we need to scan lists HCOMB and VCOMB once; since the lengths of these lists are O(n), the time required for this step is $O(n^5)$.

Building and processing the cover chart : We now build a two-dimensional cover chart which summarizes the covering relationships between the basic rectangles and the subrectangles of figure F, as explained earlier. The processing of the chart is carried out exactly as described earlier.

To build the chart, we need to scan the list of basic rectangles once for each subrectangle. Since the number of subrectangles is $O(n^4)$ and the number of basic rectangles is $O(n^2)$ (see [6] for proof), the time complexity for building the chart is $O(n^6)$.

To find the time required to process the chart, we note that in the worst case, we will have a cyclic chart to process, which will reduce to a number cn^2 (for some constant c) of smaller charts which are again cyclic, and so on.

Therefore if T(n) denotes the time taken to solve a problem of size n, we have

$T(1) = 1$

$T(n) \leq cn^2T(n - 1)$

$\leq (cn^2)^2T(n - 2)$

$\leq \ldots$

$\leq (cn^2)^{n-1}T(1)$

$= c^{n-1}n^{2n-2}.$

Hence the overall time complexity for this step in the worst case is exponential.

Note that a solution can be found in time $O(2^{cn^2})$ by trying every possible combination of basic rectangles for covering the figure. This time bound is better than that derived above; however

in situations other than the worst case, where we do not have cyclic charts, our chart processing algorithm will be superior to this brute force method.

Examples

The minimum rectangle partitions and covers for some figures which were obtained using this algorithm are shown in Figures 5, 6, 7, 8, 9, and 10.

Conclusions

The results presented in this paper give a complete description of a method for obtaining a minimum rectangle partition and rectangle cover for a rectilinear figure with or without holes. The fact that the number of rectangles in a minimum rectangle cover is less than or equal to the number of rectangles in a minimum rectangle partition may in some cases be overshadowed by the great disparity between the time required for finding minimum rectangle covers and that required for finding minimum rectangle partitions for a rectilinear figure, which have been shown to be exponential and $O(n^{5/2})$ respectively. Thus in certain applications in which a minimum rectangle cover is required, it may be worthwhile to consider using a minimum rectangle partition instead, especially if the problem size is large. In this case, the number of rectangles obtained for covering the figure may be higher than the minimum number of rectangles in a rectangle cover for the figure, but the time taken to find this cover (which is actually the minimum partition) will be considerably less than that required for finding the minimum number of rectangles required to cover the figure.

The time bound of $O(n^{5/2})$ for the algorithm for finding a minimum rectangle partition for a rectilinear figure was achieved by transforming the problem of finding a maximum set of non-intersecting concave lines into that of finding a maximum independent set for a bipartite graph. This in turn was done by using Hopcroft and Karp's algorithm for maximum matchings in bipartite graphs [1], which runs in time $O(n^{5/2})$. Hence the time complexity of the partitioning algorithm described here is dependent on the time complexity of Hopcroft and Karp's algo-

rithm, and would be improved if a faster algorithm than that given by Hopcroft and Karp were developed for finding maximum matchings in bipartite graphs.

It may be possible to improve the time complexity of the minimum rectangle partition algorithm for rectilinear figures without holes. For this restricted class of figures, it may be possible to find a maximum matching algorithm which runs faster than the $O(n^{5/2})$ algorithm developed by Hopcroft and Karp, which would therefore reduce the running time for the partitioning algorithm for this class of figures.

Another field for further research is suggested by Figures 9,10. A special class of figures for which an improved algorithm could be developed is the set of rectilinear figures where the outer contour is a rectangle, each hole is a square, and holes may only appear in certain positions determined by regularly spaced rows and columns.

Finding a polynomial time algorithm for deriving a minimum rectangle cover for a rectilinear figure, however, may not be feasible, since Masek [8] established a proof in 1979 of the fact that this problem is NP-complete. It may thus be desirable to concentrate on developing fast approximation algorithms which will give a near-optimal solution to the problem of deriving a minimum rectangle cover for rectilinear figures.

References

1. J.E. Hopcroft, R.M. Karp, *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs*, SIAM Journal of Computing 2 (1973), pp. 225-231.

2. D. S. Johnson, *The NP-completeness column : An ongoing guide*, Journal of Algorithms, 3 (1982), pp. 182-195.

3. J. M. Keil, *Decomposing a polygon into simpler components*, SIAM Journal on Computing, 14, 4 (1985), pp. 799-817.

4. J. M. Keil, *Minimally covering a horizontally convex orthogonal polygon*, Proc. of the 2nd Annual Symposium on Computational Geometry (1986), pp. 43-51.

5. Z. Kohavi, *Switching and finite automata theory*, McGraw-Hill, New York 1970.

6. W. Lipski, E. Lodi, F. Luccio, C. Mugnai, L. Pagli, *On two-dimensional data organization I*, Fundamenta Informaticae, 2 (1978), pp. 211-226.

7. W. Lipski, E. Lodi, F. Luccio, C. Mugnai, L. Pagli, *On two-dimensional data organization II*, Fundamenta Informaticae, 2 (1978), pp. 245-260.

8. W. J. Masek, *Some NP-complete set-covering problems*, unpublished manuscript (1979).

9. J. O'Rourke, K. J. Supowit, *Some NP-hard polygon decomposition problems*, IEEE trans. on Information Theory, vol. IT-29, 2 (1983), pp. 181-190.

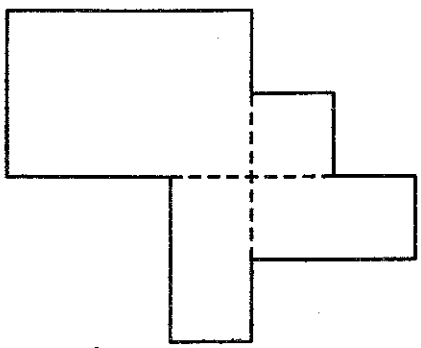

Figure 1. Two intersecting concave lines

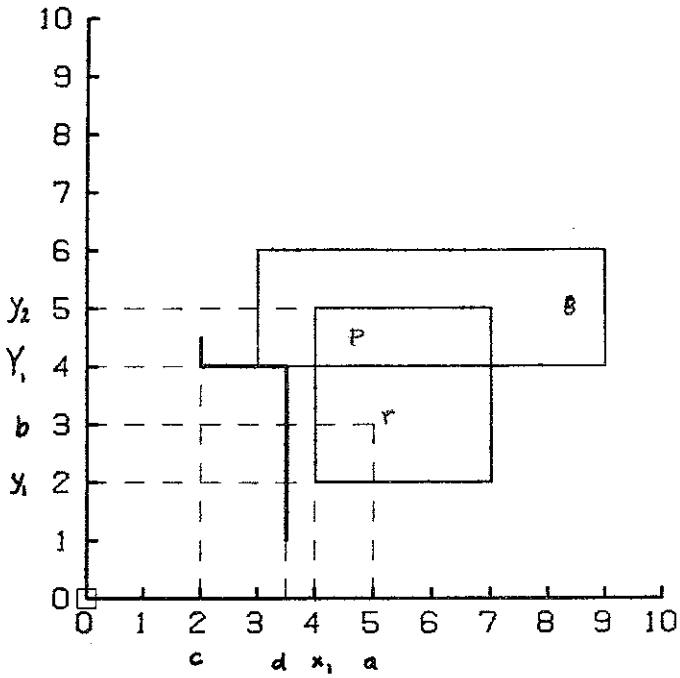

Figure 2. Intersection of a subrectangle and a basic rectangle

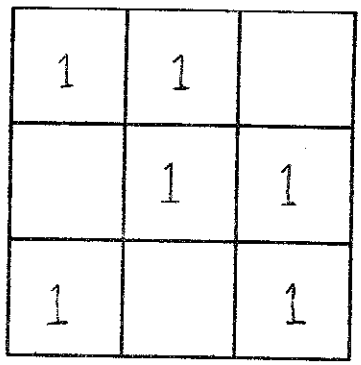

Figure 3. A cyclic chart

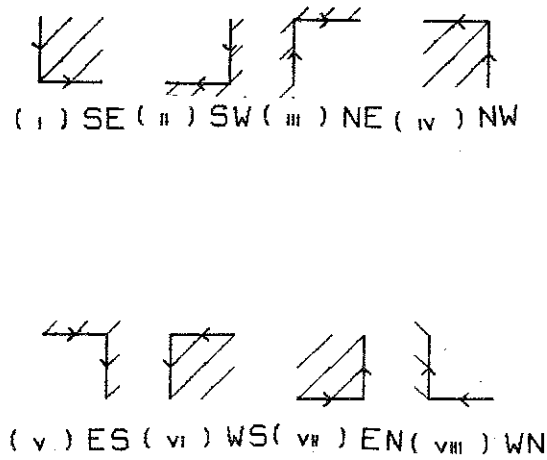

Figure 4. Determining where the figure lies (external contour)

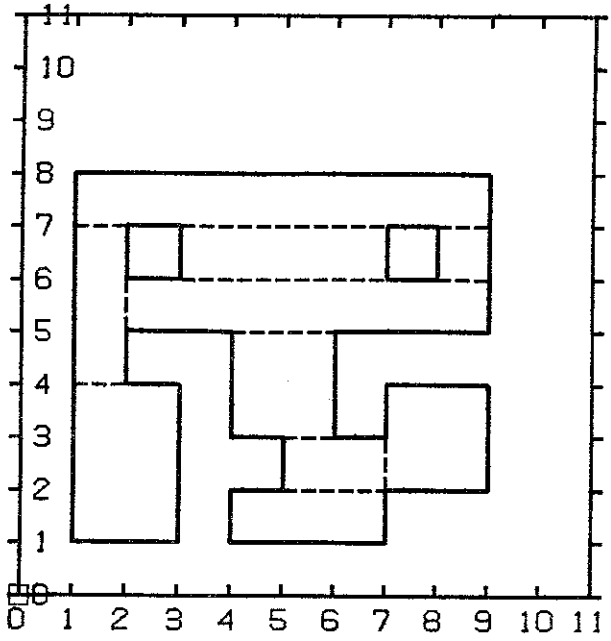

Figure 5. Minimum partition

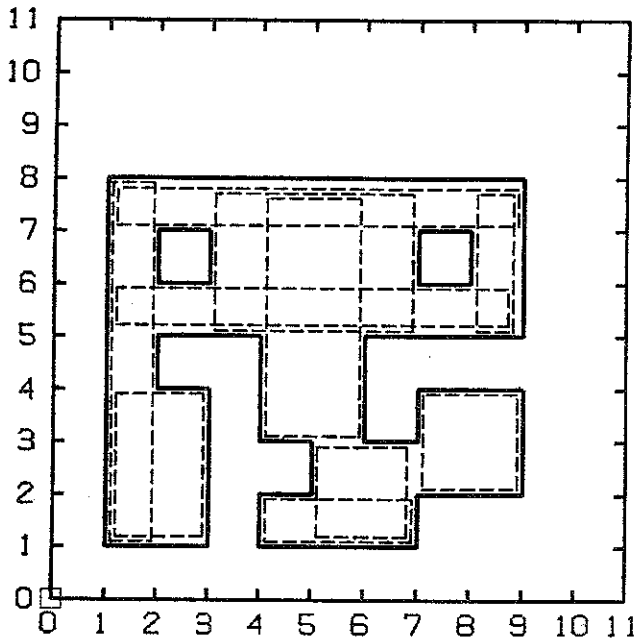

Figure 6. Minimum cover

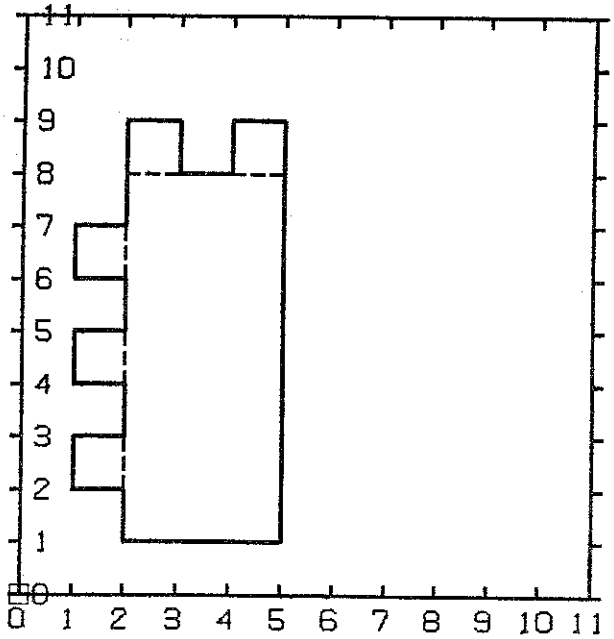

Figure 7. Minimum partition

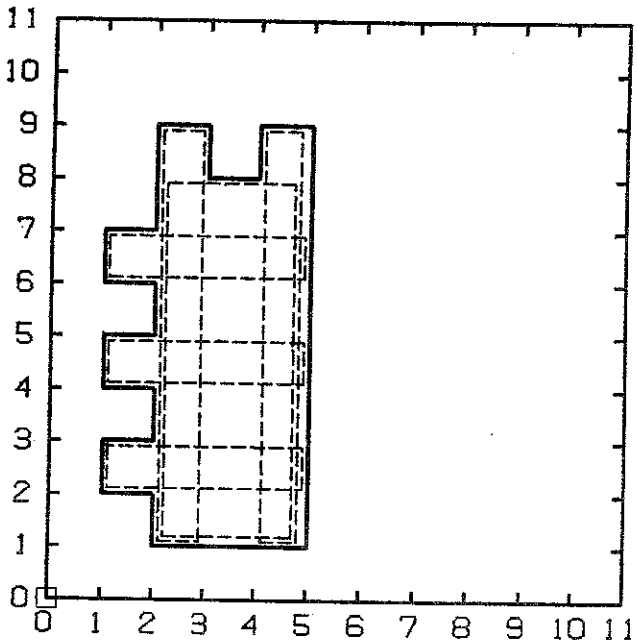

Figure 8. Minimum cover

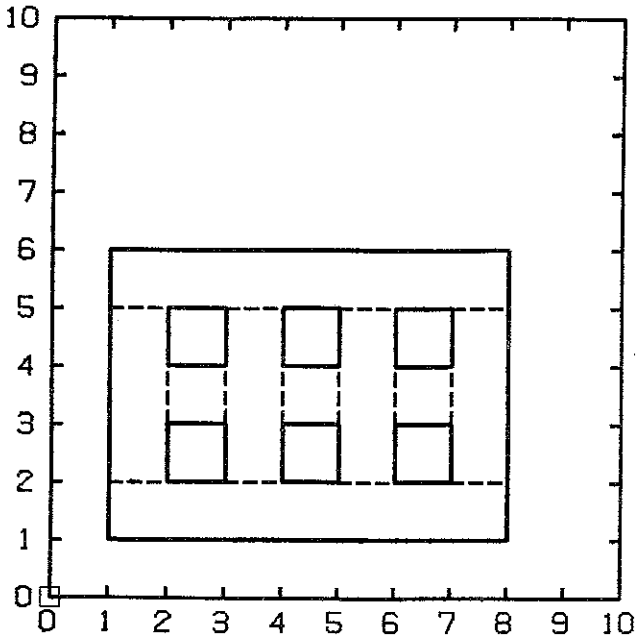

Figure 9. Minimum partition

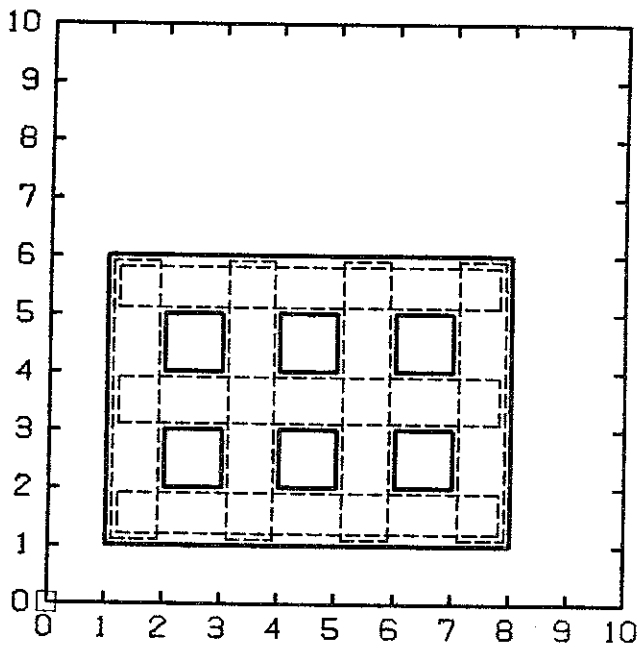

Figure 10. Minimum cover