DECOMPOSING RECTILINEAR REGIONS INTO RECTANGLES

by

Ritu Chadha

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Master of Science
in
Computer Science

APPROVED:

__________________________
Donald C. S. Allison, Chairman

__________________________  ______________________
Lenwood S. Heath               Monte B. Boisen

November, 1987
Blacksburg, Virginia
DECOMPOSING RECTILINEAR REGIONS INTO RECTANGLES

by

Ritu Chadha

Donald C. S. Allison, Chairman

Computer Science

(ABSTRACT)

This thesis discusses the problem of decomposing rectilinear regions, with or without holes, into a minimum number of rectangles. There are two different types of decomposition 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 $O(n^{3/2})$, where $n$ is the number of vertices of the figure, whereas the covering problem is exponential in its time complexity.
I would like to express my thanks to Dr. Donald Allison for his support and guidance during the research conducted for this thesis. His suggestions and advice were an invaluable source of help to me. I would also like to thank Dr. Lenwood Heath and Dr. M. Boisen for serving as members of my committee. Thanks are also due to Adil Godrej for helping me to use NAGPLOT, a software package developed by him which I used to generate the figures in this thesis.
Table of Contents

1.0 Introduction ......................................................... 1
  1.1 Problem description ............................................... 1
  1.2 Literature Review ................................................. 2
  1.3 Document description ............................................. 5

2.0 Theoretical discussion ............................................. 7
  2.1 Minimum rectangle partition ..................................... 7
    2.1.1 Definitions and terminology ................................ 7
    2.1.2 Notation ..................................................... 9
    2.1.3 Lemma I. ................................................... 9
    2.1.4 Corollary I. ............................................... 12
    2.1.5 Theorem I. ............................................... 13
    2.1.6 Theorem II. .............................................. 14
    2.1.7 Corollary II. ............................................. 15
    2.1.8 Theorem III. ............................................. 24
  2.2 Minimum rectangle cover ......................................... 26
    2.2.1 Definitions and terminology ................................ 26
2.2.2 Lemma .......................................... 27
2.2.3 Theorem I ...................................... 27
2.2.4 Theorem II ..................................... 28
2.2.5 Some properties of basic rectangles and subdivisions. ............ 30
  2.2.5.1 Property 1 .................................... 30
  2.2.5.2 Property 2 .................................... 30
  2.2.5.3 Property 3 .................................... 31
  2.2.5.4 Property 4 .................................... 36
2.2.6 Theorem III ..................................... 38
2.2.7 Constructing a minimum rectangle cover .......................... 39
  2.2.7.1 Choosing basic rectangles for the minimum cover. ............ 39
  2.2.7.2 Removing dominated rows .................................. 40
  2.2.7.3 Removing dominating columns ................................ 41
  2.2.7.4 Cyclic charts ...................................... 41
  2.2.7.5 Points to note ..................................... 42

3.0 Data structures and implementation .................................. 44
3.1 Definitions and terminology ......................................... 44
3.2 Data structures ........................................... 44
  3.2.1 Lists of vertices ..................................... 45
  3.2.2 Lists of lines ........................................ 45
  3.2.3 Lists of rectangles ..................................... 46
  3.2.4 Description of the different types of lists used ................ 46
3.3 Implementation ............................................. 48
  3.3.1 Input and output ...................................... 48
  3.3.2 Preliminary data sorting .................................. 49
    3.3.2.1 Data input and sorting ............................. 49
    3.3.2.2 Construction of the edges of the figure. ............. 53

Table of Contents  
v
3.3.2.3 Derivation of the contour of the figure ............................. 54
3.3.2.4 Finding concave vertices ........................................... 55
3.3.2.5 Sorting the vertices by column ...................................... 61

3.3.3 Finding a minimum rectangle partition .................................. 63
  3.3.3.1 Finding internal lines joining pairs of collinear edges .......... 63
  3.3.3.2 Finding a maximum set of partitioning lines ............... 65
  3.3.3.3 Completion of the partition ....................................... 66

3.3.4 Finding a minimum rectangle cover .................................... 68
  3.3.4.1 Separating left and right edges ................................. 68
  3.3.4.2 Creating the subdivision of the figure ....................... 69
  3.3.4.3 Constructing the basic rectangles of the figure .......... 71
  3.3.4.4 Constructing the subrectangles of the figure ............. 75
  3.3.4.5 Building and processing the cover chart .................... 76

4.0 Presentation of results ..................................................... 79

5.0 Conclusions ........................................................................ 111

References .............................................................................. 114

Vita ....................................................................................... 116
# List of Illustrations

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Overlapping edge of type (i)</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>Overlapping edge of type (ii)</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>A minimum partition of a figure without collinear edges.</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>a and b belong to external contour.</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>a and b belong to the contour of the same hole.</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>a belongs to external contour and b belongs to contour of a hole</td>
<td>21</td>
</tr>
<tr>
<td>7</td>
<td>a and b belong to contours of different holes</td>
<td>22</td>
</tr>
<tr>
<td>8</td>
<td>Transforming a figure with collinear edges into one without any.</td>
<td>23</td>
</tr>
<tr>
<td>9</td>
<td>Example illustrating two intersecting concave lines</td>
<td>25</td>
</tr>
<tr>
<td>10</td>
<td>Subdivision of a figure</td>
<td>29</td>
</tr>
<tr>
<td>11</td>
<td>$Y_2 \leq y_1$</td>
<td>32</td>
</tr>
<tr>
<td>12</td>
<td>$y_2 \leq Y_1$</td>
<td>33</td>
</tr>
<tr>
<td>13</td>
<td>$X_2 \leq x_1$</td>
<td>34</td>
</tr>
<tr>
<td>14</td>
<td>$x_2 \leq X_1$</td>
<td>35</td>
</tr>
<tr>
<td>15</td>
<td>Intersection of a subrectangle and a basic rectangle.</td>
<td>37</td>
</tr>
<tr>
<td>16</td>
<td>A cyclic chart</td>
<td>43</td>
</tr>
<tr>
<td>17</td>
<td>Example input figure.</td>
<td>50</td>
</tr>
<tr>
<td>18</td>
<td>Ordering of input data.</td>
<td>52</td>
</tr>
<tr>
<td>19</td>
<td>Definition of a concave vertex</td>
<td>56</td>
</tr>
<tr>
<td>20</td>
<td>Convex and concave vertices</td>
<td>58</td>
</tr>
<tr>
<td>21</td>
<td>Directions of travel around contour.</td>
<td>59</td>
</tr>
</tbody>
</table>
Figure 50. Minimum partition for Data Set 8. ................................. 102
Figure 51. Minimum cover for Data Set 8. ................................. 103
Figure 52. Data Set 9. ................................................................. 104
Figure 53. Minimum partition for Data Set 9. ................................. 105
Figure 54. Minimum cover for Data Set 9. ................................. 106
Figure 55. Data Set 10. ................................................................. 107
Figure 56. Minimum partition for Data Set 10. ................................. 108
Figure 57. Minimum cover for Data Set 10. ................................. 109
1.0 Introduction

1.1 Problem description

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. A minimum cover is one of minimum cardinality among all covers; a minimum partition is one of minimum cardinality among all partitions. Our goal in this work is to establish a method for finding a minimum rectangle cover and a minimum rectangle partition for a rectilinear figure.
1.2 Literature Review

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, rectangles, etc. 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.

The decomposition of polygons into simpler components is broadly divided into two categories: decomposition into overlapping components or covering, and decomposition into non-overlapping components or partitioning. According to Keil [10], 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 [19].

A number of polygon covering problems have been studied in the past. In 1979, Masek [17] proved that the problem of finding a minimum rectangle cover for a rectilinear polygon is NP-complete. Chaiken et. al. [2] therefore restricted the above problem to that of finding a minimum rectangle cover for a horizontally and vertically convex figure (a figure is said to be convex if any line having both extremities lying in the the figure lies entirely inside the figure).
They solved this problem in polynomial time. O'Rourke and Supowit [19] studied the complexity of the problem of finding minimum decompositions of a polygon into convex, star-shaped, or spiral subsets, which may overlap. They showed that the problem is NP-hard by demonstrating that it is polynomially transformable to a known NP-complete problem. O'Rourke [18] described an algorithm which solves the problem of deriving a cover consisting of convex polygons for any general polygon. The method he used is derived from the method generally used for minimizing Boolean switching functions and runs in exponential time. According to O'Rourke, this method may not be optimal.

Franzblau and Kleitman [4] also investigated finding minimum rectangle covers. The class of polygons whose decomposition they studied were vertically convex rectilinear figures. The method they developed was based on projecting each horizontal edge onto a straight line and using the horizontal intervals thus generated to form bases of maximal rectangles of the cover (a maximal rectangle of a polygon is a rectangle contained in the polygon which is not properly contained in any other rectangle contained in the polygon.). The problem of finding a minimum rectangle cover for the polygon is thus reduced to finding a minimum set of horizontal intervals whose union is equal to the union of all horizontal intervals of the figure. The order of their algorithm is quadratic. Keil [10] worked on the problem of finding minimum orthogonal and convex star-shaped decompositions, possibly overlapping, of horizontally convex orthogonal figures. He demonstrated that this can be done in $O(n^2)$ time ($n$ is the number of vertices of the figure).

Partitioning problems which have been studied more often deal with the decomposition of general polygons into a minimum number of simpler components. Lipski et. al. [15] 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 $O(m^3)$. Chazelle and Dobkin [3] studied the problem of partitioning a polygon into a minimum number of convex parts. Although it was initially conjectured that this problem was NP-complete, Chazelle and Dobkin show in their paper that it can be solved in polynomial time. However, they believe that the algorithm given by them
is not optimal and could be further improved. Asano and Asano [1] studied the decomposition of polygonal regions into a minimum number of non-overlapping trapezoids with two horizontal sides. They showed that this problem is polynomially transformable into the maximum independent set problem for a certain class of graphs, which is NP-complete, thus proving that the problem of minimum partitions of polygonal regions into trapezoids is NP-complete. They then developed an $O(n \log n)$ approximation algorithm for solving this problem and an $O(n^3)$ exact algorithm for polygonal regions without holes. Keil [9] used dynamic programming to develop 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. Using the same technique, he also produced polynomial time algorithms for the decomposition of a polygon into the simpler components described above, while minimizing the lengths of the internal edges which formed the decomposition. The method of decomposition presented by Keil did not introduce Steiner points (Steiner points are additional vertices which may be introduced when decomposing a polygon into simpler parts). For polygons with holes, however, many of the decomposition problems become NP-hard.

Kahn, Klawe, and Kleitman [8] proved that rectilinear regions can be partitioned into convex quadrilaterals. The motivation for this problem comes from Chvatal's watchman theorem, which shows that if the walls of an art gallery form an $n$-sided polygon, then at most $\lfloor n/3 \rfloor$ watchmen are needed to guard it (assuming that a watchman can see in all directions) and this number is the lowest possible. Kahn, Klawe, and Kleitman show in this paper that if the walls of the gallery form a rectilinear polygon, then at most $\lfloor n/4 \rfloor$ watchmen are required and this result is again the lowest possible. The proof developed depends on showing that any rectilinear polygon can be partitioned into convex quadrilaterals. Lubiw [16] generalized this result to a larger class of polygons. The purpose of Lubiw's work was to characterize some classes of polygons which can be decomposed into convex quadrilaterals.

Other work on partitioning polygons consists of minimizing the sum of the lengths of the lines forming the partition of the polygon. Lingas et. al. [13] showed that a minimum edge length partition of a rectilinear polygon into rectangles can be derived in $O(n^4)$ time if the
polygon has no holes; if the polygon has holes, the problem becomes NP-hard. Keil [9] established a polynomial time algorithm for partitioning a general polygon into convex, star-shaped, spiral, and monotone subsets with minimum length partition lines. Gonzalez and Zheng [5] provided a polynomial time approximation algorithm for deriving a minimum edge length partition of a rectilinear figure with interior points into rectangles; an exact solution of this problem is intractable. Levcopoulos [12] developed a fast approximation algorithm for finding minimum length partitions of rectilinear polygons into rectangles. His algorithm runs in time $O(n \log n)$. Johnson [7] gave a brief overview of the advances made in these areas in his quarterly column in the Journal of Algorithms.

This thesis 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. [14,15] 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.

1.3 Document description

This thesis is organized as follows. Section 2 gives a complete theoretical proof of the methods used for deriving the minimum rectangle cover and partition of a given rectilinear figure. Section 3 describes the data structures used in the solution of the problem and the algorithms for performing the decomposition of the rectilinear figure into a minimum rectangle cover and partition. This section also contains a complete time-complexity analysis of the given algorithms. Section 4 demonstrates the results obtained from programming the algo-
rithms and running them for several different data sets. The same data sets are used for both the algorithms, i.e. the algorithm for partitioning the figure and the algorithm for covering the figure. Finally, the conclusions of this study are presented in Section 5.
2.0 Theoretical discussion

This chapter discusses the methods used to derive a minimum rectangle partition and a minimum rectangle cover for a given rectilinear figure. In section 2.1, the number of rectangles in a minimum rectangle partition is derived and a method is given for finding a minimum rectangle partition for a given rectilinear figure. In section 2.2, we describe a method for deriving a minimum rectangle cover for a rectilinear figure. The main thrust of this method is the use of Boolean charts for choosing rectangles to be part of the minimum cover for the figure.

2.1 Minimum rectangle partition

2.1.1 Definitions and terminology

A rectilinear figure is a polygon in which each edge is either parallel or perpendicular to a given direction. We assume that the polygon is not self-intersecting and does not have
overlapping boundaries, i.e. we must be able to travel around the external contour and around the contours of the holes of the figure in a counterclockwise direction and during the travel visit each vertex of the figure exactly once. There are two possible exceptions to this rule, mentioned after the following definitions.

A vertex of F is a point of intersection of two edges of the figure F.

A hole of F is a region which is not part of the figure F and is completely surrounded by F.

The contour of a hole of F is the sequence of vertices of the hole, ordered so that the first vertex in this sequence is the leftmost vertex of the bottommost row of vertices of the hole, and the remaining vertices are in the order in which they would be visited while traveling around the figure in a counterclockwise direction.

The external contour of F is the sequence of all vertices of F which are not vertices of any hole of F, arranged in the same order as that described for the contour of a hole.

The contour of F is the union of the contours of all the holes of F and of the external contour of F.

A decomposition of a figure into non-overlapping rectangles is called a rectangle partition of the figure.

A rectangle partition of a figure of minimum cardinality is called a minimum rectangle partition.

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.
We allow the following two types of overlapping boundaries in a figure:

(i) The figure can be obtained by joining a concave vertex of the external contour to a concave vertex of a hole by a concave line; the resulting figure has one hole less than the original figure.

(ii) The figure can be obtained by joining a concave vertex of a hole to a concave vertex of another hole by a concave line; the resulting figure has one hole less than the original figure.

The ordering of the vertices in the external contour for case (i) and for the resulting hole in case (ii) are shown in Figure 1 on page 10 and Figure 2 on page 11 respectively.

Unless otherwise stated, any rectangle contained in the figure $F$ is assumed to have each of its sides parallel to at least one edge of $F$.

2.1.2 Notation

Let $F$ be a rectilinear polygon. We use the following notation:

- $n =$ number of vertices of $F$
- $h =$ number of holes of $F$
- $x =$ number of convex vertices of $F$
- $v =$ number of concave vertices of $F$
- $r =$ number of created vertices of $F$.

2.1.3 Lemma I.

$$x - v = 4 - 4h.$$ 

Proof:
Figure 1. Overlapping edge of type (i)
Figure 2. Overlapping edge of type (ii)
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.

Let

\[ x_e = \text{number of convex vertices of external contour} \]
\[ x_i = \text{number of convex vertices of hole } i \]
\[ v_e = \text{number of concave vertices of external contour} \]
\[ v_i = \text{number of concave vertices of hole } i. \]

Then we have the following relations:

\[ x_e = v_e + 4 \]
\[ x_i = v_i - 4 \]

\[ \ldots \]
\[ \ldots \]
\[ \ldots \]
\[ x_h = v_h - 4. \]

Summing these, we get
\[ x = v + 4 - 4h, \]

i.e.
\[ x - v = 4 - 4h \] (1)

2.1.4 Corollary I.

\[ x = n/2 - 2h + 2 \] (2)
\[ v = n/2 + 2h - 2 \] (3)

Proof:

Using (1) we know that
\[ x - v = 4 - 4h, \]
and obviously
\[ x + v = n. \]
Solving for \( x \) and \( v \) simultaneously, we get
\[ x = \frac{n}{2} - 2h + 2 \]
\[ v = \frac{n}{2} + 2h - 2. \]

Define a function \( M \) of two variables \( n \) and \( h \) as follows:
\[ M(n, h) = \frac{n}{2} + h - 1, \]
where
\[ n = \text{number of vertices of the figure} \]
\[ h = \text{number of holes of the figure}. \]

\section*{2.1.5 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
\[ P \geq M(n, h) \quad (4). \]

\textbf{Proof :} Let \( F \) be partitioned into \( P \) rectangles, and let \( r \) denote the number of created vertices in \( F \). Since \( F \) has no collinear edges, there is no partition line with extremities which are both concave vertices; also every concave vertex must be the extremity of at least one partition line. Thus every partition line which has a concave vertex as one endpoint must have a created vertex as the other endpoint; hence we have
\[ r \geq v \quad (5). \]

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 is at least equal to
\[ n + 2r. \]
Since each of the \( P \) rectangles of the partition has 4 vertices, we have
4P ≥ n + 2r
≥ n + 2v by (5)
= n + 2(n/2 + 2h - 2) by (3)
= 2n + 4h - 4.
Therefore
P ≥ n/2 + h - 1 = M(n, h). ■

2.1.6 Theorem II.

Let F be a figure without collinear edges. Then there exists a partition of F into P rectangles with

P = M(n, h).

Proof: The proof is by construction. For every concave vertex, draw a horizontal partition line (internal to the figure) starting at the vertex and ending as soon as a vertical edge of the figure is encountered (see for example Figure 3 on page 16). This rids the figure of all the concave vertices and produces a partition of F into P rectangles. Now, the vertex of any rectangle of this partition is either

a convex vertex of F
a concave vertex of F
or a created vertex of F.

Any convex vertex of F is the vertex of exactly one rectangle of the partition; any concave vertex of F is the vertex of exactly one rectangle of the partition (by our construction); and any created vertex of F is the vertex of exactly two adjacent rectangles of the partition (by our construction).

Hence the total number of vertices in this partition is

x + v + 2r
= n + 2r (6) (since x + v = n).
But the number of created vertices is in one-to-one correspondence with the number of concave vertices, since by our construction we created one ‘created vertex’ for each concave vertex of F.

Hence

\[ r = v. \]

Therefore (6) can be rewritten as

\[ n + 2v \]

\[ = n + (n + 4h - 4) \quad \text{by (3)} \]

\[ = 2n + 4h - 4. \]

This is the number of vertices of a set of \( P \) rectangles; since each rectangle has four vertices, we have

\[ P = (2n + 4h - 4)/4 \]

\[ = n/2 + h - 1 \]

\[ = M(n, h) \]

i.e. we have found a partition of \( F \) into \( M(n, h) \) rectangles.

2.1.7 Corollary II.

The minimum number of rectangles into which a rectilinear polygon without collinear edges can be partitioned is

\[ M(n, h) \quad (7). \]

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.

Consider any concave line joining concave vertices \( a \) and \( b \) in \( F \). The following four cases arise:

Case (i): Both vertices \( a \) and \( b \) belong to the external contour of \( F \).
Figure 3. A minimum partition of a figure without collinear edges.
Case (ii): Both vertices $a$ and $b$ belong to the contour of the same hole in $F$.

Case (iii): $a$ belongs to the external contour of $F$ and $b$ belongs to the contour of a hole.

Case (iv): $a$ and $b$ belong to the contours of different holes in $F$.

In the above cases, drawing a partition line from $a$ to $b$ has the following effect:

**Case (i):** The figure $F$ is partitioned into two smaller figures, $F_1$ and $F_2$, as demonstrated in Figure 4 on page 18.

In this case, if $F_1$ has $n_1$ vertices, then $F_2$ has $n-n_1$ vertices (note that if point $a$ is a vertex for $F_1$, then it is not a vertex for $F_2$, and conversely; similarly for point $b$). The total number of holes in the figure is unchanged; thus if $F_1$ has $h_1$ holes, then $F_2$ has $h-h_1$ holes.

Hence the value of the function $M$ (defined earlier) for figures $F_1$ and $F_2$ is:

$$M(n_1,h_1) = \frac{n_1}{2} + h_1 - 1$$

for $F_1$,

$$M(n-n_1,h-h_1) = \frac{n}{2} - \frac{n_1}{2} + h - h_1 - 1$$

for $F_2$.

The sum of these two values is

$$\frac{n}{2} + h - 2$$

$$= M(n,h) - 1.$$

**Case (ii):** The figure is partitioned into two smaller figures, $F_1$ and $F_2$, as demonstrated in Figure 5 on page 19.

In this case, if $F_1$ has $n_1$ vertices, then $F_2$ has $n-n_1$ vertices (note that if point $a$ is a vertex for $F_1$, then it is not a vertex for $F_2$, and conversely; similarly for point $b$). The total number of holes in the figure is unchanged; thus if $F_1$ has $h_1$ holes, then $F_2$ has $h-h_1$ holes.

Hence the value of the function $M$ (defined earlier) for figures $F_1$ and $F_2$ is:

$$M(n_1,h_1) = \frac{n_1}{2} + h_1 - 1$$

for $F_1$,

$$M(n-n_1,h-h_1) = \frac{n}{2} - \frac{n_1}{2} + h - h_1 - 1$$

for $F_2$.

The sum of these two values is

$$\frac{n}{2} + h - 2$$

$$= M(n,h) - 1.$$
Figure 4. a and b belong to external contour.
Figure 5. a and b belong to the contour of the same hole.
**Case (iii):** The figure F is transformed into a new figure F' which has one hole less than F (see Figure 6 on page 21). Note that in this case F' has a pair of overlapping boundaries of type (i).

The number of vertices of the new figure F' is unchanged. Hence the value of the function M for figure F' is

\[
M(n,h-1) = \frac{n}{2} + (h-1) - 1 = \frac{n}{2} + h - 2 = M(n,h) - 1.
\]

**Case (iv):** As in case (iii), the figure F is transformed into a new figure F' which has one hole less than F (see Figure 7 on page 22). Note that in this case F' has a pair of overlapping boundaries of type (ii).

The number of vertices of the new figure F' is unchanged. Hence the value of the function M for figure F' is

\[
M(n,h-1) = \frac{n}{2} + (h-1) - 1 = \frac{n}{2} + h - 2 = M(n,h) - 1.
\]

Now, to transform a figure F with collinear edges into one without collinear edges, we need to draw enough concave lines to get rid of collinear edges.

E.g. in Figure 8 on page 23, the figure in (i) with two collinear edges is transformed in (ii) into a figure without collinear edges.

Hence assume that a figure F with collinear edges has been transformed into a figure without collinear edges by drawing a number of concave lines. Let n, h be the number of vertices and holes respectively of the figure F. From the above analysis of cases (i) through (iv), we see that the sum of the values of the function M for the figure(s) obtained after drawing a concave line is 1 less than the sum of the values of the function M for the previous figure(s). Thus if we need to draw a total of 'C' concave lines to transform F into a figure without collinear edges, then the sum of the values of the function M for the resulting figure(s) will be

\[
M(n,h) - C \quad (8)
\]
Figure 6.  a belongs to external contour and b belongs to contour of a hole
Figure 7. a and b belong to contours of different holes
Figure 8. Transforming a figure with collinear edges into one without any.
(since the value of M decreases by 1 with the addition of each concave line, therefore after adding C concave lines, the value of M decreases by C).

Now, each of the resulting figures obtained after drawing C concave lines is a figure without collinear edges; hence if the number of vertices and holes of any such figure is n, and h (respectively), then the minimum number of rectangles into which it can be partitioned is

\[ \frac{n}{2} + h - 1 \quad \text{(by Corollary II)} \]

\[ = M(n, h). \]

And the minimum number of rectangles into which the figure F can be partitioned is the sum of the minimum numbers of rectangles into which each of the new figures (resulting from drawing concave lines) can be partitioned; from (8) this number is

\[ M(n, h) - C \]

\[ = \frac{n}{2} + h - 1 - C \quad (9). \]

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 9 on page 25 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

2.1.8 Theorem III.

The minimum number of rectangles into which a figure F can be partitioned is

\[ \frac{n}{2} + h - 1 - C \]
Figure 9. Example illustrating two intersecting concave lines
where

\[ n = \text{number of vertices of } F \]
\[ h = \text{number of holes of } F \]
\[ C = \text{maximum cardinality of a set } S \text{ of concave lines, no two of which intersect each other.} \]

**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.

### 2.2 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 \( F \).

#### 2.2.1 Definitions and terminology

The symbol \( \subseteq \) means 'is a subset of'; the symbol \( \subset \) means 'is a proper subset of'.

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 \(R = (x_1,y_1 ; x_2,y_2)\).

A horizontal line joining points \((x_1,y)\) and \((x_2,y)\) is denoted by \((x_1, x_2 ; y)\); similarly, a vertical line joining points \((x,y_1)\) and \((x,y_2)\) is denoted by \((x ; y_1, y_2)\).

### 2.2.2 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 \(\pm d\) 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 \(P\). ◾

### 2.2.3 Theorem I.

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

**Proof**: Let \(a = (x, y)\) be any point in \(F\), and let \(R = (x, y; x, y)\). From the lemma, we can find a basic rectangle \(B\) containing \(R\); thus the point \(a\) is contained in basic rectangle \(B\).

Since \(a\) was an arbitrary point in the figure \(F\), we have shown that every point in \(F\) is contained in at least one basic rectangle; thus the set of all basic rectangles forms a rectangle cover for \(F\). ◾
2.2.4 Theorem II.

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

Proof: Let S be any minimum cover of F, and let $S = \{P_1, P_2, ..., P_n\}$, where each $P_i$ is a rectangle. Consider any rectangle $P_i$ which belongs to this cover. By the previous lemma, there exists a basic rectangle $B_i$ such that $P_i \subseteq B_i$.

Define the set $S'$ by

$$S' = \{B_1, B_2, ..., B_n\},$$

where each $B_i$ is defined as above for $1 \leq i \leq n$. Then the cardinality of the set $S'$ is $n$, which is equal to the cardinality of the set $S$, and $S'$ contains only basic rectangles. Also, since for each $i$ with $1 \leq i \leq n$ we have

$$P_i \subseteq B_i,$$

we get

$$\bigcup_{i=1}^{n} P_i \subseteq \bigcup_{i=1}^{n} B_i,$$

i.e. the area covered by the set $S$ is also covered by the set $S'$.

Hence $S'$ is a cover for F consisting only of basic rectangles and containing the same number of rectangles as $S$; therefore $S'$ is a minimum cover for F. •

Now, the union of all the basic rectangles of a figure F is the figure F itself (by Theorem I). We eliminate as many rectangles as possible from this union in order to get a rectangle cover for F consisting of a minimum number of basic rectangles (this is possible by Theorem II). 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 (see Figure 10 on page 29).

We call the resulting partition the subdivision of F, and the rectangles of this partition are called subrectangles.
Figure 10. Subdivision of a figure
2.2.5 Some properties of basic rectangles and subdivisions.

2.2.5.1 Property 1.

(i) The left and right edges of a basic rectangle each have a segment in common with some vertical boundary of the figure.

(ii) 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.

2.2.5.2 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: Suppose R is not a basic rectangle. Then by definition there exists a rectangle R' contained in F such that R ⊊ R'.

Since R' ⊊ R, therefore at least one of its sides does not coincide with one of the edges of R. But this edge of R has a segment in common with some boundary of F; hence some segment of a boundary of F is strictly contained inside R'. Thus R' does not lie entirely within the figure F, which is a contradiction.

Hence R must be a basic rectangle of the figure F.
2.2.5.3 **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) Suppose \( Y_2 \leq y_1 \). Then the entire rectangle \( S \) lies below the rectangle \( R \), and the only points which \( S \) and \( R \) could possibly have in common are points lying on the lower boundary of \( R \) or on the upper boundary of \( S \), which contradicts the fact that the rectangles \( S \) and \( R \) intersect non-trivially (see Figure 11 on page 32).

(ii) Suppose \( y_2 \leq Y_1 \). Then the entire rectangle \( R \) lies below the rectangle \( S \), and the only points which \( S \) and \( R \) could possibly have in common are points lying on the lower boundary of \( S \) or on the upper boundary of \( R \), which contradicts the fact that the rectangles \( S \) and \( R \) intersect non-trivially. (see Figure 12 on page 33).

(iii) Suppose \( X_2 \leq x_1 \). Then the entire rectangle \( S \) lies to the left of rectangle \( R \), and the only points which \( S \) and \( R \) could possibly have in common are points lying on the right boundary of \( S \) or on the left boundary of \( R \), which contradicts the fact that the rectangles \( S \) and \( R \) intersect non-trivially. (see Figure 13 on page 34).

(iv) Suppose \( x_2 \leq X_1 \). Then the entire rectangle \( R \) lies to the left of rectangle \( S \), and the only points which \( S \) and \( R \) could possibly have in common are points lying on the right boundary of \( R \) or on the left boundary of \( S \), which contradicts the fact that the rectangles \( S \) and \( R \) intersect non-trivially. (see Figure 14 on page 35). ■
Theoretical discussion

Figure 11. \( Y \leq y \),
Theoretical discussion

Figure 12. $y_1 \leq Y_1$.
Figure 13. \( X_r \leq x_s \)
Figure 14. \( x_s \leq X_s \)
2.2.5.4 Property 4.

If \( P \) is a rectangle of the subdivision of \( F \) and \( B \) is a basic rectangle, then either
\[ B \cap P = \emptyset \quad \text{or} \quad P \subseteq B. \]

Proof: Note that \( B \cap P = \emptyset \) is equivalent to saying that \( B \) and \( P \) do not intersect non-trivially; i.e. they have no point in common which does not lie on the boundary of either \( B \) or \( P \).

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 otherwise 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 15 on page 37).
Figure 15. Intersection of a subrectangle and a basic rectangle.
Then the vertex \((d, Y_i)\) 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 \((d, Y_i)\)). Hence by our method for obtaining a subdivision of \(F\), we must draw a horizontal line internal to the figure, joining the concave vertex \((d, Y_i)\) 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 \(P \subseteq B\).

\[\blacksquare\]

2.2.6  Theorem III.

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

\[p_1 \in B \implies p_2 \in B\]

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 therefore we have proved

\[p_1 \in B \implies p_2 \in B\]

for any arbitrary basic rectangle \(B\).

By symmetry, it is obvious that the converse also holds; thus
p₁ ∈ B ⇐⇒ p₂ ∈ B for any arbitrary basic rectangle B of F, where p₁ and p₂ are two points belonging to the same subrectangle of the subdivision of F. ■

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 [11]). The method is described in some detail in the following pages.

2.2.7 Constructing a minimum rectangle 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.

2.2.7.1 Choosing basic rectangles for the minimum cover.

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 we first scan all the columns of the chart for columns containing only one ‘1’. When such a column is found, the row i in which the ‘1’ is located is removed from the chart, along with every column in which there is a ‘1’ in the ith row. The reason for eliminating all such columns is that the basic rectangle B corresponding to row i covers all the subrectangles corresponding to these columns; hence since B is being selected for the minimum rectangle cover of F, these subrectangles are covered by B and need no longer be considered. The basic rectangle B corresponding to row i is added to a list L (L is initially empty) of basic rectangles being selected to form a minimum cover for F.

2.2.7.2 Removing dominated rows

After all columns containing a single ‘1’ are located and eliminated according to the procedure described above, we scan the chart A for dominated rows. A row i is said to be dominated by a row i’ if and only if

\[ A[i,j] = 1 \Rightarrow A[i',j] = 1 \quad \text{for all columns } j \text{ in the chart } A. \]

A dominated row i which is dominated by row i’ can be removed from the chart for the following reason: since the row i’ contains 1’s in all the columns in which row i contains 1’s, it is clear that every subrectangle covered by the basic rectangle corresponding to row i is also covered by the basic rectangle corresponding to row i’; hence the basic rectangle corresponding to row i is redundant and can be removed from the chart.
2.2.7.3 **Removing dominating columns.**

A column $j$ is said to dominate another column $j'$ if and only if

$$A[i,j'] = 1 \Rightarrow A[i,j] = 1$$

for all rows $i$ in the chart $A$.

i.e. the column $j$ has 1's in all the rows in which column $j'$ has 1's. We now remove all dominating columns from $A$. The reason for this is that any basic rectangle which covers the subrectangle corresponding to column $j'$ also covers the subrectangle corresponding to column $j$; hence when we choose a basic rectangle to cover the subrectangle corresponding to column $j'$, we will also have chosen a basic rectangle which covers the subrectangle corresponding to column $j$. Thus for all practical purposes, column $j$ can be ignored and therefore every dominating column is removed from the chart $A$.

The three basic steps for chart reduction have been described above. After all three steps have been carried out in this order, they may need to be repeated several times in order to reduce the chart completely. The chart is completely reduced when all its rows and columns have been eliminated.

2.2.7.4 **Cyclic charts**

It is possible that the repeated application of these steps does not reduce the chart; in this case, the chart is said to be cyclic. For example, Figure 16 on page 43 gives an instance of a chart which cannot be reduced any further by any of the three previously described methods. In this chart, each column has two 1's (therefore we cannot find any column with only one '1' in it); there is no dominated row, and there is no dominating column. To reduce this chart, we use the following method. Choose a column $j$ of the chart with a minimum number of 1's in it (in the above chart, each chart has two 1's and therefore we may choose any column at random). Suppose this column has $m$ 1's in it. We now obtain $m$ different charts from the cyclic chart as follows: let row $i$, be the first row which has a '1' in column $j$. Remove
row i, from the chart, along with every column in which row i, has a '1', and put the basic rectangle corresponding to row i, in a new list of basic rectangles, called L_i, say. We are now left with a chart A_i which has one row less than the original chart and one or more columns less than the original chart (depending on the number of 1's in row i,).

Going back to the original chart, we proceed like this for all the m rows which have 1's in column j, and obtain charts A_1, A_2, ..., A_m, and lists L_1, L_2, ..., L_m, where each subchart is obtained from the original chart by the method described above. Each of these m subcharts is smaller than the original chart A and can be reduced by all the methods described above (viz. choosing basic rectangles, eliminating dominated rows, and eliminating dominated columns). After each subchart has been completely reduced, we scan the lists L_1 to L_m to find which one has the smallest number of elements. Suppose L_i has the smallest number of elements among L_1, L_2, ..., L_m. We append the list L_i to the list L of basic rectangles; L now contains the minimum number of basic rectangles required to cover every subrectangle of the subdivision of F, and hence is a minimum rectangle cover.

2.2.7.5 Points to note

1. While processing a cyclic chart, the reason for choosing a column with the minimum number 'm' of 1's to start with is that the number of charts which we will have to process independently will be 'm'; hence choosing the smallest possible value for 'm' will minimize the number of charts which will have to be processed independently.

2. Note that when reducing the m smaller charts produced from a cyclic chart, we may obtain smaller cyclic charts during the process of reduction. These charts must then be reduced in the same way as described for cyclic charts. Thus the procedure for reducing charts is of a recursive nature.
Figure 16. A cyclic chart
3.0 Data structures and implementation

3.1 Definitions and terminology

The horizontal neighbor of a vertex v is the other extremity of the horizontal edge through v; similarly, the vertical neighbor of a vertex v is the other extremity of the vertical edge through v.

3.2 Data structures

The major data structures used in this problem are lists of vertices, lists of edges, and lists of rectangles.
3.2.1 Lists of vertices

Each vertex \( v \) is stored in a record which holds the following information:

(i) The \( x,y \) coordinates of the vertex.

(ii) An information field indicating whether \( v \) is the upper or the lower extremity of the vertical edge through \( v \).

(iii) An information field indicating whether \( v \) is the right or the left extremity of the horizontal edge through \( v \).

(iv) An information field indicating whether \( v \) is concave or convex.

(v) An information field indicating whether \( v \) has been visited or not.

(vi) A pointer to the horizontal neighbor of \( v \).

(vi) A pointer to the vertical neighbor of \( v \).

A list of vertices is an array of pointers to a set of vertices.

3.2.2 Lists of lines

Lists of lines are differentiated into two categories: vertical lines and horizontal lines. A vertical line is represented by three numbers: the \( x \)-coordinate of the line, the \( y \)-coordinate of its lower extremity, and the \( y \)-coordinate of its upper extremity. A horizontal line is also represented by three numbers: the \( y \)-coordinate of the line, the \( x \)-coordinate of its left extremity, and the \( x \)-coordinate of its right extremity. Lists of lines are arrays of lines.
3.2.3 Lists of rectangles

A rectangle is represented by four numbers: the x,y coordinates of its lower left corner and the x,y coordinates of its upper right corner. A list of rectangles is represented by an array of rectangles.

3.2.4 Description of the different types of lists used

- **Hvertices**: list of vertices sorted such that for any pair of vertices i, j with i < j, we have either
  
  (i) \( y_i < y_j \)
  
  or (ii) \( y_i = y_j \) and \( x_i < x_j \)
  
  where the \( i^{th} \) vertex is \((x_i, y_i)\) and the \( j^{th} \) vertex is \((x_j, y_j)\).
  
  In other words, the vertices in Hvertices are sorted vertically.

- **Vvertices**: list of vertices sorted such that for any pair of vertices i, j with i < j, we have either
  
  (i) \( x_i < x_j \)
  
  or (ii) \( x_i = x_j \) and \( y_i < y_j \)
  
  where the \( i^{th} \) vertex is \((x_i, y_i)\) and the \( j^{th} \) vertex is \((x_j, y_j)\).
  
  In other words, the vertices in Vvertices are sorted horizontally.

- **EXT**: list of vertices of the external contour of the figure. These vertices are arranged in the order in which they would be visited during a counterclockwise traversal of the figure, starting at the leftmost vertex of the bottommost row of vertices.
• HOLE: array of 'h' lists of vertices (where h is the number of holes in the figure); the
  \(i^{th}\) list contains the vertices of the \(i^{th}\) hole of the figure, in the same order as described for the
  list \text{EXT}.

• \text{HElist}: list of horizontal edges of the figure. Since the figure has \(n\) vertices, there are
  \(n/2\) horizontal edges.

• \text{VElist}: list of vertical edges of the figure. Since the figure has \(n\) vertices, there are \(n/2\)
  vertical edges.

• \text{LElist}: list of left vertical edges of the figure. Since the figure has \(n/2\) vertical edges,
  there are less than \(n/2\) left vertical edges.

• \text{RElist}: list of right vertical edges of the figure. Since the figure has \(n/2\) vertical edges,
  there are less than \(n/2\) right vertical edges.

• \text{HorizCol}: list of horizontal lines which form a partition of \(F\) into a minimum number
  of non-overlapping rectangles.

• \text{VertCol}: list of vertical lines which form a partition of \(F\) into a minimum number of
  non-overlapping rectangles.

• \text{BR}: list of basic rectangles of the figure.

• \text{SUBRECT}: list of subrectangles of the figure.
• HCOMB : list of horizontal edges of the figure and of horizontal lines which partition the figure into subrectangles of the subdivision of F. The list HCOMB is sorted so that for any pair of edges \( \text{HCOMB}[i] = (x_1, x_2; y_1) \) and \( \text{HCOMB}[j] = (X_1, X_2; Y_1) \) with \( i < j \), we have either
(i) \( y_1 < Y_1 \)
or (ii) \( y_1 = Y_1 \) and \( x_1 < X_1 \).

• VCOMB : list of vertical edges of the figure and of vertical lines which partition the figure into subrectangles of the subdivision of F.

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

• LVCOMB : list of left vertical edges of the figure and of vertical lines which partition the figure into subrectangles of the subdivision of F.

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

3.3 Implementation

3.3.1 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.
E.g. for the figure shown in Figure 17 on page 50, the input file could be as shown below:

```
1 1
1 2
2 2
2 1
```

Note that the order in which the vertices are input is immaterial.

The output of the program consists of a minimum rectangle partition of F and a minimum rectangle cover of F.

### 3.3.2 Preliminary data sorting

The algorithm developed here consists of several stages which are described in the following pages.

#### 3.3.2.1 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 \(v_1, v_2, \ldots, v_n\), \(v_i = (x_i, y_i)\), \(1 \leq i \leq n\), 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) \(y_i < y_j\)

or

(ii) \(y_i = y_j\) and \(x_i < x_j\).
Figure 17. Example input figure.
The resulting list contains the vertices of the figure ordered from top to bottom, and from left to right in each row of vertices.

e.g. in Figure 18 on page 52, the order of the vertices after sorting them is (1,1), (4,1), (1,2), (2,2), (3,2), (4,2), (2,3), (3,3).

After being sorted, the vertices are in a list named Hvertices.
Figure 18. Ordering of input data.
Time complexity analysis

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 $O(n \log n)$.

3.3.2.2 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. Recall that we defined 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 $H_{\text{vertices}}$. 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 $HE_{\text{list}}$.

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 $VE_{\text{list}}$. $v_i$ 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 ($HE_{\text{list}}$) and a list of vertical edges ($VE_{\text{list}}$); 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.

**Time complexity analysis**

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 \( v_{i+1}, v_{i+2}, \ldots, v_n \) 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 up to \( O(n^2) \), which is the overall time complexity of this step.

### 3.3.2.3 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 \( \text{Hvertices} \). Mark this vertex 'visited' and move right to get the next element of the contour, which is also the second element of the list \( \text{Hvertices} \). 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'.

**Time complexity analysis**

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).

### 3.3.2.4 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. The method used is described below. Recall that a vertex of a rectilinear polygon is said to be concave if and only if the angle at the vertex, measured inside the polygon, is 270 degrees; a vertex which is not concave is called convex. For example, in Figure 19 on page 56, w is a concave vertex and v is a convex vertex.

#### Description for external contour

The method used is to travel along the contour of the figure in a counterclockwise direction and to determine whether a vertex is convex or concave according to the direction of travel at any particular time.

Let \( V = \{v_1, v_2, \ldots, v_m\} \) be the set of vertices of the external contour of the figure \( F \). Let \( v_i = (x_i, y_i) \) for \( 1 \leq i \leq m \). We assume that the vertices are numbered in the order in which they would be visited if the figure were to be traversed in a counterclockwise direction. Also assume that \( v_1 \) is the leftmost vertex in the bottommost row of edges of the figure; i.e.

\[
y_i = \min \{y_i \mid 1 \leq i \leq m\}
\]

and

\[
x_i = \min \{x_i \mid y_i = y_1\}.
\]
Figure 19. Definition of a concave vertex
Every vertex \( v \) has exactly two edges incident on it, viz. one horizontal edge and one vertical edge. To determine whether \( v \) is convex or concave, we need to determine the side of these edges on which the figure \( F \) lies.

E.g. in Figure 20 on page 58, the figure \( F \) lies on the shaded side of the edges incident on vertex \( v \); in (i) \( v \) is a convex vertex, whereas in (ii) \( v \) is a concave vertex.

There are eight possible configurations of the directions in which the two edges incident on a vertex can be traversed while visiting the vertex \( v \). These configurations are South and East, East and South, South and West, West and South, North and East, East and North, North and West, West and North (see Figure 21 on page 59).

Claim: In configurations (i), (iv), (vi), and (vii) of Figure 21 on page 59, \( v \) is a convex vertex, whereas in configurations (ii), (iii), (v) and (viii) \( v \) is a concave vertex.

Proof: Consider (i), (iv), (vi) and (vii) in Figure 21 on page 59. Since the directions South and East, North and West, West and South, East and North correspond to a counterclockwise movement, the figure \( F \) must lie on the side shown shaded in Figure 22 on page 60; this shows that \( v \) is a convex vertex. In configurations (ii), (iii), (v) and (viii), the directions of motion are South and West, North and East, East and South, West and North, which correspond to a clockwise movement. For this to be possible in a counterclockwise traversal of the contour of the figure, the figure \( F \) must lie on the side shown shaded in Figure 22 on page 60, and we see that in this case \( v \) is a concave vertex.

Thus to determine whether a vertex is convex or concave, all we need to do is note the directions in which the edges incident on the vertex are traversed during a counterclockwise traversal of the external contour of the figure and use the results depicted in Figure 22 on page 60 to label the vertex as convex or concave.

Description for holes

The method for determining whether the vertex of a hole is convex or concave is exactly the reverse of that for vertices of the external contour. This can readily be explained as follows: if a figure \( F \) has a hole \( F' \), and if we consider \( F' \) as a figure by itself, then every vertex of the figure \( F' \) which is convex when \( F' \) is considered as a hole will be concave when \( F' \) is
Figure 20. Convex and concave vertices.
Figure 21. Directions of travel around contour.
Figure 22. Determining where the figure lies (external contour)
considered as a figure by itself, and vice-versa (see Figure 23 on page 62). The reason for this is that in (i) of Figure 23 on page 62, the area of the figure lies outside F', whereas in (ii) the area of the figure lies inside F'.

Thus to label the vertices of a hole as convex or concave, we use the procedure outlined for the external contour except that any vertex which would have been labelled as convex in the external contour is now labelled concave and vice-versa.

Thus to determine whether a vertex belonging to the external contour of the figure is concave, we implement the method described; 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.

**Time complexity analysis**

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

### 3.3.2.5 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 v₁, v₂, ..., vₙ; vᵢ = (xᵢ, yᵢ), 1 ≤ i ≤ n, then for any vᵢ = (xᵢ, yᵢ), and vⱼ = (xⱼ, yⱼ), with i < j, the following statements hold: either

(i) xᵢ < xⱼ

or

(ii) xᵢ = xⱼ and yᵢ < yⱼ,
Figure 23. Determining where the figure lies (contour of a hole)
Time complexity analysis

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

### 3.3.3 Finding a minimum rectangle partition

#### 3.3.3.1 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. Recall that the vertices in the list 'Vvertices' are ordered column-wise, starting from the leftmost column and going from bottom to top in each column. 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 Vvertices with the same x 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 horizontal 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 HElist of horizontal edges. A horizontal edge running from a point $(a,c)$ to a point $(b,c)$ with $a < b$ 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

$$y_{i+1} < c < y_{i+2}$$

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

The scan of the horizontal edges is interrupted if a horizontal 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 'VertCol', consisting of vertical lines connecting collinear edges.

*Time complexity analysis*
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 $O(n^2)$.

### 3.3.3.2 Finding a maximum set of partitioning lines

We now have to find a maximum subset of HorizCol $\cup$ 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 $\cup$ 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 $\cup$ 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 [6,15]. 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.

**Time complexity analysis**

From [6], the time required here is $O(n^{5/2})$. 

---

*Data structures and implementation*
3.3.3.3 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 \( v_i \) (see Figure 24 on page 67).

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

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.
Figure 24. A concave vertex at the left of an edge
Time complexity analysis

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 $O(n^2)$.

3.3.4 Finding a minimum rectangle cover

3.3.4.1 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.

*Time complexity analysis*

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

### 3.3.4.2 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 \( \text{HCOMB} \). 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 \( \text{VCOMB} \) will contain the vertical lines of the subdivision. The list \( \text{Vvertices} \) of the vertices of the figure is scanned for concave vertices. As soon as a concave vertex is found, the list \( \text{VCOMB} \) 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 \( \text{HElist} \) 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 \( \text{VCOMB} \). 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 \( \text{HCOMB} \) and \( \text{VCOMB} \) have been created, the horizontal edges of the figure are added to the list \( \text{HCOMB} \), and the vertical edges of the figure are added to the list \( \text{VCOMB} \). Before the vertical edges of the figure are added to the list \( \text{VCOMB} \), we first make a copy of the list \( \text{VCOMB} \) and store it in a list named \( \text{LVCOMB} \), and add the left edges of the figure to \( \text{LVCOMB} \). Thus the list \( \text{HCOMB} \) 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 \( \text{VCOMB} \) 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 \( \text{LVCOMB} \) 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_i, x_2; y_i) and HCOMB[j] = (x_i, x_2; y_j) with i < j, we have either

(i) y_i < y_j

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

The list VCOMB is sorted so that for any pair of edges VCOMB[i] = (x_i; y_i, y_2) and VCOMB[j] = (x_i; y_i, y_2) with i < j, we have either

(i) x_i < x_j

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

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

Time complexity analysis

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).

3.3.4.3 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 vertical 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 (see Figure 25 on page 73 for an example of extending a left edge $L$).

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 $L'$ (see Figure 26 on page 74 for examples of this).

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 $L$.

**Time complexity analysis**

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 $O(n^2)$. 
Figure 25. Extending a left edge
Figure 26. Finding the right edge of a basic rectangle
3.3.4.4 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 L₁ (see Figure 27 on page 77). We proceed in exactly the same manner as before to form a subrectangle with L₁ 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.

**Time complexity analysis**

From [14], 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 $O(n^5)$.

### 3.3.4.5 Building and processing the cover chart

We must now build a two-dimensional cover chart which summarizes the covering relationships between the basic rectangles and the subrectangles of figure F (as explained in Chapter 2). The chart has one row for each basic rectangle and one column for each subrectangle of the figure. Entries are made in the chart as follows: for every subrectangle $R$ of the figure, we scan the list of basic rectangles to see which basic rectangles cover subrectangle $R$; if the basic rectangle corresponding to a particular row covers $R$, then a '1' is entered in that row in the column corresponding to $R$; otherwise a '0' is entered.

The processing of the chart is carried out exactly as described in Chapter 2. After the chart has been completely reduced, the basic rectangles which have been chosen to form a minimum rectangle cover for the figure are the basic rectangles in list $L$.

**Time complexity analysis**

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 [14] 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
Figure 27. Leftover portion of a left edge
\[ T(1) = 1 \]
\[ T(n) \leq cn^2 T(n - 1) \]
\[ \leq (cn^2)^2 T(n - 2) \]
\[ \leq ... \]
\[ \leq (cn^2)^{n-1} T(1) \]
\[ = c^{n-1} n^{2n-2}. \]

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.
4.0 Presentation of results

In the previous section, we examined the algorithms for obtaining minimum rectangle partitions and minimum rectangle covers for rectilinear figures. In this section, we present the results obtained after the given algorithms were programmed and executed for a number of different data sets. The data sets were chosen so that for some data sets (e.g. data sets 3, 5, 7, 9, 10), the number of rectangles in a minimum rectangle partition equal the number of rectangles in a minimum rectangle cover; for the other data sets, the number of rectangles in the minimum rectangle cover is at least one less than the number of rectangles in the minimum rectangle partition. The data sets were kept reasonably small to reduce computation costs. The programming language used was Pascal. In the following pages we present the different data sets used to demonstrate the working of this algorithm, along with the resulting minimum partition and minimum cover of the figures. Recall that a set of input data contains the number 'n' of vertices of the figure at the beginning, followed by the x and y coordinates of the 'n' vertices of the figure.

At the end of the chapter is a table summarizing the results obtained by executing the algorithm for these data sets.
Figure 28. Data Set 1.
Figure 29. Minimum partition for Data Set 1.
Figure 30. Minimum cover for Data Set 1.
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>41</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td></td>
<td>42</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>73</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10 4</td>
<td>74</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12 5</td>
<td>10 5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>35</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 6</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>68</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 7</td>
<td>67</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 9</td>
<td>11 9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 9</td>
<td>12 9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7 10</td>
<td>8 10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12 10</td>
<td>11 10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9 13</td>
<td>10 13</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8 12</td>
<td>9 12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10 12</td>
<td>12 12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 12</td>
<td>7 12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 11</td>
<td>2 11</td>
<td></td>
</tr>
</tbody>
</table>

Figure 31. Data Set 2.
Figure 32. Minimum partition for Data Set 2.
Figure 33. Minimum cover for Data Set 2.
| 22 | 21 |
| 42 | 41 |
| 16 | 12 |
| 13 | 14 |
| 34 | 33 |
| 44 | 24 |
| 52 | 23 |
| 43 | 45 |
| 35 | 26 |
| 36 | 46 |
| 27 | 57 |
| 22 |   |

Figure 34. Data Set 3.
Figure 35. Minimum partition for Data Set 3.
Figure 36. Minimum cover for Data Set 3.
<table>
<thead>
<tr>
<th>16</th>
<th>7</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 4</td>
<td>3 3</td>
<td>2 2</td>
</tr>
<tr>
<td>5 2</td>
<td>5 3</td>
<td>8 4</td>
</tr>
<tr>
<td>3 5</td>
<td>2 6</td>
<td>6 5</td>
</tr>
<tr>
<td>1 7</td>
<td>8 8</td>
<td>4 7</td>
</tr>
<tr>
<td>1 1</td>
<td>4 8</td>
<td>6 6</td>
</tr>
</tbody>
</table>

Figure 37. Data Set 4.
Figure 38. Minimum partition for Data Set 4.
Figure 39. Minimum cover for Data Set 4.
Figure 40. Data Set 5.
Figure 41. Minimum partition for Data Set 5.
Figure 42. Minimum cover for Data Set 5.
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>11</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>71</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>92</td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>53</td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>73</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>94</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>86</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>37</td>
<td></td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>87</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>98</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 43.** Data Set 6.
Figure 44. Minimum partition for Data Set 6.
Figure 45. Minimum cover for Data Set 6.
Figure 46. Data Set 7.
Figure 47. Minimum partition for Data Set 7.
Figure 48. Minimum cover for Data Set 7.
<table>
<thead>
<tr>
<th>28</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
</tbody>
</table>

**Figure 49.** Data Set 8.
Figure 50. Minimum partition for Data Set 8.
Figure 51. Minimum cover for Data Set 8.
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>35</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>46</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>72</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>64</td>
<td>84</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>56</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>37</td>
<td>612</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>11</td>
<td>12</td>
<td>57</td>
</tr>
<tr>
<td>107</td>
<td>11</td>
<td>17</td>
<td>8</td>
</tr>
<tr>
<td>13</td>
<td>6</td>
<td>38</td>
<td>88</td>
</tr>
<tr>
<td>98</td>
<td>510</td>
<td>610</td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>910</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td>39</td>
<td>1210</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>13</td>
<td>10</td>
<td>49</td>
</tr>
<tr>
<td>311</td>
<td>69</td>
<td>511</td>
<td></td>
</tr>
<tr>
<td>812</td>
<td>12</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

Figure 52. Data Set 9.
Figure 53. Minimum partition for Data Set 9.
Figure 54. Minimum cover for Data Set 9.
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>11</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>71</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>92</td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>53</td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>73</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>94</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>86</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>37</td>
<td></td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>87</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>98</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 55. Data Set 10.
Figure 56. Minimum partition for Data Set 10.
Figure 57. Minimum cover for Data Set 10.
## Summary of Results

<table>
<thead>
<tr>
<th>Data set #</th>
<th>Data set</th>
<th># of holes</th>
<th>Minimum partition size</th>
<th>Minimum cover size</th>
<th>Time taken to find minimum partition cover (time unit: 1/100 second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>36</td>
<td>1</td>
<td>12</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>22</td>
<td>2</td>
<td>7</td>
<td>7</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>1</td>
<td>8</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>30</td>
<td>2</td>
<td>10</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>28</td>
<td>6</td>
<td>9</td>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>9</td>
<td>44</td>
<td>2</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>10</td>
<td>24</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>11</td>
</tr>
</tbody>
</table>

Presentation of results
5.0 Conclusions

The results presented in this thesis give a complete description of a method for obtaining a minimum rectangle partition and a minimum rectangle cover for a rectilinear figure with or without holes. The results produced by the algorithm outlined in Chapter 3 are illustrated by running the algorithm for several data sets. From these we see that the number of rectangles in a minimum rectangle cover for a given figure is less than or equal to the number of rectangles in a minimum rectangle partition for the same figure. This fact is intuitively obvious, since any partition of a figure is also a cover for that figure; hence we cannot have more rectangles in a minimum rectangle cover than in a minimum rectangle partition for any given figure. In some cases the number of rectangles in a minimum rectangle cover is equal to the number of rectangles in a minimum rectangle partition for a figure (see for example the figures given by data sets 3, 5, 7, 9, 10 in Chapter 4). However, 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 for the algorithm presented in this thesis. 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 mini-
mum 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 [6], 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-

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 par-
titioning algorithm for this class of figures.

Another field for further research is suggested by Data Set 8 (see Figure 50 on page
102). 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 [17] established a proof in 1979
of the fact that this problem is NP-complete. It may thus be desirable to concentrate on de-
veloping fast approximation algorithms which will give a near-optimal solution to the problem
of deriving a minimum rectangle cover for rectilinear figures. Here too, there are some open
questions; for instance, we have not shown that all $c_1n^2 \times c_2n^4$ (where $c_1$ and $c_2$ are constants)
Boolean matrices are possible as charts in our chart processing method. If they are not, then an investigation into the properties of these charts could lead to more refined and therefore more efficient methods for the solution of the problem of finding minimum rectangle covers for rectilinear figures.
References


The vita has been removed from the scanned document.