Topological Routing Of Polygonal Circuit Modules A Computational Geometry Appro~~a~~ch

A dissertation submitted
in partial fulfilment of the requirements for the
M.Tech. (computer science) degree
of the
Indian Statistical Institute

By
Padma Mahalingam

Under The Supervision of
Dr. Subhas C. Nandy and *Dr.(Mrs.) Sushmita S.Kolay*

Certificate Of Approval

This is to certify that the thesis titled **Topological Routing Of Polygonal Circuit Modules - A Computational Geometry Approach** submitted by **Miss. Padma Mahalingam** , towards partial fulfilment of the requirements for the degree of M.Tech. in Computer Science at the Indian Statistical Institute , Calcutta , embodies the work done under our supervision .

[illegible]
(Sushmita S. Kolay)

[illegible]
(Subhas C. Nandy)

Acknowledgements

My sinceremost thanks are due to Dr. Subhas C. Nandy and Dr.(Mrs) Sushmita Sur-Kolay, without whom this work would never have been a reality.

I would like to thank all the staff here who provided me with encouragement in work, and made the course work interesting.

I would also like to thank all my classmates for their wonderful company making the two years enjoyable.

A special word of thanks to Miss Rekha Menon, Mr. Piyush Kumar, Mr. Manas Ranjan Jagdev, Mr. Anirvan Basu for their help in completing my project work.

Finally, I would also to take this opportunity to gratefully acknowledge the help provided to me by Electronics unit and the computing facilities provided to me by Computer and Statistical Service Center in I.S.I.

Abstract

In this thesis we propose the topological routing of a floorplan where the circuit modules are of convex polygonal shape. We assume that a net will never appear more than once in a circuit module and the pins corresponding to nets are placed along the edges of the polygonal circuits. The objective is to route the nets such that a specific distance is maintained between the wires corresponding to two different nets. We first partition the free space into trapeziums and along the edges of these trapeziums a capacity constraint is assigned to satisfy the objective function. We use line sweep approach to design a multipass heuristic algorithm for routing the nets concurrently, i.e. considering all nets together. Since standard benchmarks are not available, floorplans containing polygonal modules are generated randomly and the algorithm is tested. The algorithm works very fast and for most of the examples of moderate size it outputs 100 % routing in a pair of Top to Bottom and Bottom to Top sweep. For some examples more than two passes are required. The limitation of assigning the considered capacity constraint to satisfy the objective function regarding the separation of routing wires is mentioned in the conclusion of the thesis.

Contents

Chapter 1

Introduction :

A VLSI chip may contain several million transistors. As a result, tens of thousands of nets have to be routed to complete the layout. In addition there may be several hundreds of possible routes for each net. This makes the routing problem computationally hard. In fact, even when the routing problem is restricted to channels, it cannot be solved in polynomial time i.e. the channel routing problem is NP-complete. [1], i.e. it cannot be solved in polynomial time. Therefore, routing has traditionally been divided into two phases. The first phase is called *global routing* and generates a 'loose' route for each net. In fact it assigns a list of routing regions to each net without specifying the actual geometric layout of wires (Figure - 1 a). The second phase, which is called *detailed routing* , finds the actual geometric layout of each net within the assigned routing regions (Figure - 1 b). Unlike global routing which considers the entire layout, a detailed router considers just one region at a time. The exact layout is produced for each wire segment assigned to a region, and vias are inserted to complete the layout. Detailed routing includes channel routing and switchbox routing. Another approach to routing is called *Area Routing* which, is a phase

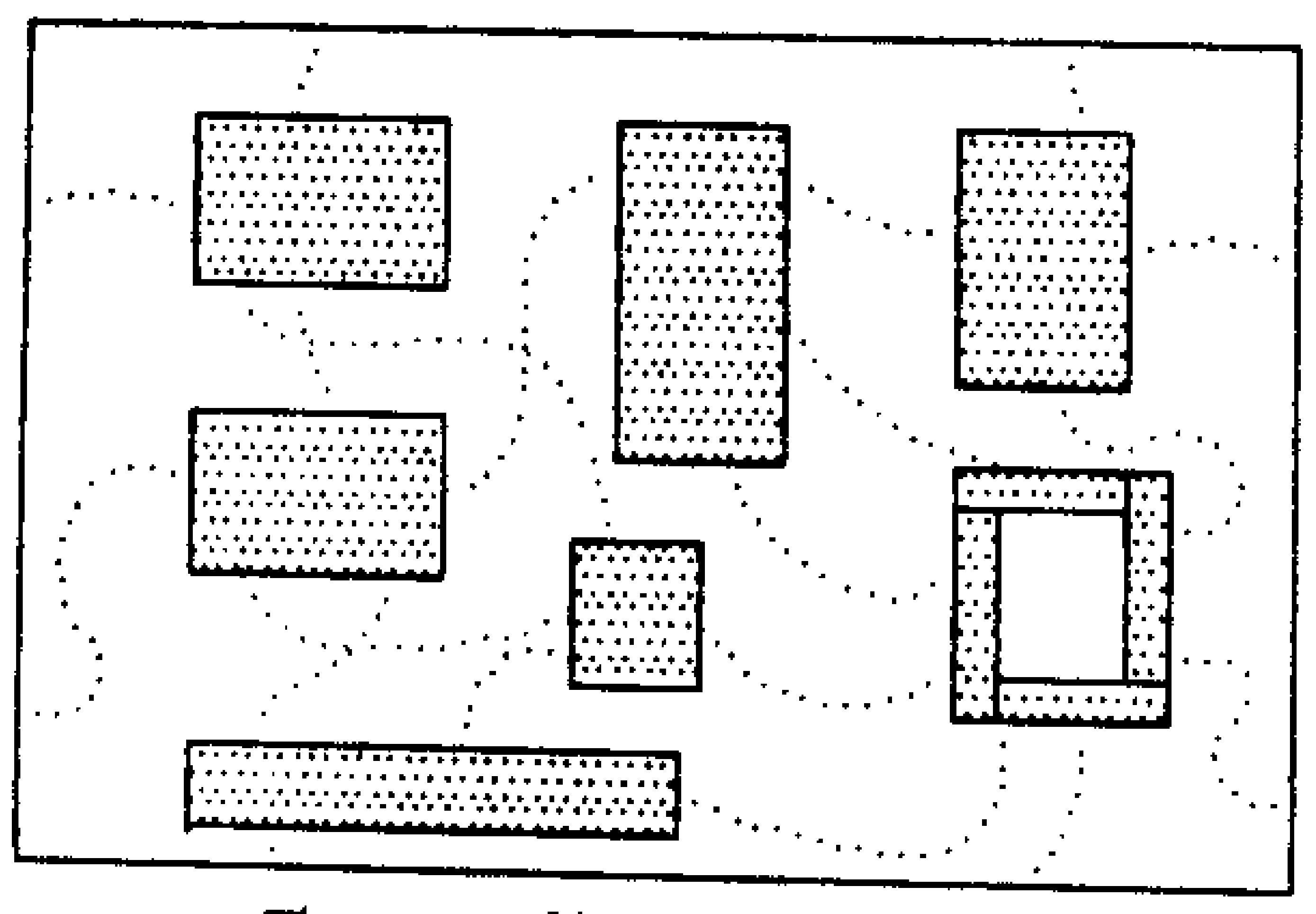
Fig 1a. Global Routing

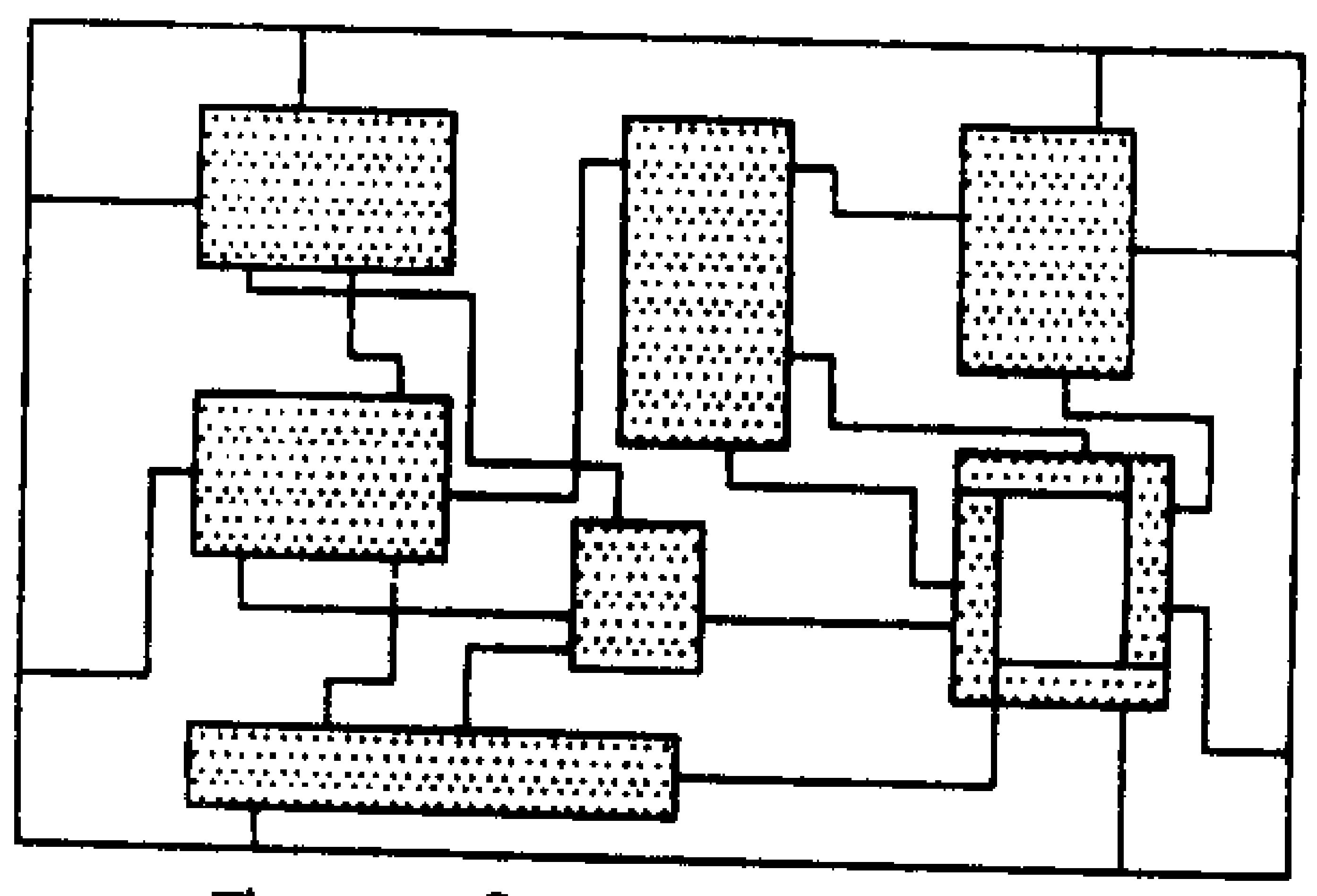
Fig 1b. Detailed Routing

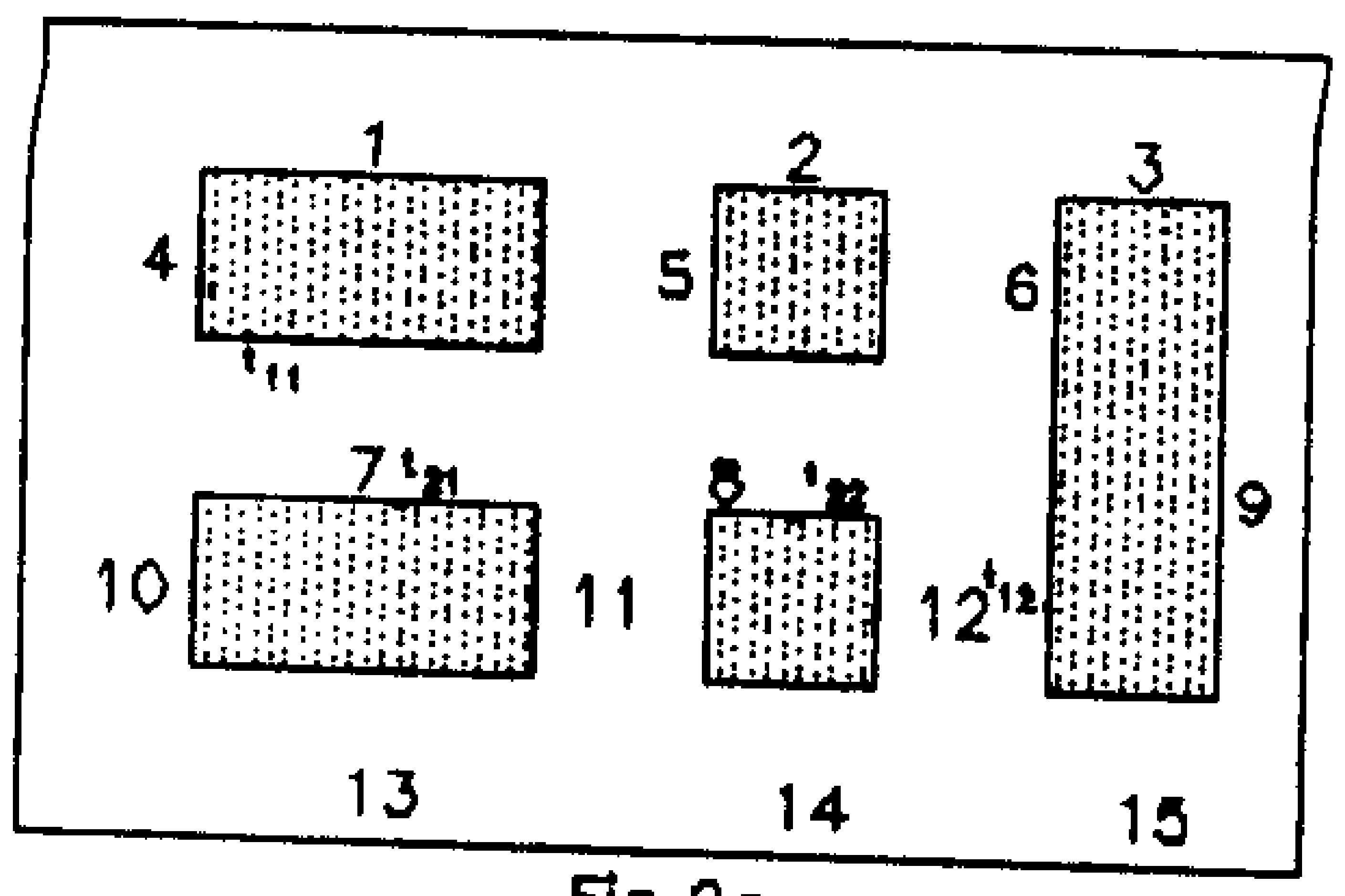

Fig 2a

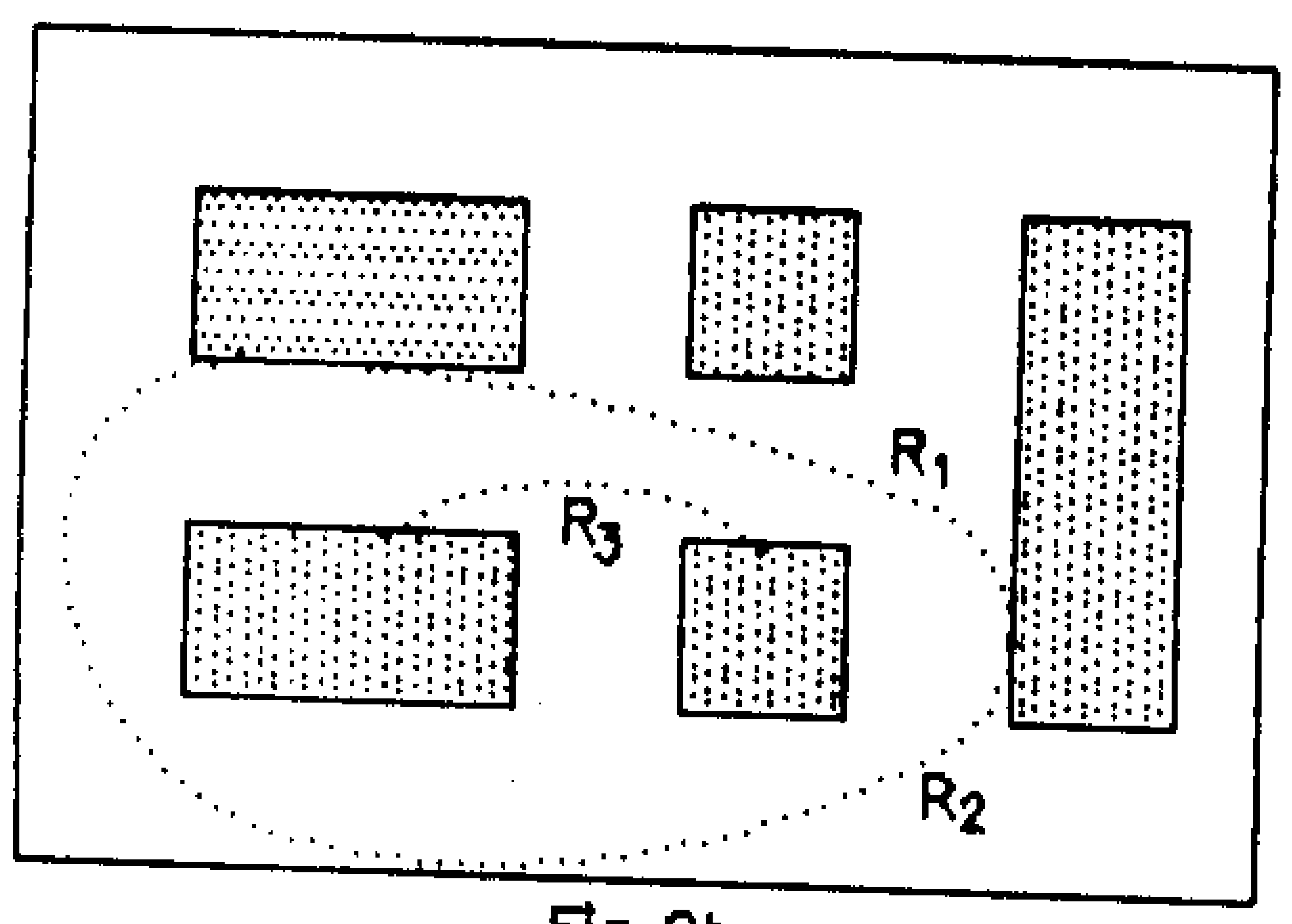

Fig 2b

routing technique. However, this technique is computationally infeasible for general VLSI circuits and is typically used for specialized problems. Here we discuss techniques for global routing.

The following example illustrates the main concepts of global routing.

In this example each channel is supposed to be of unit capacity. Routing of two nets $N_1 = \{ t_{11}, t_{12} \}$ and $N_2 = \{ t_{21}, t_{22} \}$. There are several possible routes for net N_1. Two such routes R_1 and R_2 are shown in (Figure-2). If the choice is to route just N_1 then R_1 is a better choice. If both N_1 and N_2 are to be routed, it is not possible to use R_1 for N_1 since it would make N_2 unroutable. Thus global routing is computationally hard since it involves trade-offs between routability of all nets and minimization of the objective function. In fact, global routing of even a single multi-terminal net is NP-complete.

There are two kinds of approaches to solve global routing problem.

1. Sequential
2. Concurrent

Sequential Approach : In this approach nets are routed one by one. Once a net has been routed it may block other nets which are yet to be routed. So this result is very sensitive to the order in which the nets are considered for routing. Usually the nets are sequenced according to their criticality, perimeter of the bounding rectangle and number of terminals.The criticality of a net is determined by the importance of the net. The criticality and other factors can be used to sequence the nets. Actually these sequencing techniques do not solve the net ordering problem. In a practical router in addition to a net ordering scheme an improvement phase is used to remove blockages when further routing of nets is not possible. Even this may not overcome the shortcomings of sequential approach. One such improvement phase involves 'rip - up and reroute ' technique, [2, 3] while other involves 'shove aside' technique. Another approach has been suggested in [4].

The Sequential approach includes :

a) Two - terminal algorithms :

i. Maze routing algorithms.

ii. Line - probe algorithms.

iii. Shortest path based algorithms.

b) Multi - terminal algorithms :

i. Steiner tree based algorithms

Maze Routing Algorithms :

Lee [5] introduced an algorithm for routing two terminal net on a grid in 1961. Since then the basic algorithm has been improved for both speed and memory requirements. Lee's algorithm and its various improved versions form the class of maze routing algorithms.

Maze routing algorithms are used to find a path between a pair of points, called the source (s) and the target (t) respectively, in a planar rectangular grid graph. The geometric regularity in the standard cell and gate array design style lead us to model the whole plane as grid. The areas available for routing are represented as blocked vertices. The objective of a maze routing algorithm is to find a path between the source and the target vertex without using any blocked vertex. The process of finding a path begins with the exploration phase, in which several paths start at the source, and are expanded until one of them reaches the target. Once the target is reached, the vertices need to be retraced to the source to identify the path. The retrace phase can be easily implemented as long as the information about the parentage of each vertex is kept during the exploration phase. Several methods of path exploration have been developed.

Line - Probe Algorithms :

The line - probe algorithms are developed independently by Mikami and Tabuchi in 1968 [6] and Hightower in 1969 [7]. The basic idea of a line probe algorithm is to reduce the size of memory requirement by using line segments instead of grid nodes in the search. The time and space complexities of these line - probe algorithms is O(L), where L is the number of line segments produced by these algorithms.

The basic operations of these algorithms are as follows. Initially lists *slist* and *tlist* contain the line segments generated from the source and target respectively. The generated line segments do not pass through any obstacle. If a line segment from *slist* intersects with a line segment in *tlist* , the exploration phase ends ; otherwise, the exploration phase proceeds iteratively. During each iteration new line segments are generated. These segments originate from 'escape ' points on existing line segments in *slist* and *tlist* . The new line segments generated from *slist* are appended to *slist*. Similarly, segments generated from a segment in *tlist* are appended to *tlist* . If a line segment from *slist* intersects a line segment from *tlist* , then the exploration phase ends. The path can be formed by retracing the line segments in set *tlist* , starting from the target, and then going through the intersection, and finally retracing the line segments in set *slist* until the source is reached.

Shortest Path Based Algorithms :

A simple approach to route a two - terminal net uses Dikjstra's shortest algorithm [8]. Given a routing graph G = (V, E) a source vertex $s \in V$ and a target vertex $t \in V$ a shortest path in G joining s and t can be found in $O(|V|^2)$ time.

The length of an edge is increased by a factor $\alpha > 1$ whenever a congested edge is utilized in the path of a net. If the edge congestions are strict, the algorithm can be modified to use 'rip - up and reroute ' or 'shove aside ' techniques [2, 3].

Concurrent Approach :

This approach avoids the ordering problem by considering routing of all the nets simultaneously. The concurrent approach is computationally hard and no efficient polynomial algorithms are known even for two terminal nets. The concurrent approach takes a global view of all the nets to be routed at the same time. This approach requires use of computationally expensive methods. One such method uses integer programming. Integer program for an overall problem is normally too large to be handled efficiently. Thus, hierarchial top down methods are used to break the problem into sub - problems. These smaller sub - problems can be solved efficiently. The solutions are then combined to obtain the solution of original global routing problem. One such hierarchial based integer program for global routing has been presented by Heisterman and Lenguar [9].

Several Hieuristics have been proposed to these problems.

Usually VLSI chips are all rectangular in shape. In near future with the development of MCM's and need for compaction there may be trapezoidal or in fact arbitrary polygonal shaped chips.

Keeping this in mind we have tried to develop a concurrent approach to solve the problem of routing multi terminal nets placed along the edges of polygonal chips in two layers. To solve this some computational geometry techniques have been used.

Chapter 2

Problem Description :

Let P = { $P_1, P_2, ..., P_k$ } be a set of k non overlapping convex polygonal circuit modules distributed over the floorplan. For the sake of simplicity, let us assume that the boundary of the floorplan is rectangular.

Let N = { $n_1, n_2, ...n_m$ } be a set of m nets attached to different circuit components.

For each polygon P_i, a subset N_i of the set of nets N is attached. The pins of those nets appear along the boundary of P_i and for simplicity, we assume that each net of N_i appears only once on P_i. The gap between two adjacent pins corresponding to two different nets satisfies the required constraints i.e. the routing wires should be non-overlapping assuming the specified width of the routing wire and the capacity constraint of the region. Also its easy to see that for each net n_i we can associate a subset of P along which the net is placed.

Example - Figure (3).

P = { P_1, P_2, P_3 } N = { n_1, n_2, n_3 }

Along with each nets we can associate the following subsets :

$n_1 \rightarrow \{ P_1, P_2 \} \subseteq$ P $n_2 \rightarrow \{ P_1, P_3 \} \subseteq$ P $n_3 \rightarrow \{ P_3, P_2 \} \subseteq$ P

Along with each polygon we associate subsets as follows :

$P_1 \rightarrow \{ n_1, n_2 \} \subseteq$ N $P_2 \rightarrow \{ n_1, n_3 \} \subseteq$ N $P_3 \rightarrow \{ n_2, n_3 \} \subseteq$ N

Our objective is to route the nets.

In order to route a particular net, all the pins distributed in different polygonal modules are to be connected by wire.

We have to ensure nonoverlapping and capacity constraints are taken care in our strategy.

Overlapping :

In actual routing we shall use two layers. So we may allow the wires corresponding to different nets to overlap once in their routing path, indicating connections are made in the second layer in case of overlapping wires.

Capacity constraint :

The routing wires are of specified width. Again due to the conductivity constraints, a minimum separation of specified width needs to be maintained between two wires running parallelly accross any line. Now consider a line segment whose each of the two end terminal lie on the boundary of two different polygons or the boundary of the floorplan, which does not intersect any polygon in its interior. The maximum number of wires passing through it can be obtained from the width of the wire and the minimum separation constraint. This number may be considered as a capacity constraint of the line segment.

The objective is to route each net satisfying the capacity constraint everywhere on the floorplan.

This original problem leads to two subproblems :

1. In order to ensure that the capacity constraint is satisfied everywhere on the floorplan, identifying the minimum number of straight line segments

satisfying the above property is important.

2. Routing the floorplan satisfying the capacity constraint of those straight line segments.

Chapter 3

Problem Formulation :

Consider the floorplan with k non overlapping convex polygons. In order to find the topology of the global routing we partition the free space into a set of convex polygonal regions as follows :

• Select all the edges of the k convex polygons one by one and in any order. Both the end points of the selected edge is then considered separately. If that end point is unmarked then it is extended until it hits another line (a polygonal edge / another extended line / floorplan edge).

Remark - 1 : The convex partitioning obtained in this process is not unique ; it depends on the order of choosing the edges of the polygons.

Remark - 2 : The number of convex partitions is same for any order of choice of the edges of the polygons.

Detailed explaination is discussed in subsequent chapter.

Partitioning the free regions into convex partitions as stated above is a difficult problem. The problem can be observed in the following way :

Given a point and a direction, fire a ray and locate an existing line which it hits first. Given a set of n nonintersecting line segments in the plane the algorithm in [10] constructs an O(n $log^3 n$) size in time O($n^{3/2} log^w n$) (w <

4.3) , and computes the first intersection point by a ray originated from a given point and in a given direction in O($\sqrt{n} log^2 n$) time.

But in our situation, after generating a line by ray shooting as stated above, it is to be inserted in the data structure. No standard method of maintaining a dynamically growing data structure for this problem is observed in the literature and it becomes difficult to solve.

So we have tried with a simpler form of partitioning i.e. the trapezoidal decomposition of the free space. Since this general problem of global routing, where the circuit modules are convex polygons, is proposed for the first time, no benchmark examples are available. The experiments need to be performed with the randomly generated benchmarks.

It is also very difficult to generate a set of non overlapping convex polygons randomly. So we have limited our experiment with a set of circuit modules each of which is a convex quadrilateral.

The problem is analysed in three steps :

1. Random Floorplan Generation and Pin Assignment.
2. Decomposition of the free space.
3. Routing.

Chapter 4

Brief Study On Simpler Version Of The Problem :

Before approaching the main problem we test whether the objective of 100 % routability is actually achievable.

Even if the capacity constraint is not violated we find this desired routing is not always attained as its impossible to avoid overlapping of wires everytime.

We first study a simpler problem of placing the nets on the boundaries of the enclosing box.

Our aim is to route the nets with the same constraints as in the original problem.

In fact, even in the absence of obstacles, it is not necessary that all these connections can be made. We present a few examples here.

Some study on the ordering of nets , reveals certain instances when 100 percent connection cannot be obtained.

Orderings For a Single Layer :

If on making a clockwise (anticlockwise) traversal of all the four sides of the enclosing box ; we find a certain pair N_i, N_j occuring more than once (necessarily same order) then the segments connecting them will overlap.

Examples :

Figure 4 and Figure 5 shows same ordering of nets is mandatory.

Figure 6 shows that traversal of all the four sides is necessary.

Considering either AB and BC or AB and CD only we find the ordering suggests routability but BC and CD suggest otherwise. Hence more than one layer is actually needed. This shows that all four sides should be tested.

Figure 7 exhibits that (N_i, N_j) need not be consecutive.

Here (N_1, N_3) appears twice indicating one layer is insufficient in this case too. But (N_1, N_3) as such does not appear consecutively.

It can thus be inferred that while checking all the four sides, if no pair appears twice, single **layer routing is possible**.

Next, we shall test if two layers are sufficient. By two layers we mean presence of a second layer which will be an exact copy of the first layer nd whenever a wire in first layer intersects we connect them in second layer to avoid crossings.

In Figures 8 and 9, the connections within the box indicate connections in the first layer, and the connections outside the box represent connections in the second layer.

After connecting N_1 in the first layer as shown in Figure 8 N_2 cannot be connected in the first layer as it will lead to crossing of wires. Connecting it in the second layer results in Figure - 9.

Connecting N_3 in either layer will lead to crossing of wires, so, this shows that TWO LAYERS ARE ALSO INSUFFICIENT to have 100 percent connection sometimes.

The previous result, may be generalised to two layers as ' On making a traversal of four sides in clockwise / anticlockwise direction no triplet

(N_i, N_j, N_k) (NOT necessarily consecutive) should appear more than once along the edges to have 100 percent connection.

NOTE :

• If there is a repeated tuple (N_i, N_j), (N_i, N_j, N_k) we can infer that the given number (two and three) of layers are insufficient for routing but no conclusion can be made of the number of layers actually needed for routing successfully.

• Even in our original problem 100 % routing may not be possible in two layers as overlapping of wires cannot be entirely avoided as shown in Figure - 10. In such cases our aim will be to restrict to maximal routing.

Chapter 5

Pre process:

The first step to the problem is to generate a random floorplan. The boundaries of the enclosing box is given. The convex polygons of random shape are to be placed randomly in a non overlapping manner and nets are to be attached with each polygon randomly.

5.1 Floorplan Generation :

We have considered only convex quadrilaterals instead of convex polygons having arbitrary number of arms, for implementational simplicity.

In order to avoid overlapping of quadrilaterals the enclosing box of the floorplan is partitioned into arbitrary number of axes parallel rectangle, of various sizes and in each of the rectangle four points are chosen such that the quadrilateral generated with those four points form a convex quadrilateral.

Thus we generate the floorplan in the following two main steps :

- Partition the enclosing box into recangles.
- Generate convex quadrilaterals within each partition.

Rectangular decomposition of the floorplan

Let the bottom-left and top-right corners of the floorplan be (0,0) and

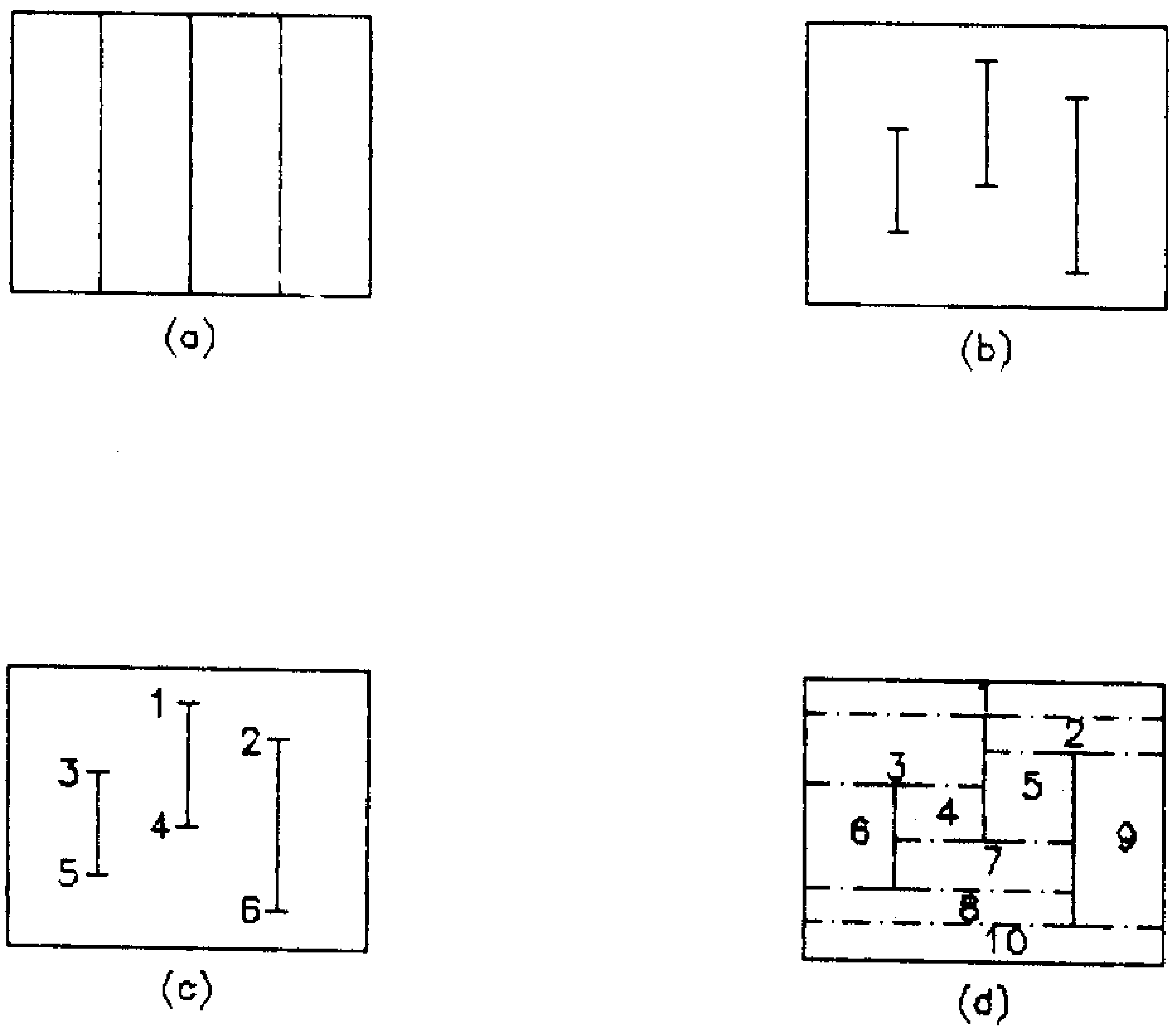

Fig 11

(α, β). The rectangular floorplan is decomposed into rectangles following the algorithm given below.

Step 1° : Generate a set of increasing x coordinate randomly within [0,α]. Figure - 11 a

Step 2° : For each chosen x-coordinate choose a pair of y coordinates in the interval $[\alpha, \beta]$ as shown in Figure - (11 b).

Step 3° : Perform a line sweep from top to bottom for generating maximal horizontal lines as explained below :

• When the sweep line encounters a top end point of a vertical line segment insert it in the AVL tree.

• When a bottom end point is encountered delete the vertical line segment delete it from the AVL tree.

• On encountering a point extend it horizontally till it hits the boundary or any other line segment in the AVL tree.

Figure - (11 d) shows the partitioned figure.

It easily seen that if there are 'K' vertical lines within the box the total number of such rectangles formed is less than or equal to 3K + 1. In case of presence of co-horizontal * points the number of rectangles formed is less than 3K + 1. Clearly as these rectangles partition the box, the quadrilaterals generated lying entirely within it, are non-overlapping.

* : Cohorizontal points are points lying on the same horizontal line.

Generation of Convex quadrilaterals inside the rectangle :

A quadrilateral is convex if and only if the diagonals intersect each other, Inside each partition we shall generate two intersecting lines lying entirely within it.

The four endpoints thus formed will be the corner points of the quadrilateral.

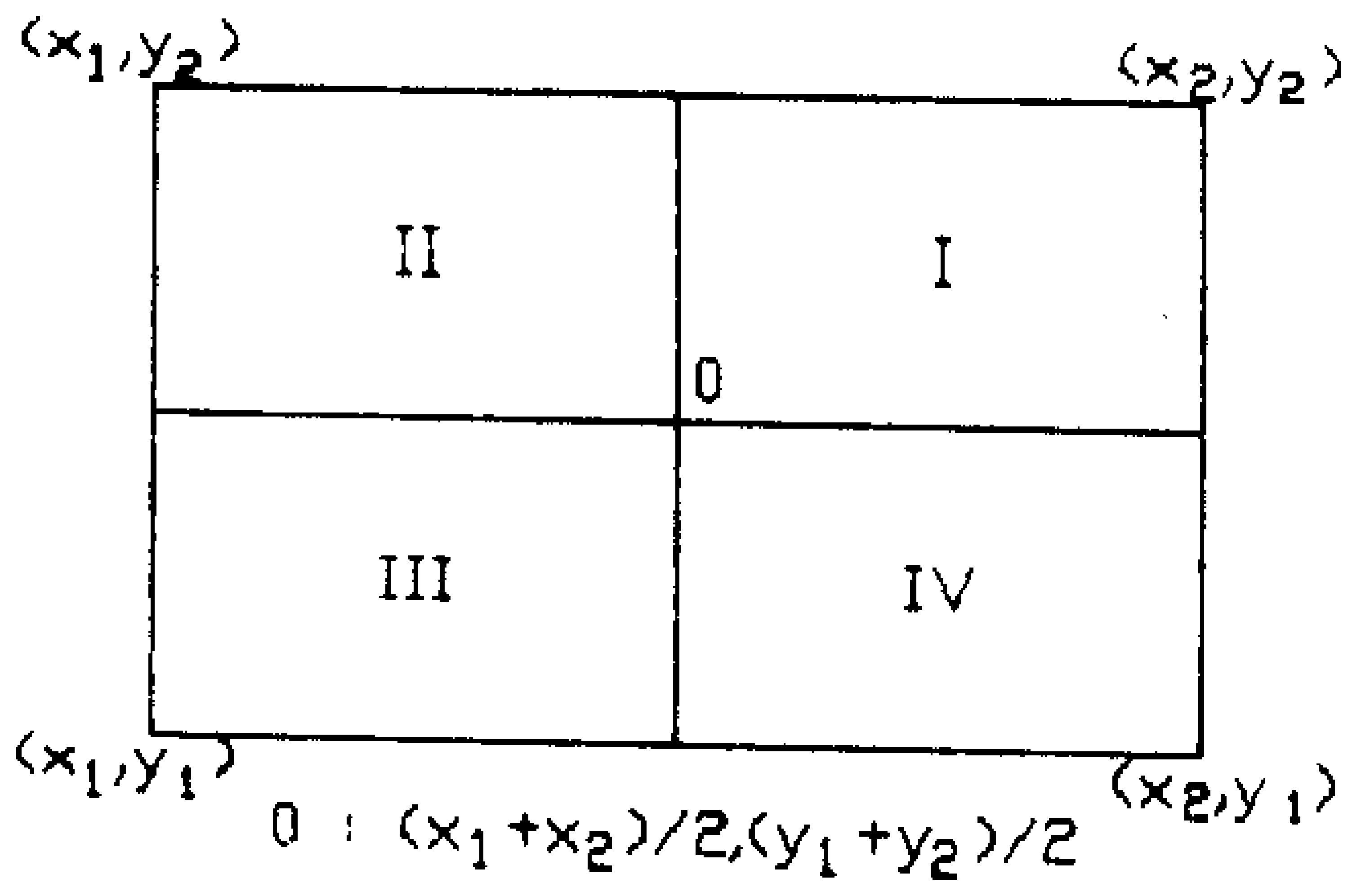

Fig 12

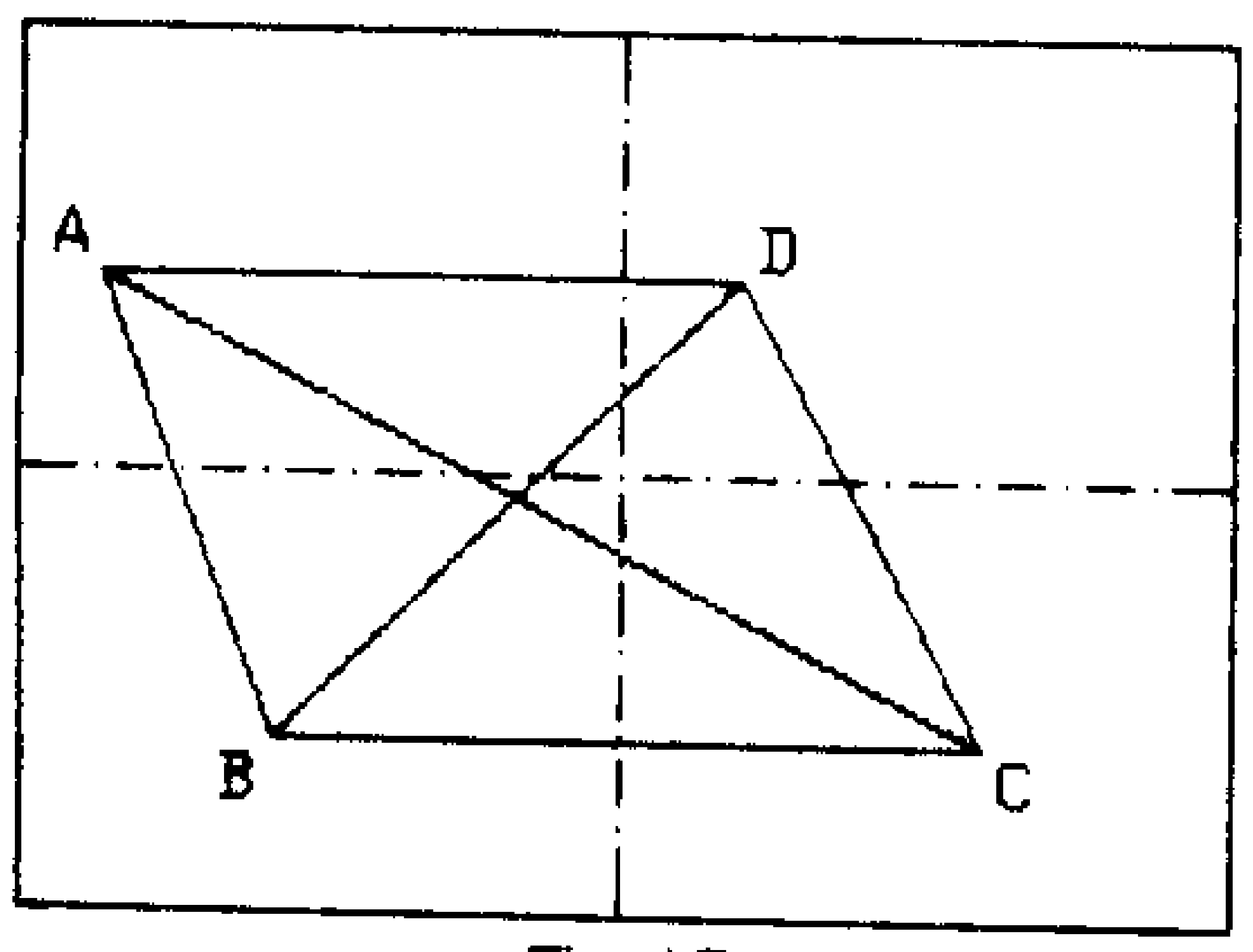

Fig 13

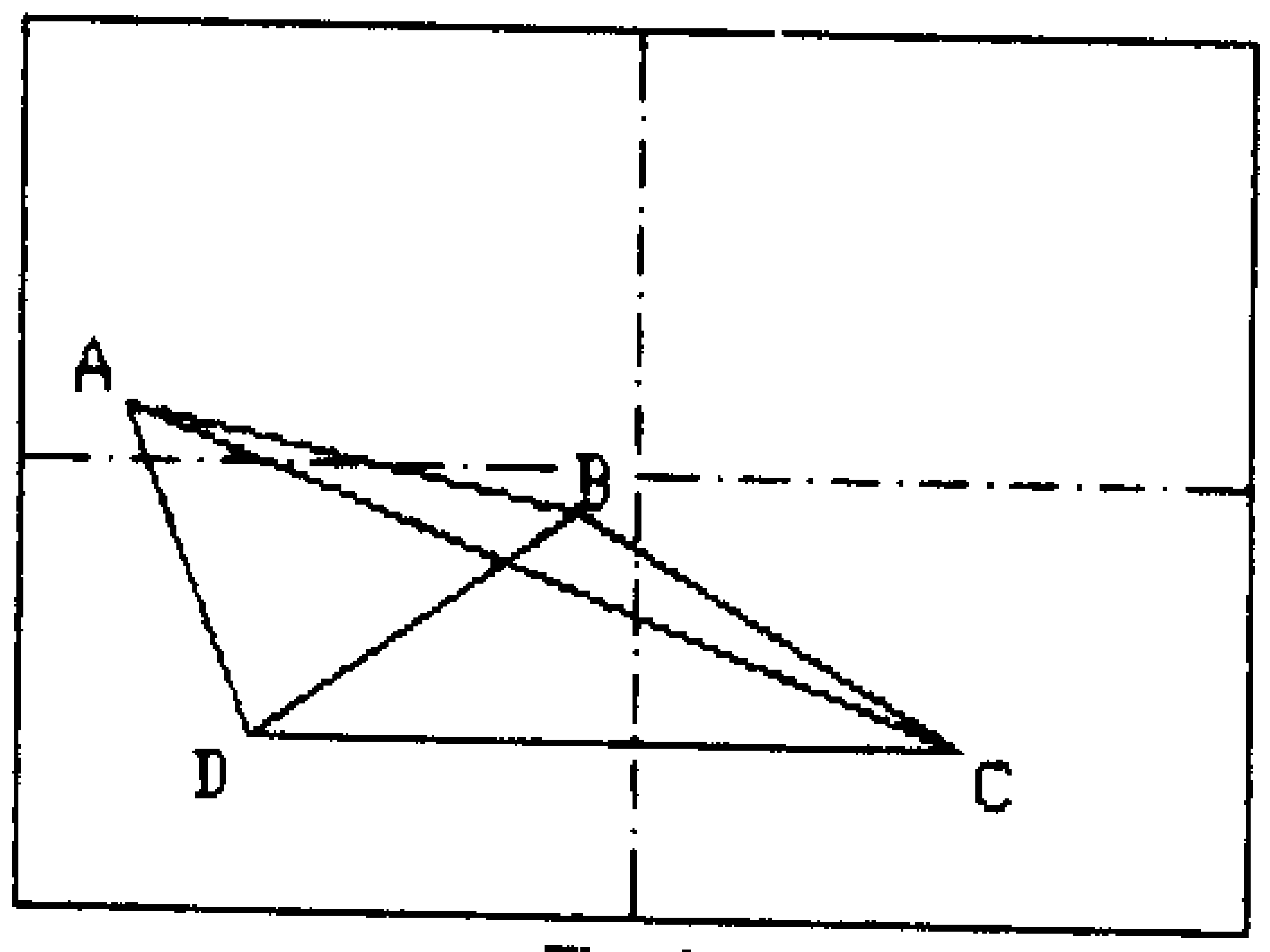

Fig 14

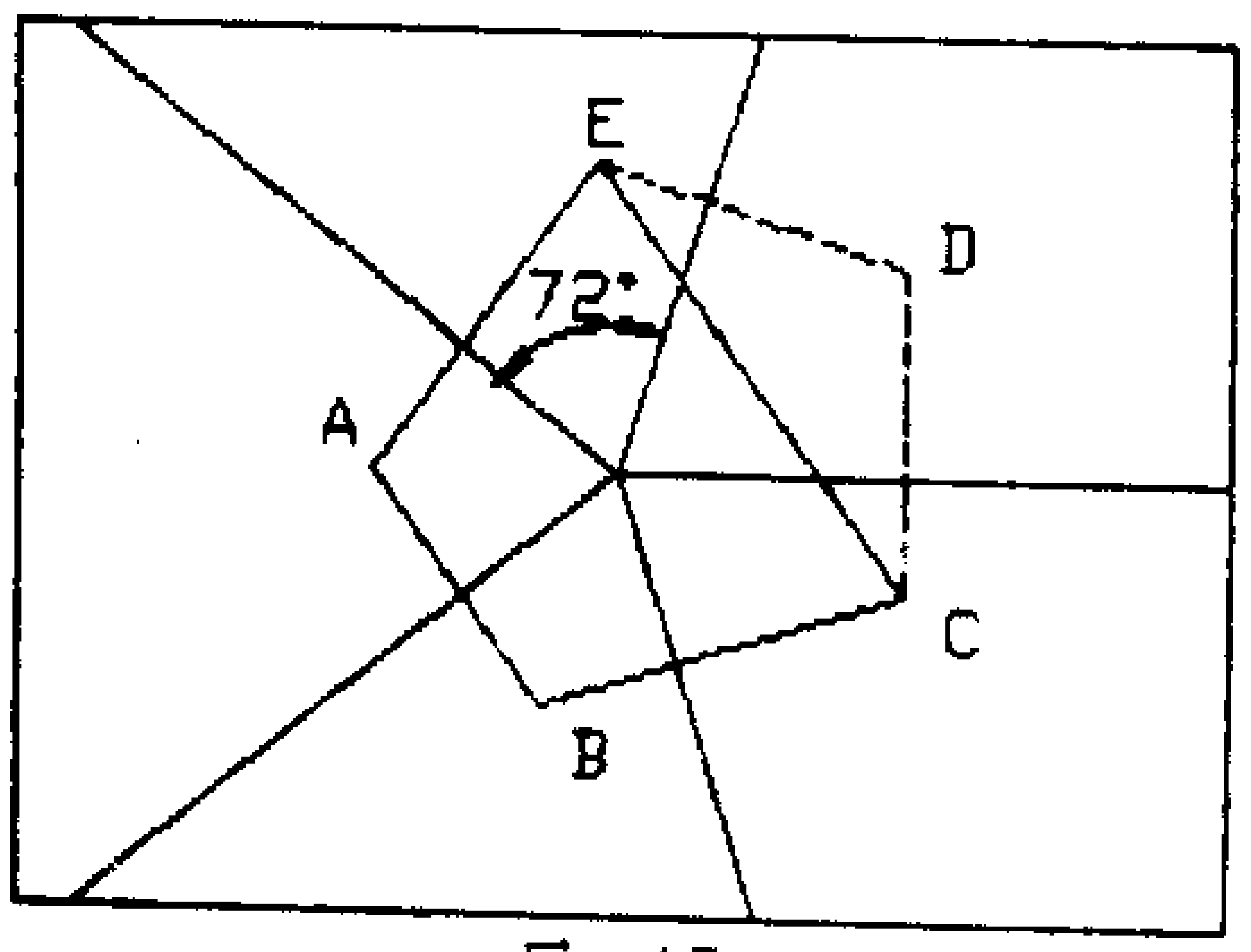

Fig 15

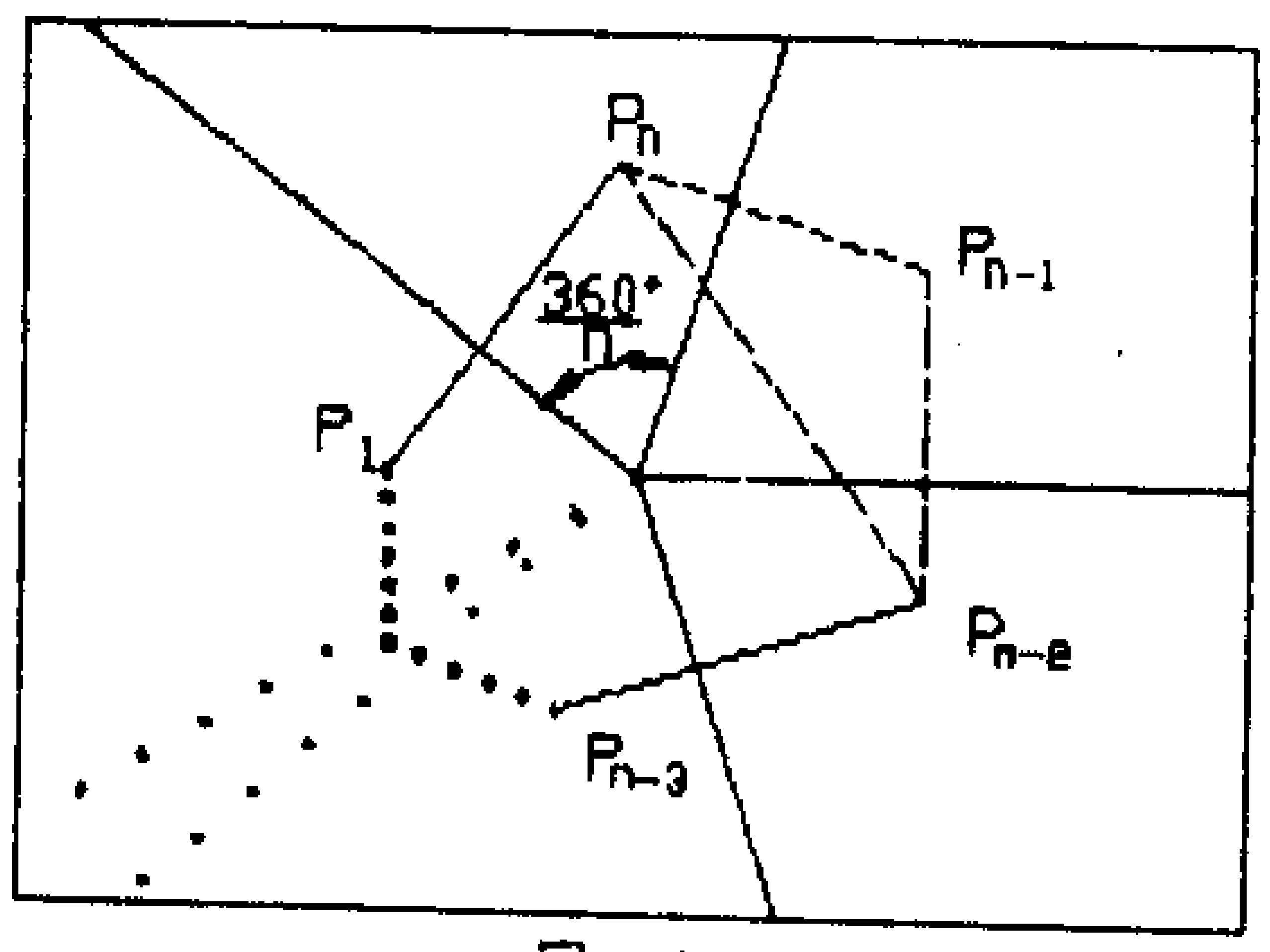

Fig 16

To form intersecting lines, we divide the rectangle into four quadrants with the midpoint of the rectangle as origin.

Choose a point in each quadrant given by the algoritm -() and join the points in the adjacent quadrants by straight lines.

Algorithm to generate convex quadrilateral within a rectangle.

ALGORITHM - 2 :

Step 1°: Form the four quadrants.

Step 2°: Choose two points A and C in second and fourth quadrant respectively.

Step 3°: Choose a point B in the third quadrant such that it is to the left of AC.

Step 4°: Choose a point D in the fourth quadrant such that BD intersects CD.

ABCD is the required convex quadrilateral.

Figures 12, 13, 14 illustrates these.

Note: In a similar fashion any convex n-polygon may be generated. For example :

Generation of a five (n)-sided polygon :-

1. Divide the rectangle into five (n) parts.

2. Choose four (n-1) parts and draw a four (n-1) sided convex polygon ABCE ($P_1P_2...P_{n-2}P_n$) by choosing points in these parts.

3. Choose a point D (P_{n-1}) on the other side (opposite to that found in step 3) of CE (P_nP_{n-2}) s.t. AD ($P_{n-3}P_{n-1}$) and BD (P_1P_{n-1}) intersects CE ($P_{n-2}P_n$).

ABCDE ($P_1P_2....P_n$) is the required five (n) sided polygon.

Figure 15, 16 illustrates these.

Once the quadrilaterals are drawn, the nets are to be placed along the sides of the polygon. This is discussed in the next section.

5.2 Pin Assignment :

In our experiment,we assume that the number of nets and the total number of pins is given. The number of pins corresponding to each net is settled randomly. The next task is :

- to assign nets corresponding to each block.
- to select sides of a polygon where the nets assigned to a particular polygon will appear.

Initially each polygonal edge is selected and is filled 50 % of its capacity. Finally for the remaining pins the polygons are selected randomly and is assigned to any of its edges selected randomly. The algorithm follows :

Algorithm - 3:

For Pin Generation

Step 1°: Using the width and distance between pins, find the capacity of each arm.

Step 2°: Choose the number of pins corresponding to each net along with the cumulative frequency.

Step 3°: ($REPEATED^{\#}$)A number less than the total is generated, find the net having its cumulative frequency lying in this range. If
pins of this net are left place it on the edge
Else
choose some other net. # : Step 3 is repeated for each edge sequentially till each edge is filled to half its capacity, then for the rest of pins an edge having capacity is randomly generated on which it will be placed.

The following example illustrates it :

```
for (i=1 ; i < DIFFERENT-PINS ; i++)
$count - of - net_i$ = rand() % (TOTAL / DIFFERENT-PINS)
s.t. (TOTAL/(2*DIFFERENT-PINS)) $\leq$
$count - of - net_i \leq$ (TOTAL/DIFFERENT-PINS)
$count - of - net_{DIFFERENT-PINS}$
= TOTAL - ($\sum_{i=1}^{DIFFERENT-PINS-1} count - of - net_i$)
```

Say the table is generated as follows :

net-number	*count*	*cumulative frequency*
1	6	6
2	5	11
3	5	16
4	9	25

A random number is generated say 14.

Then net three is placed on the concerned edge after reducing its count and capacity of the same edge by one.

Chapter 6

Decomposition :

Once the floorplan is ready and the pins are in place, the next is to decompose the obstacle free space into geometrical regions.

We discuss the following two types of decomposition here.

1. Convex polygonal decomposition.
2. Trapezoidal decomposition.

If minimality is the criteria we shall opt for the first decomposition but as already discussed we shall go for the other type for implementational simplicity.

6.1 Convex Polygonal Decomposition :

As the name suggests here the free space is decomposed into convex regions. Through each convex point of the polygon an extended edge is drawn.

By a convex point we mean the vertices on the polygon the edges on which subtend an angle less than 180 degrees.

In Figure... 17

Convex Points : A, B, D, E

Concave Points : C as $\angle BCD > 180^\circ$

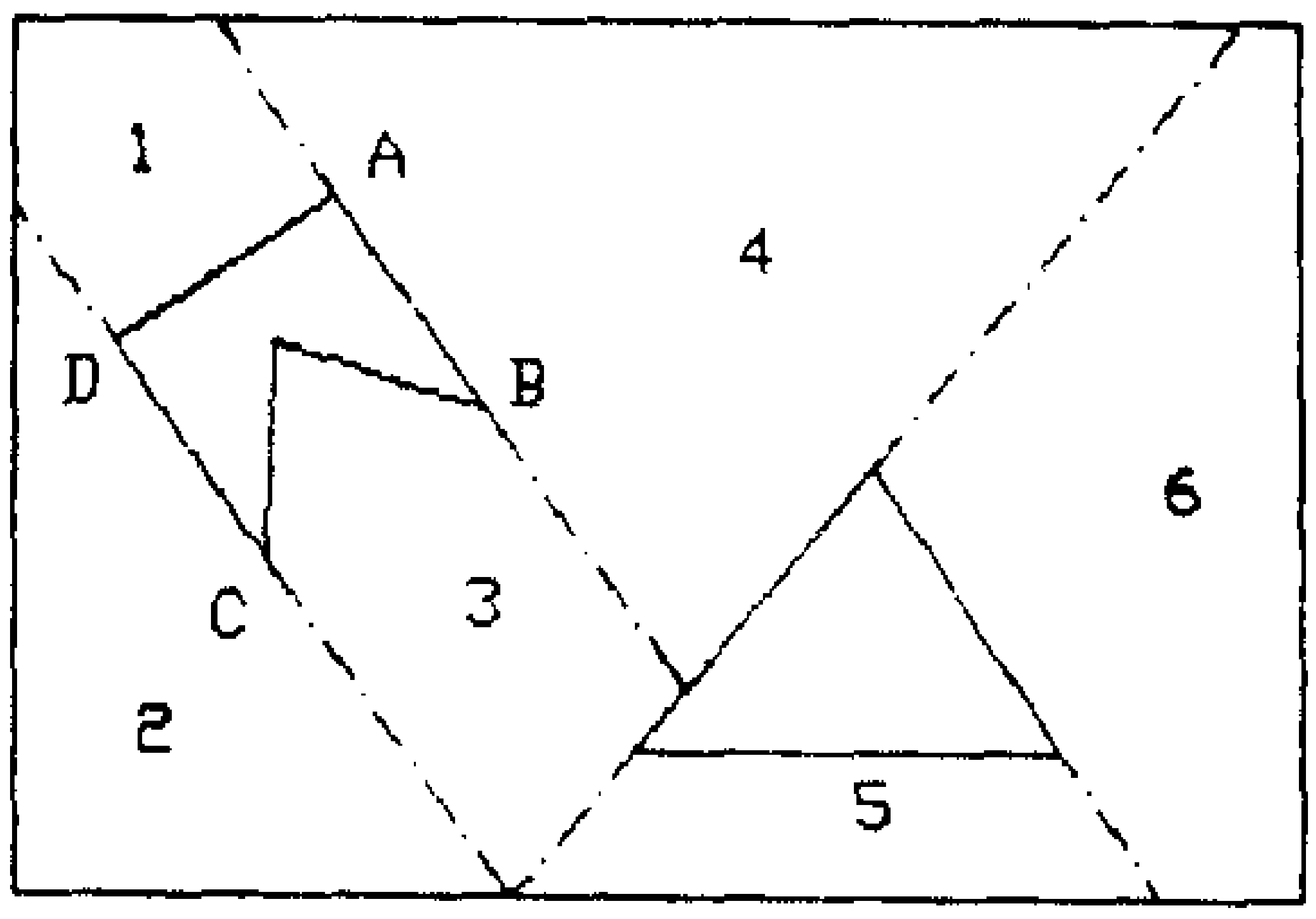

Fig 17

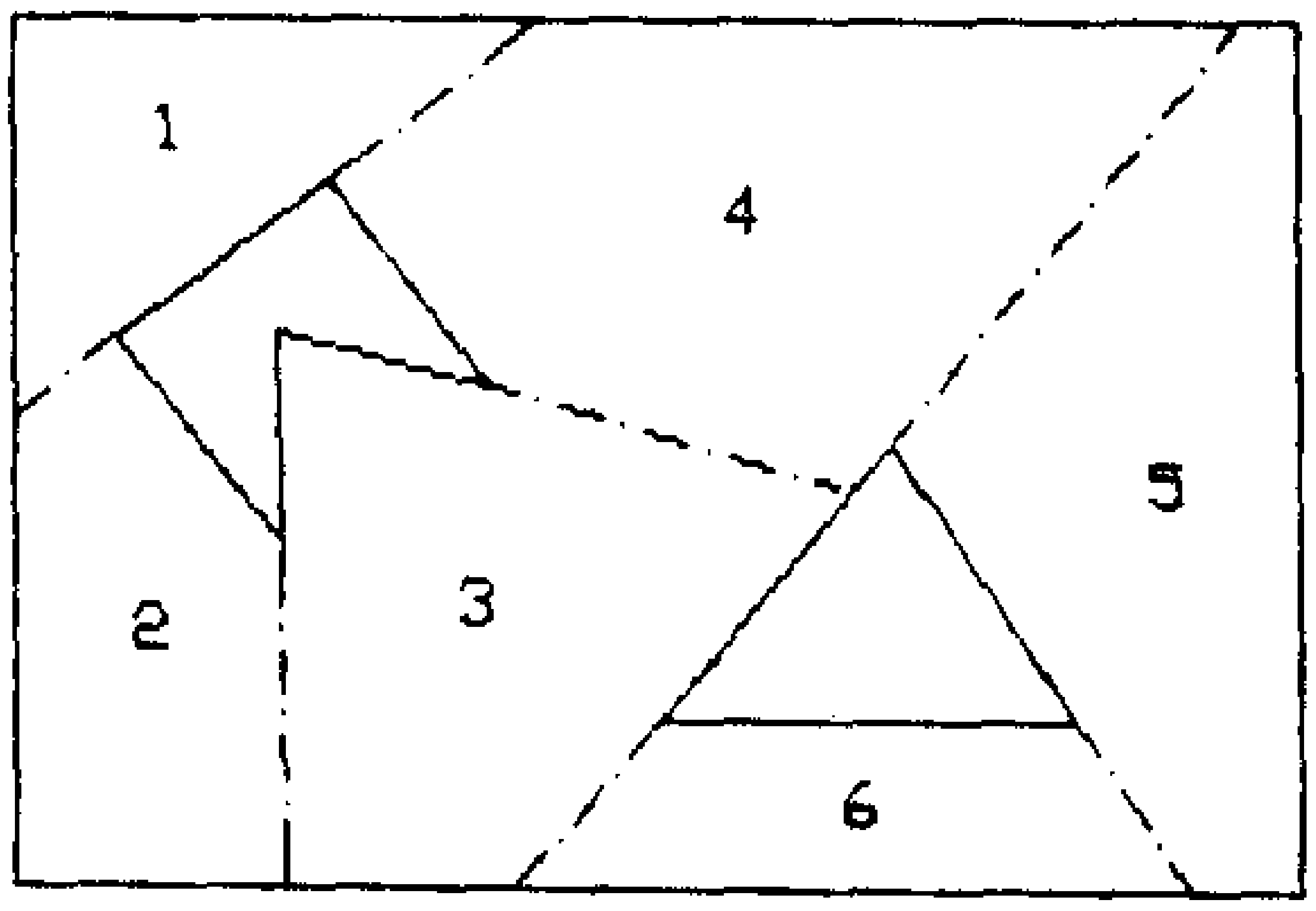

Fig 18

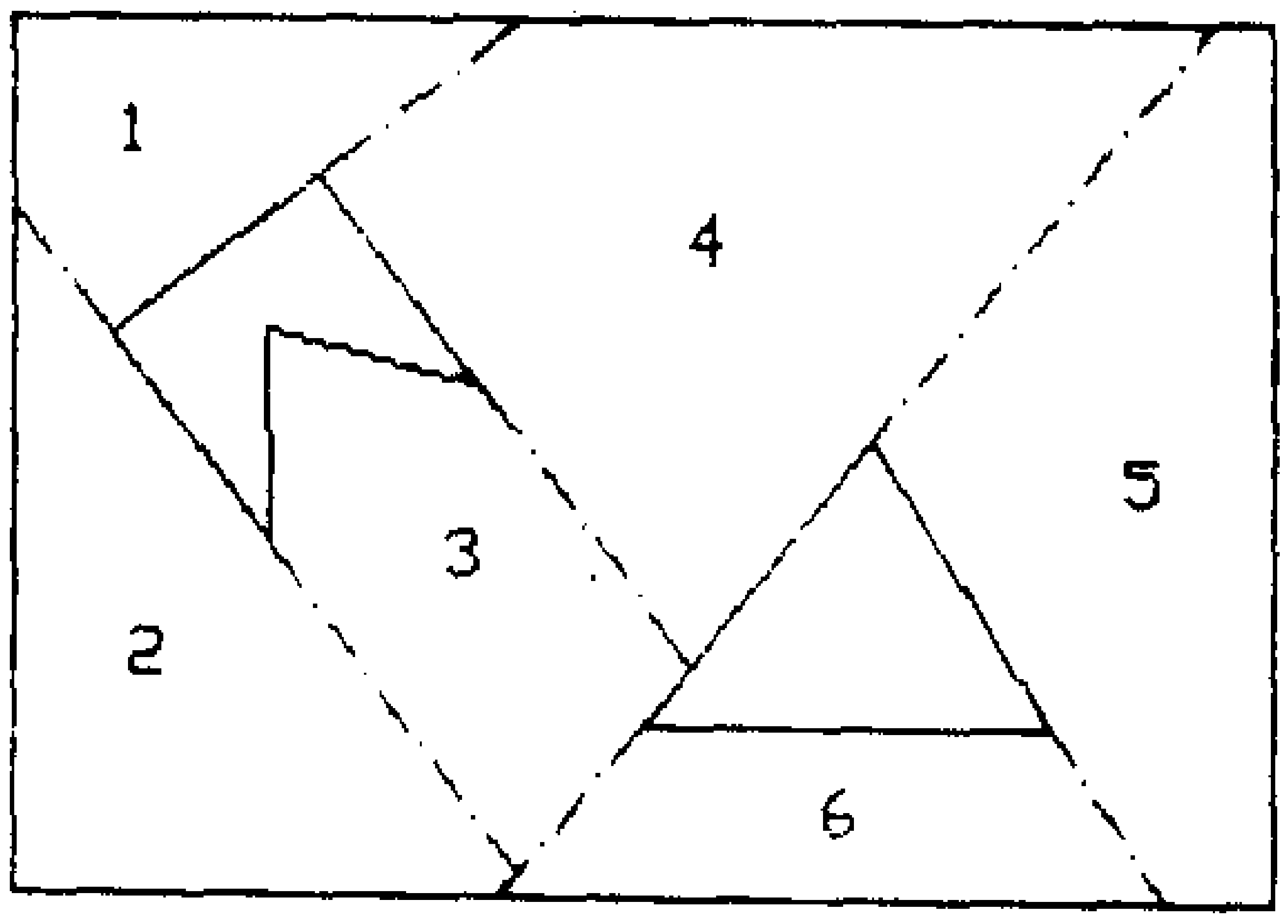

Fig 19

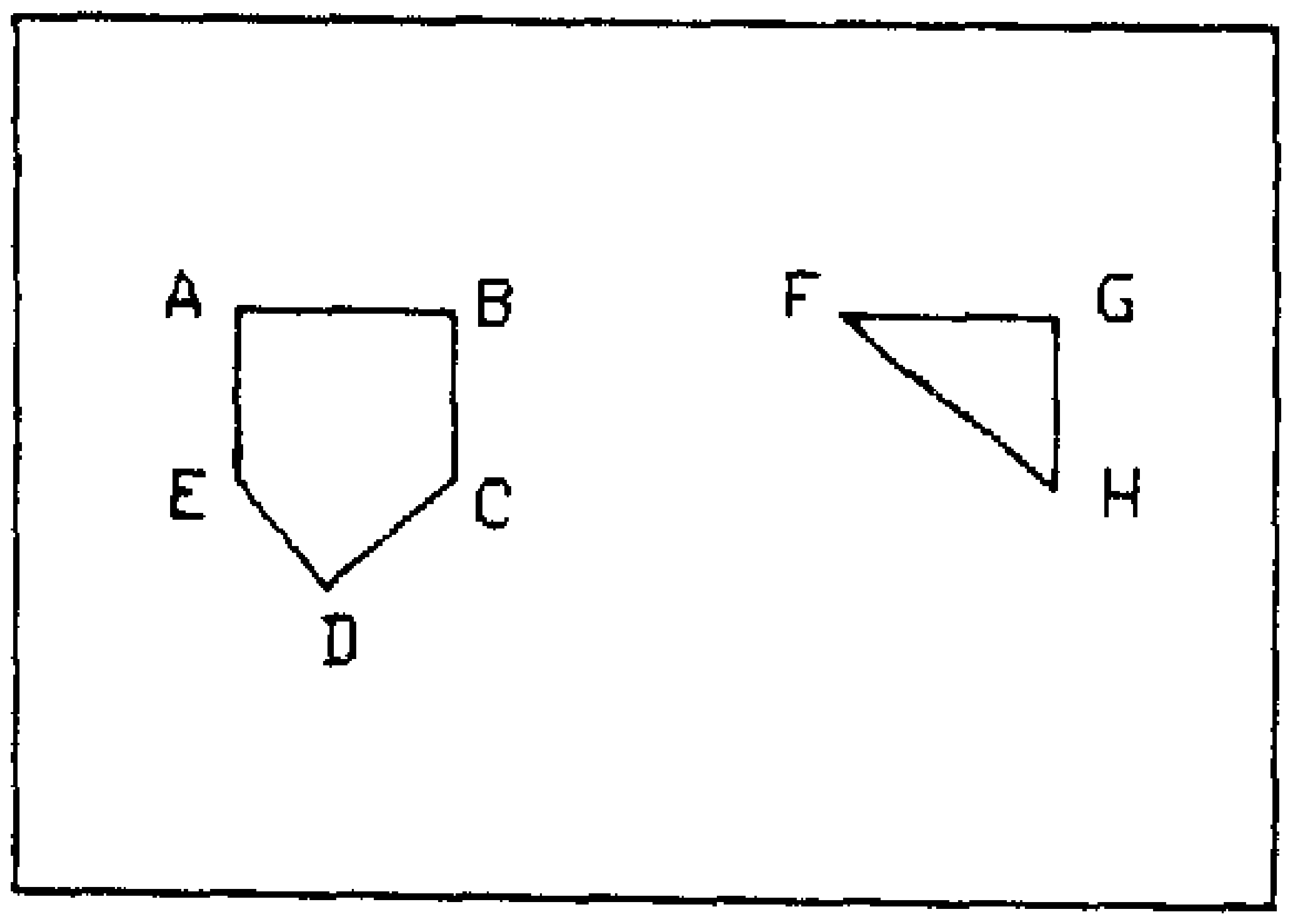

Fig 20

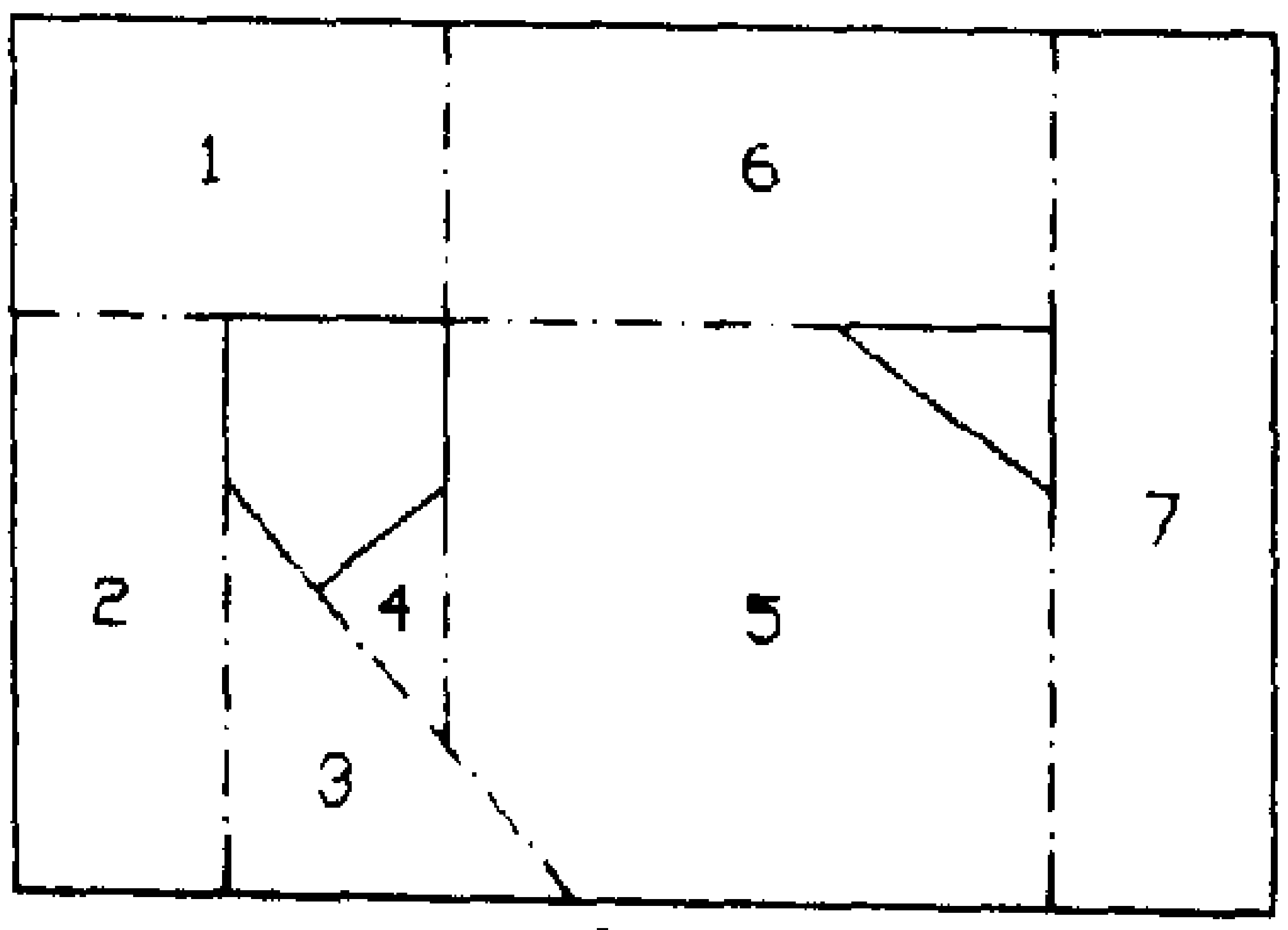

Fig 21

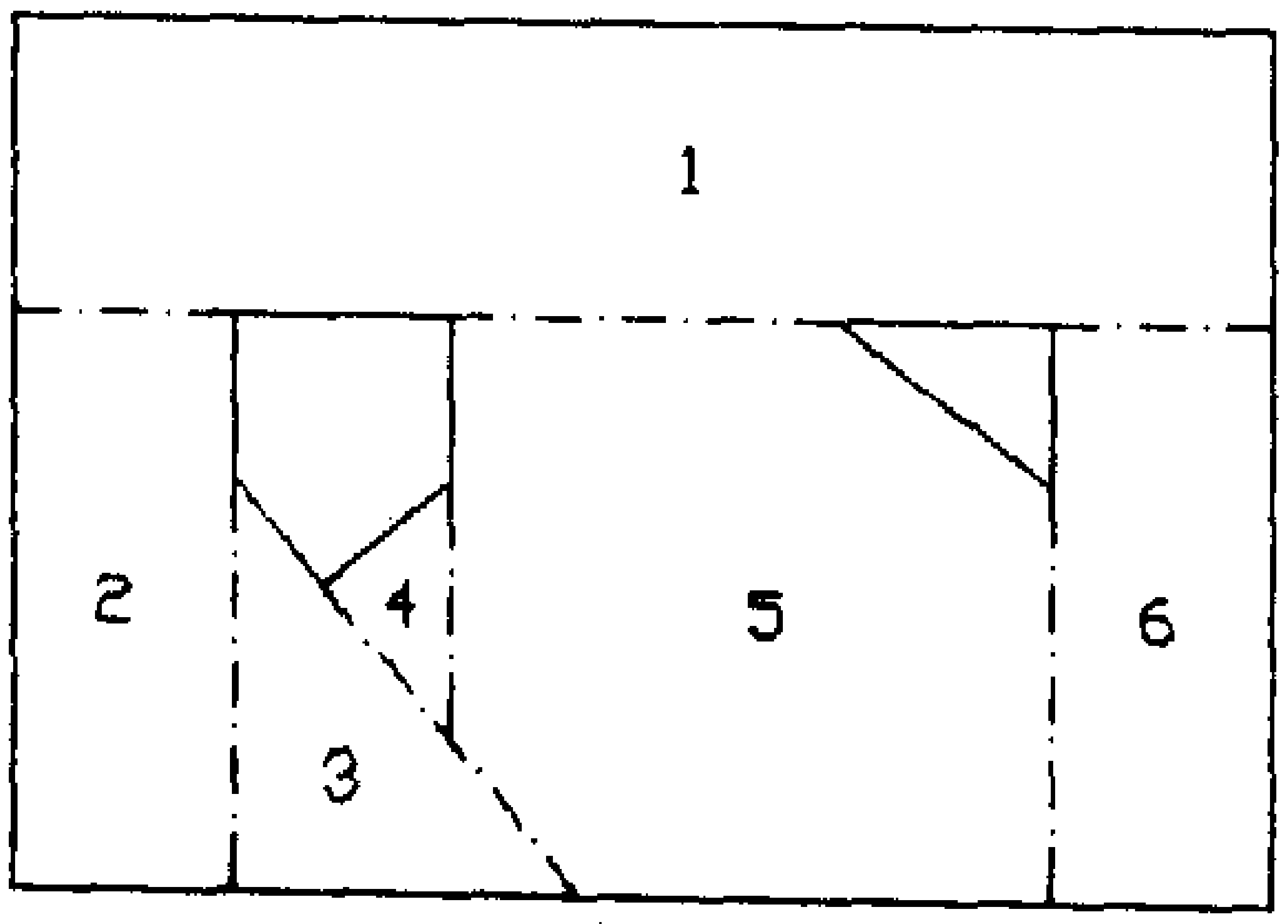

Fig 22

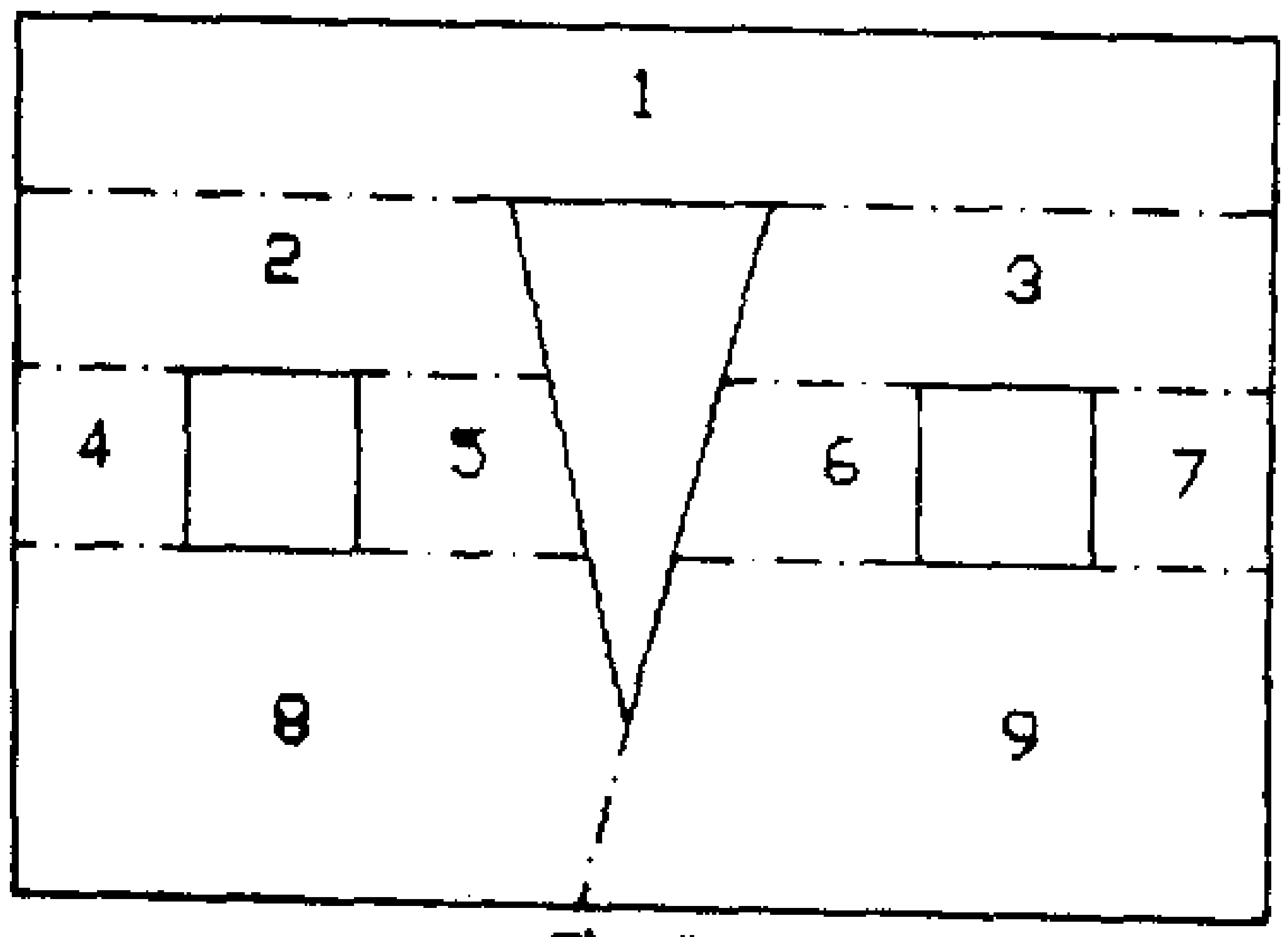

Fig 23

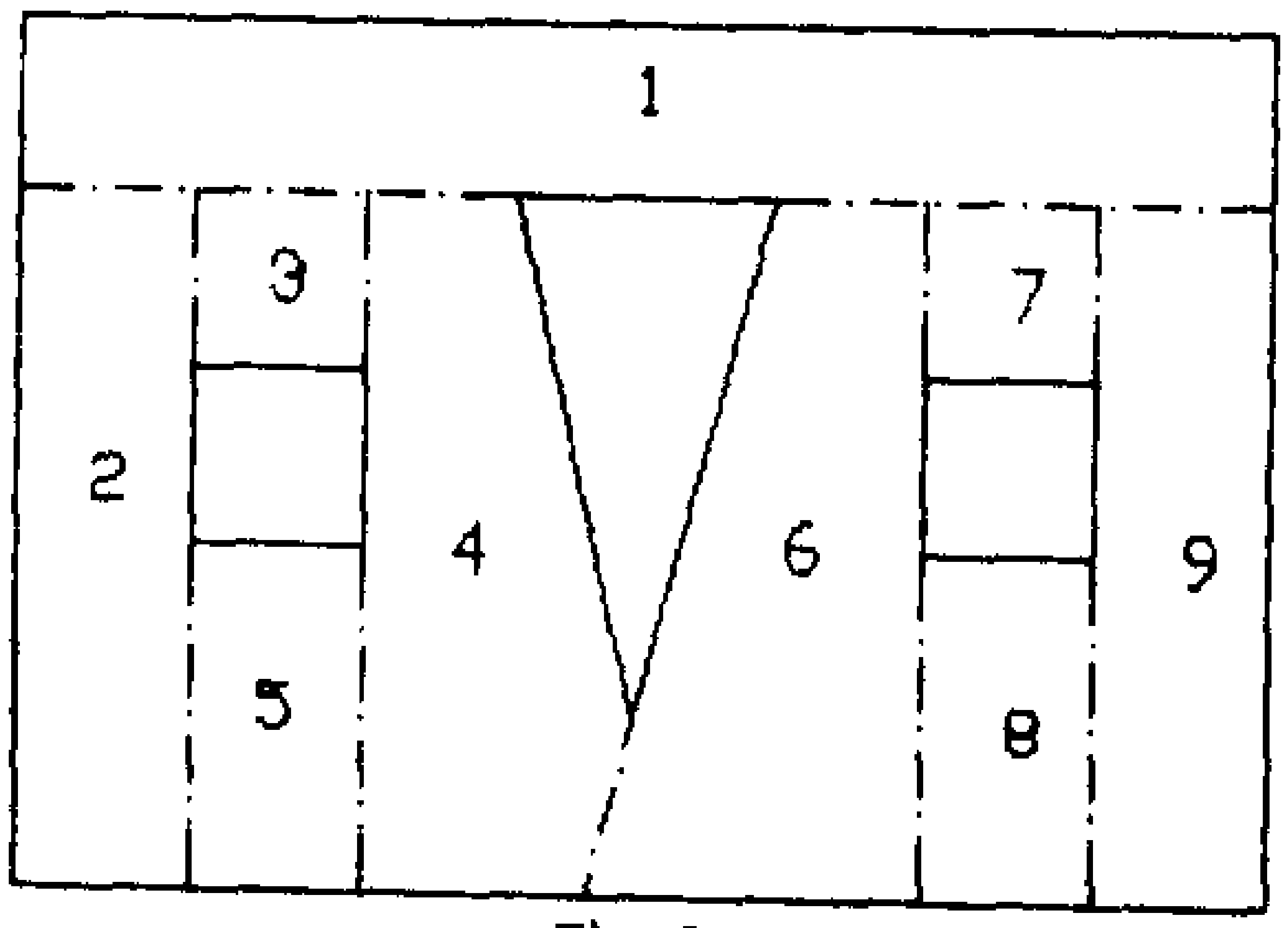

Fig 24

For each vertex there are two ways of extending it. The decomposition need not be unique. There are several ways of decomposition.

Figures 17, 18 and 19 exhibit three different decompositions of the same figure.

Note: Number of CONVEX POLYGONS ARE THE SAME.

The number of decomposed figures is constant. But in case of collinear points to maintain minimality of decomposition it is extended along the direction of collinearity as shown in Figures 20, 21 and 22.

Note: Point B can be extended along BC or AB. Also F can be extended along FG or FH. But as AB and FG are of the same slope it is necessary that B and F be extended along AB snd FG respectively.

Figure - 21 It could be seen that 1 and 6 can be merged.

Figure - 22 Exhibits the minimal decomposition.

Note: This choice matters in case of a pair of points is collinear and adjacent.

In case of non adjacency the direction of extension is immaterial.

Figures 23 and 24 show in case of non adjacency any ray may be chosen to decompose the figure.

No. of convex polygons =

{ (Total No. of Convex points)

- (No. of distinct pairs of collinear and adjacent points)

- (No. of polygons - 1) }

In Figure 25 :

The pair of collinear and adjacent points are :

(B,E) ; (C,H) ; (F,I) ; (G,L)

as the lines AB and EF, CD and HG, EF and IJ, HG and LK respectively.

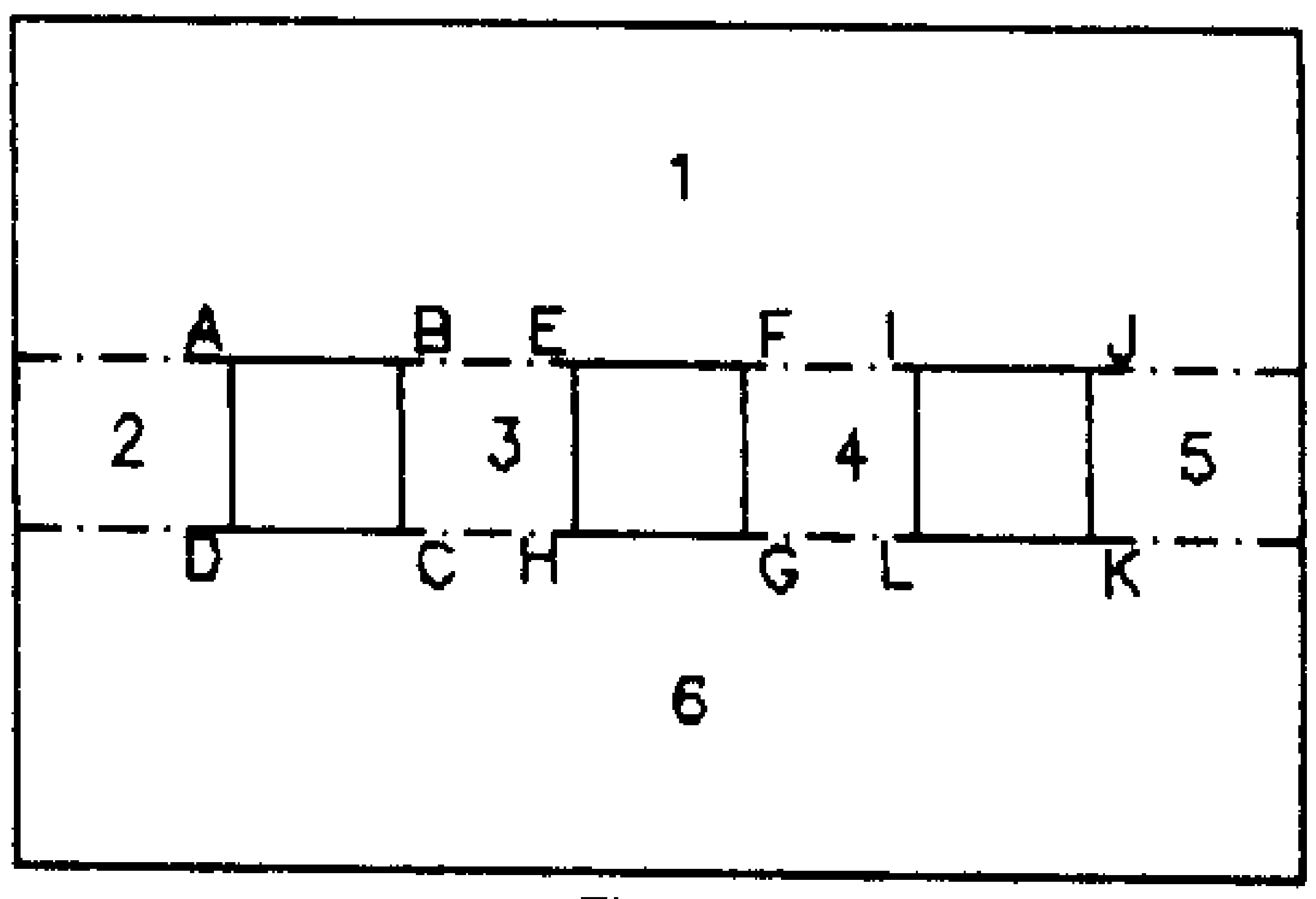

Fig 25

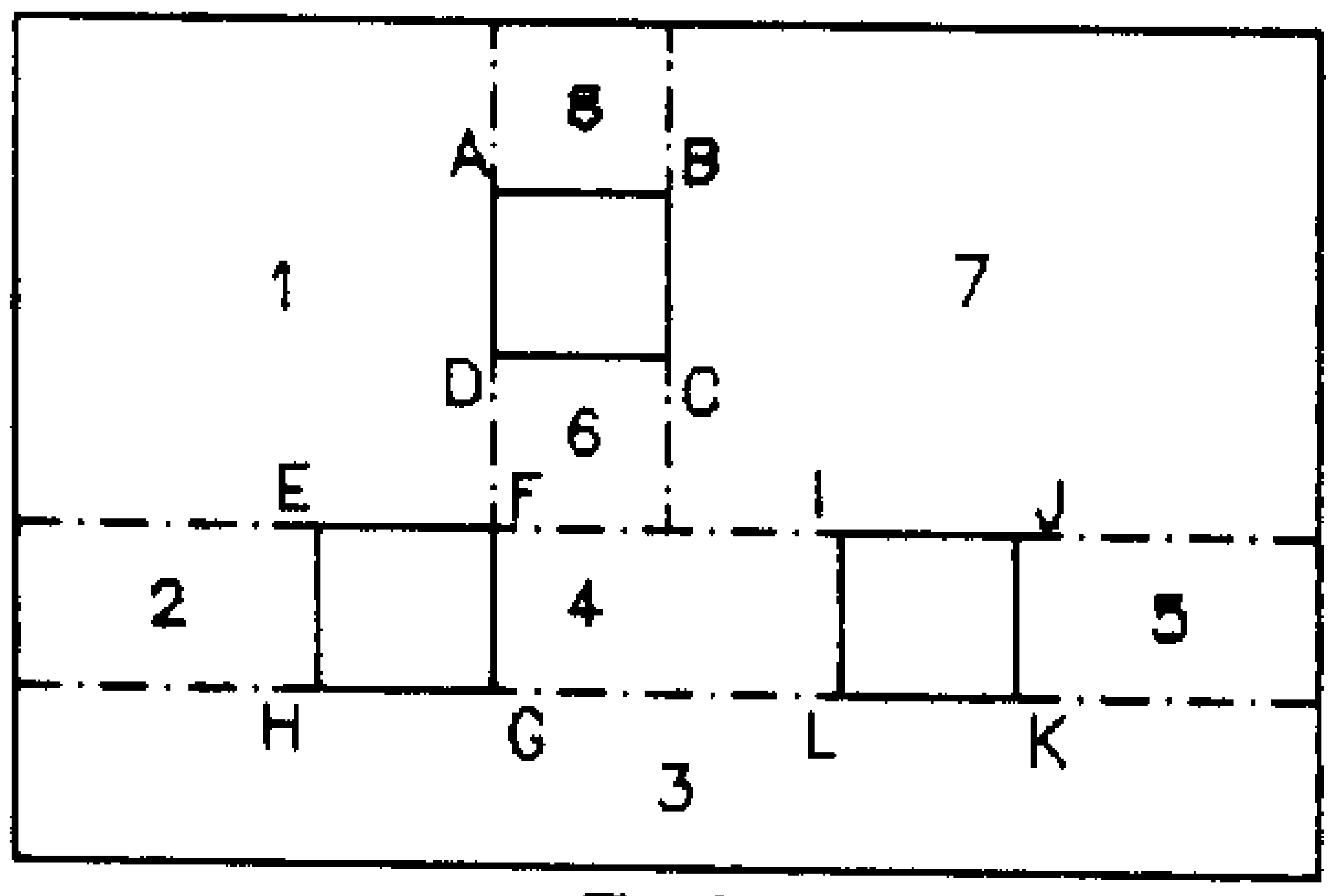

Fig 26

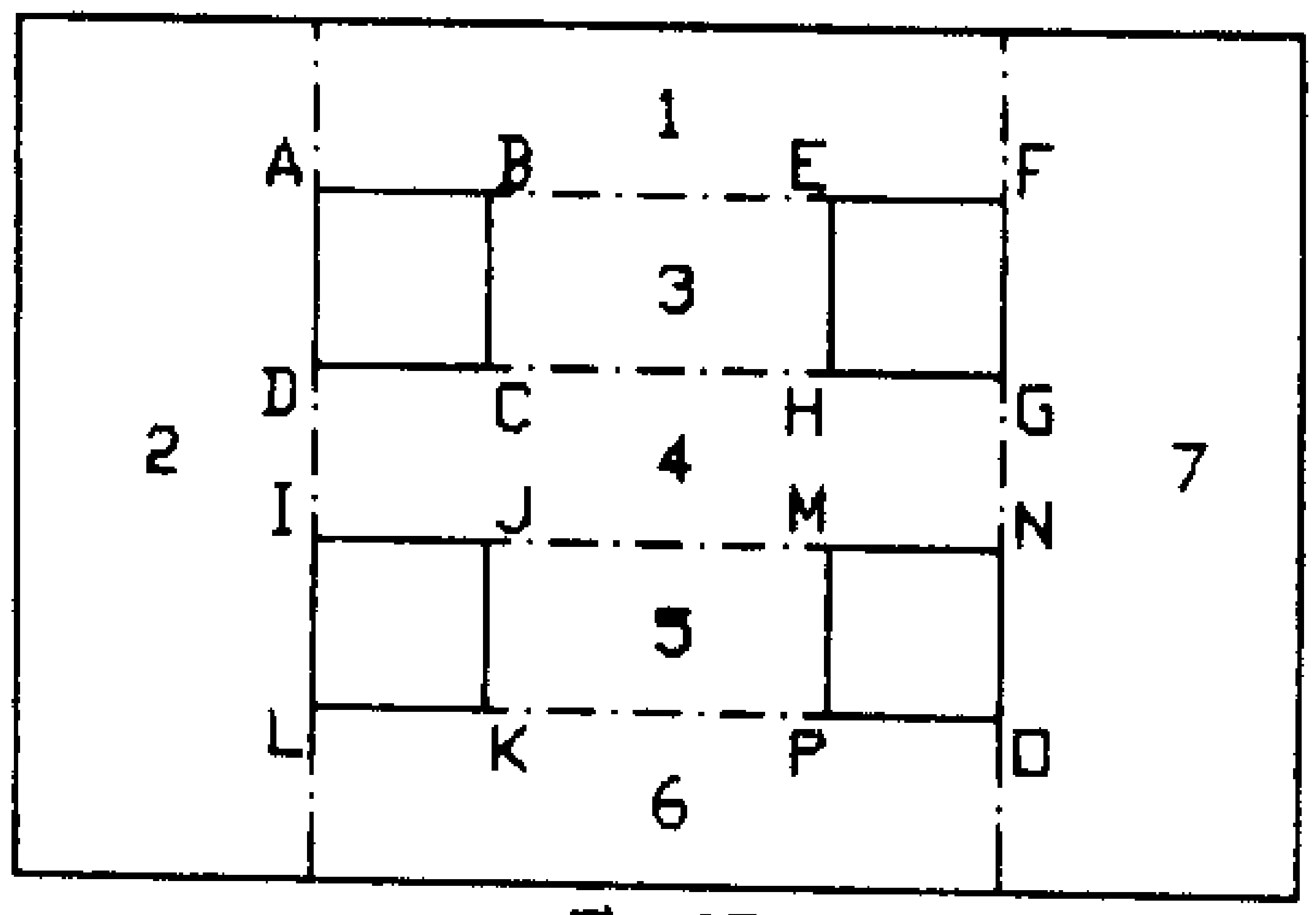

Fig 27

So number of convex points = (12 - 4 - 2) = 6 which is the case.

In Figure 26 :

(D,F) ; (F,I) ; (G,L) are collinear and adjacent points, but the count is only two as 'F' appears in both.

So number of convex polygons = (12 - 2 - 2) = 8.

In Figure 27 :

(C,H) ; (B,E) ; (D,I) ; (G,N) ; (J,M) ; (K,P) ; (C,J) ; (H,M)

The first six pairs are collinear and adjacent but the last two are repeated.

Number of collinear and adjacent distinct pairs = 6.

So number of convex polygons = (16 - 6 - 3) = 7.

6.2 Trapezoidal Decomposition :

We have already mentioned the data structure is involved to draw these, for implementation purpose trapezoidal decomposition will be dealt with in which case the decomposition is horizontally maximal.

To draw these lines an AVL tree has to be maintained which consists of polygonal edges. Each node has segments to its left as its left subtree and segments to its right as its right subtree.

We perform a Top Down sweep halting at every polygonal point.

At every point we run the following algorithm :

- Find the points position in the AVL tree.
- If the point is the topmost point of the polygon then insert both the segments incident on it in the AVL tree.
- If the point is the bottommost point of the polygon then delete both the segments incident on it from the AVL tree.

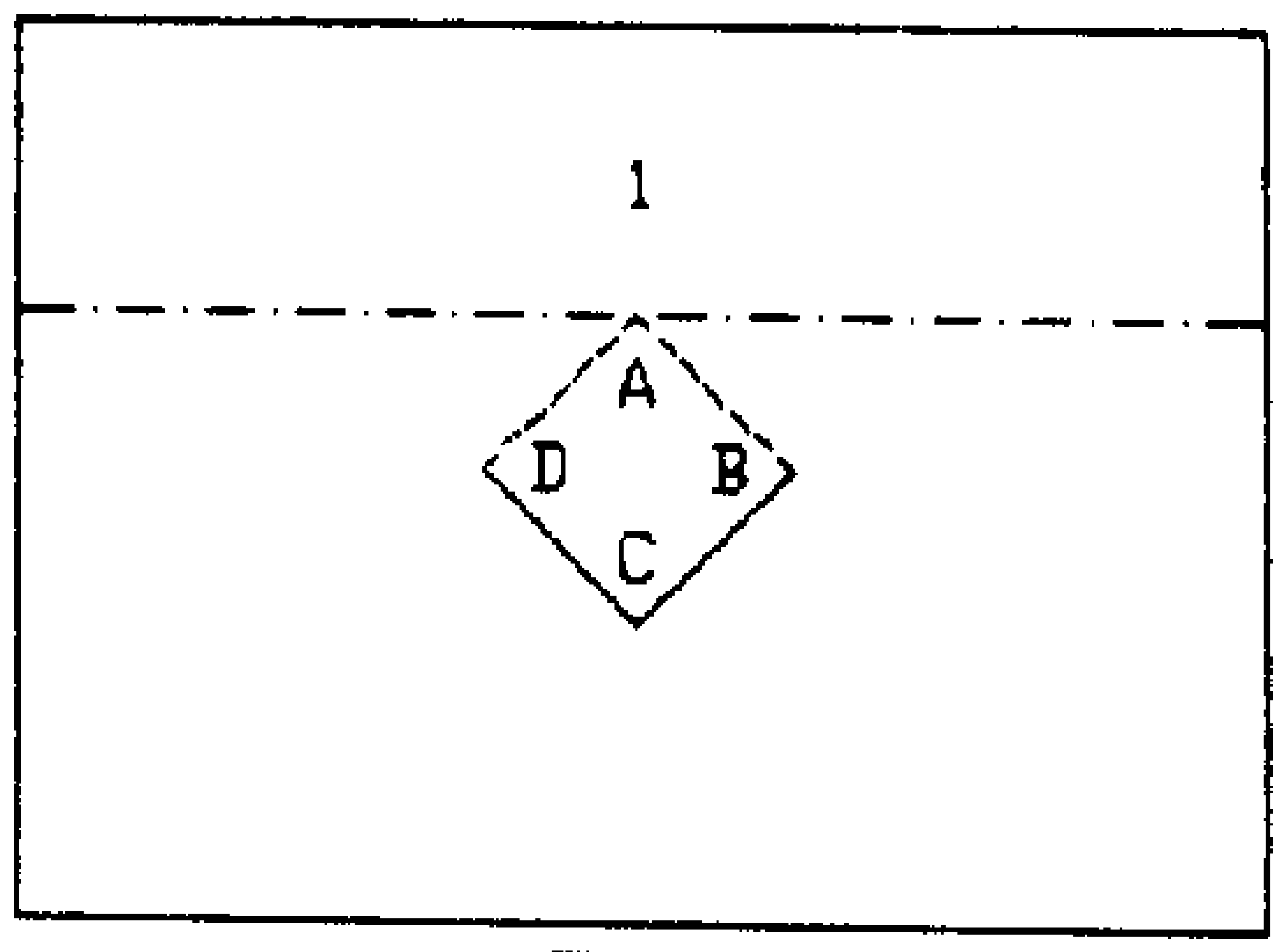

Fig 28

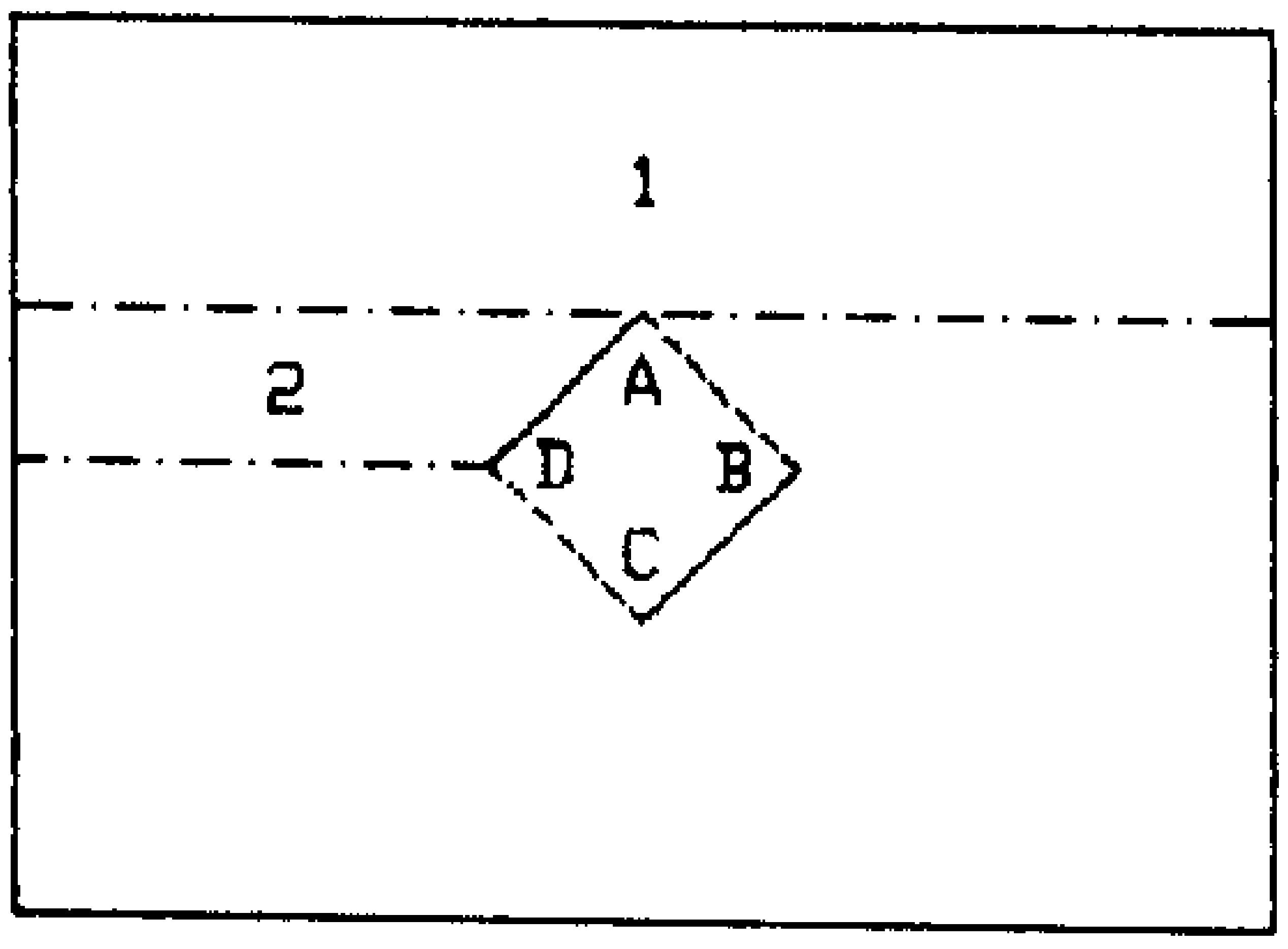

Fig 29

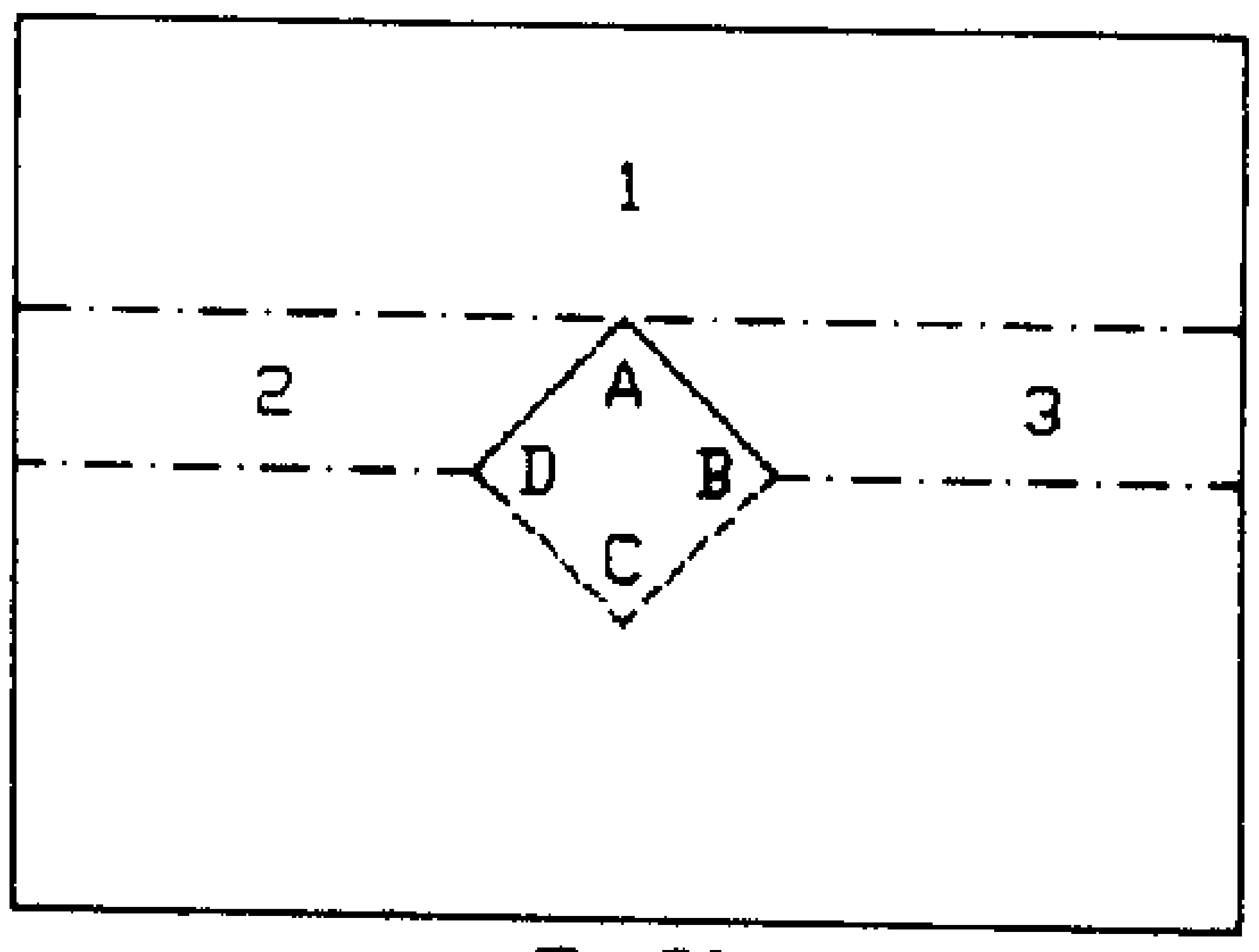

Fig 30

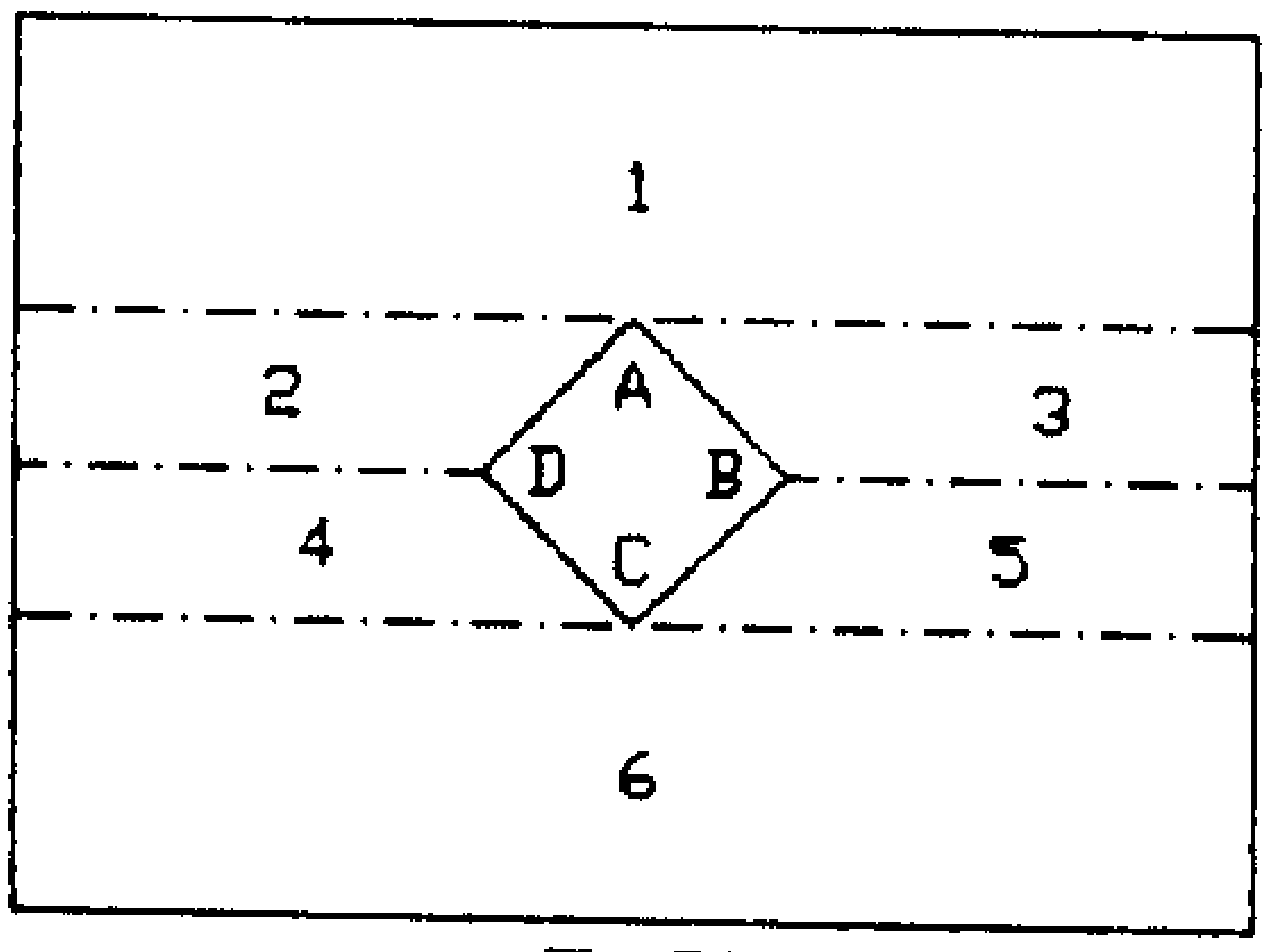

Fig 31

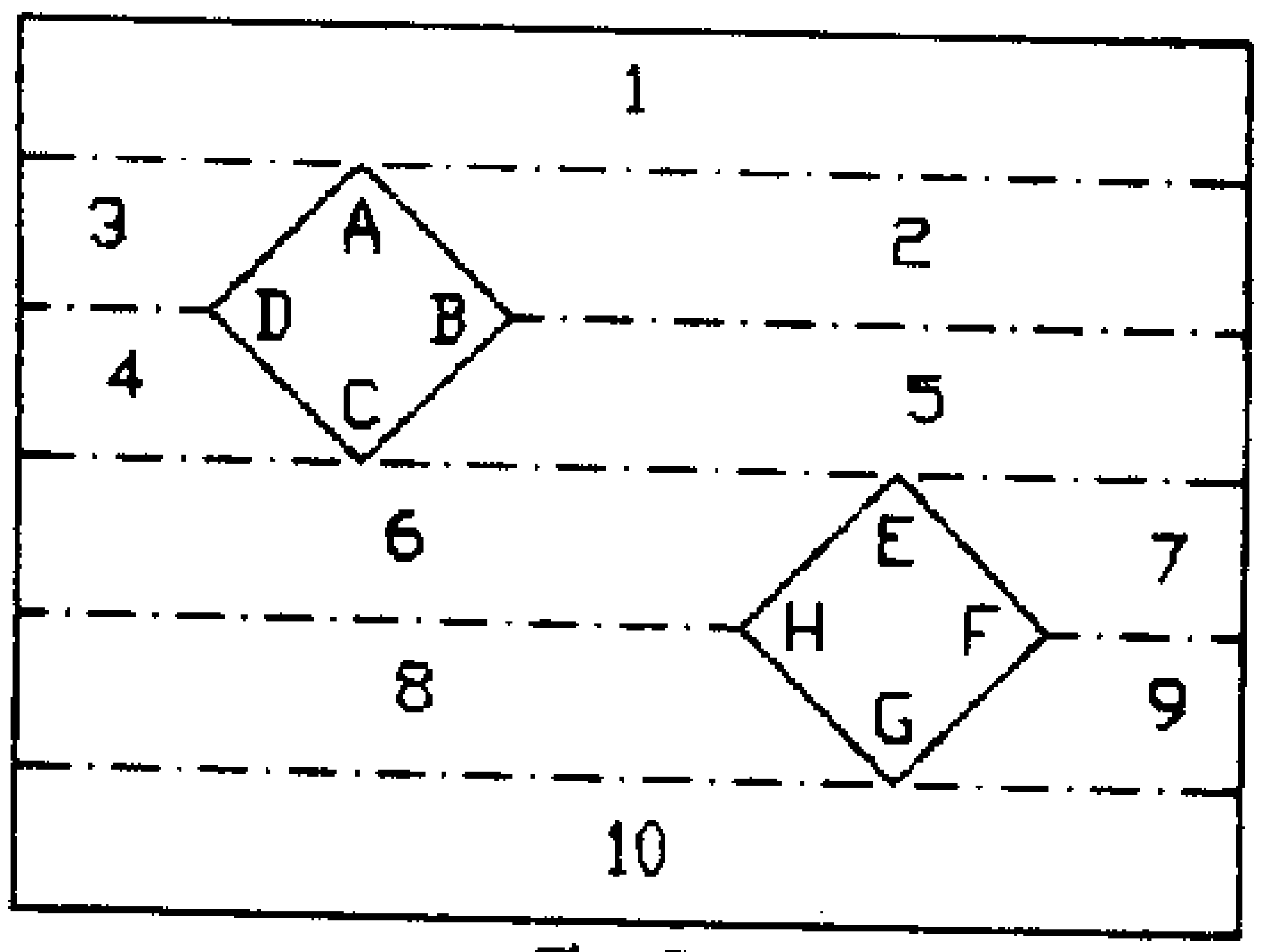

Fig 32

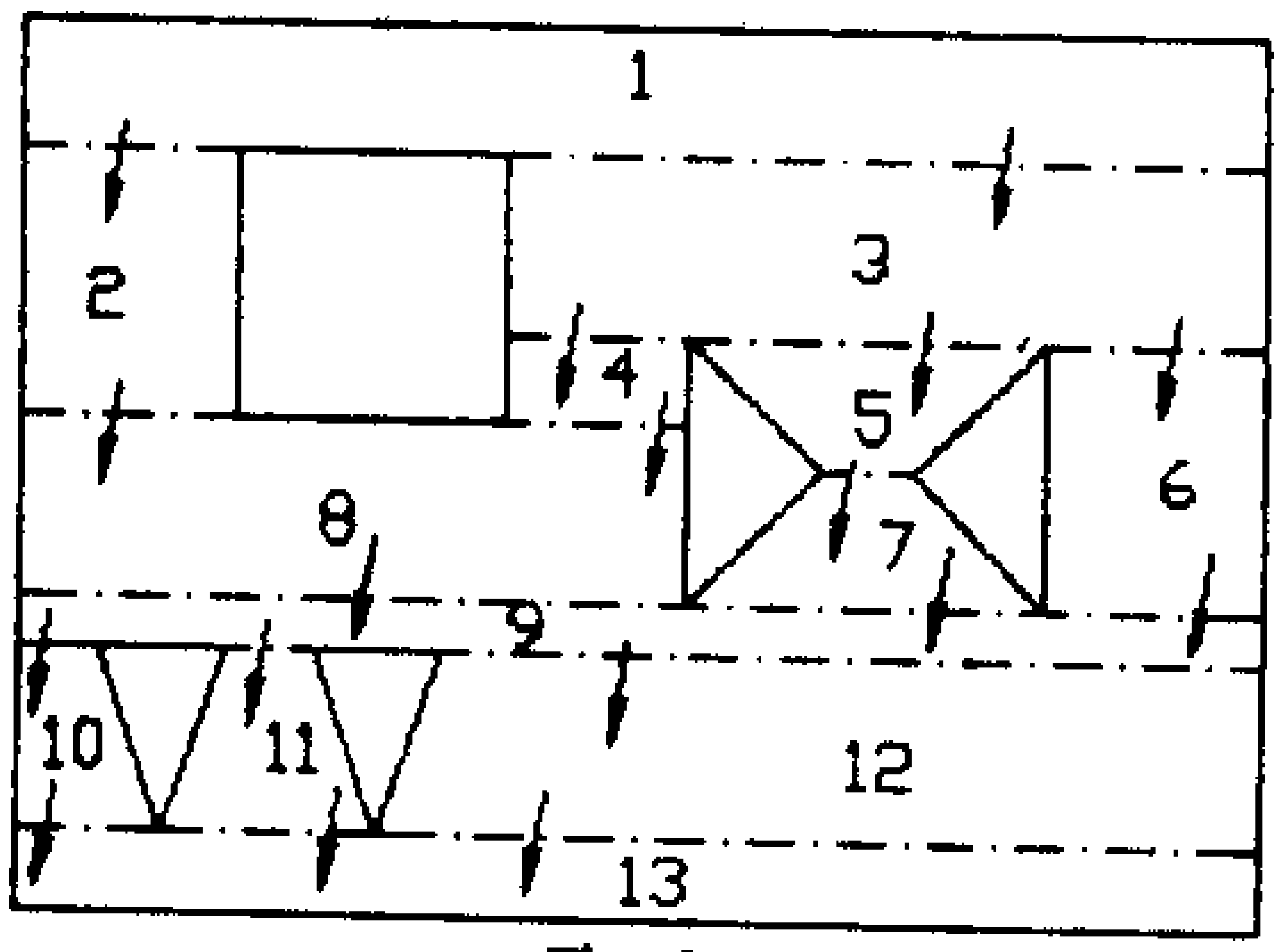

Fig 33

• If its some other point delete the upper segment incident on it from the tree and insert the lower segment incident on it in the tree.

Figure 28 - A inserted in AVL tree. Figure 29 - D inserted in AVL tree. Figure 30 - B inserted in AVL tree. Figure 31 - C inserted in AVL tree.

Algorithm - 4 :

```
Step 1° : Sort the vertices w.r.t. y coordinate.
Step 2° : Insert the two vertical boundaries in the AVL tree.
i.e. T = (PS ∪ QR)
Note : segments are in left to right order in T
Step 3° : Process the points in sorted order.
Process($P_i$)
1. Locate the position of $P_i$ in T.
2. If (∃ no neighbour $P_j$ of $P_i$)
Delete all $(P_k, P_i)$ in T, where $P_k$,
$P_i$ are vertices of the same polygon P.
Else while(∃ an unprocessed neighbour $P_j, P_i$) {
-) if (($P_k, P_i$) ∈ T for some processed k
where $P_k, P_i$ are vertices of the same polygon P)
T = T \ $(P_k, P_i)$
-) T = T ∪ $(P_i, P_j)$
}
```

Figure 32 and 33 shows the result of this algorithm. Figure 33 shows in cohorizontal cases lesser number of trapeziums result.

Complexity Analysis :

Complexity analysis of convex polygonal decomposition has already been discussed. We now give the analysis of trapezoidal decomposition. Here sorting takes O(nlogn) time, the rest of the operations are AVL tree operations. So the entire process takes O(nlogn).

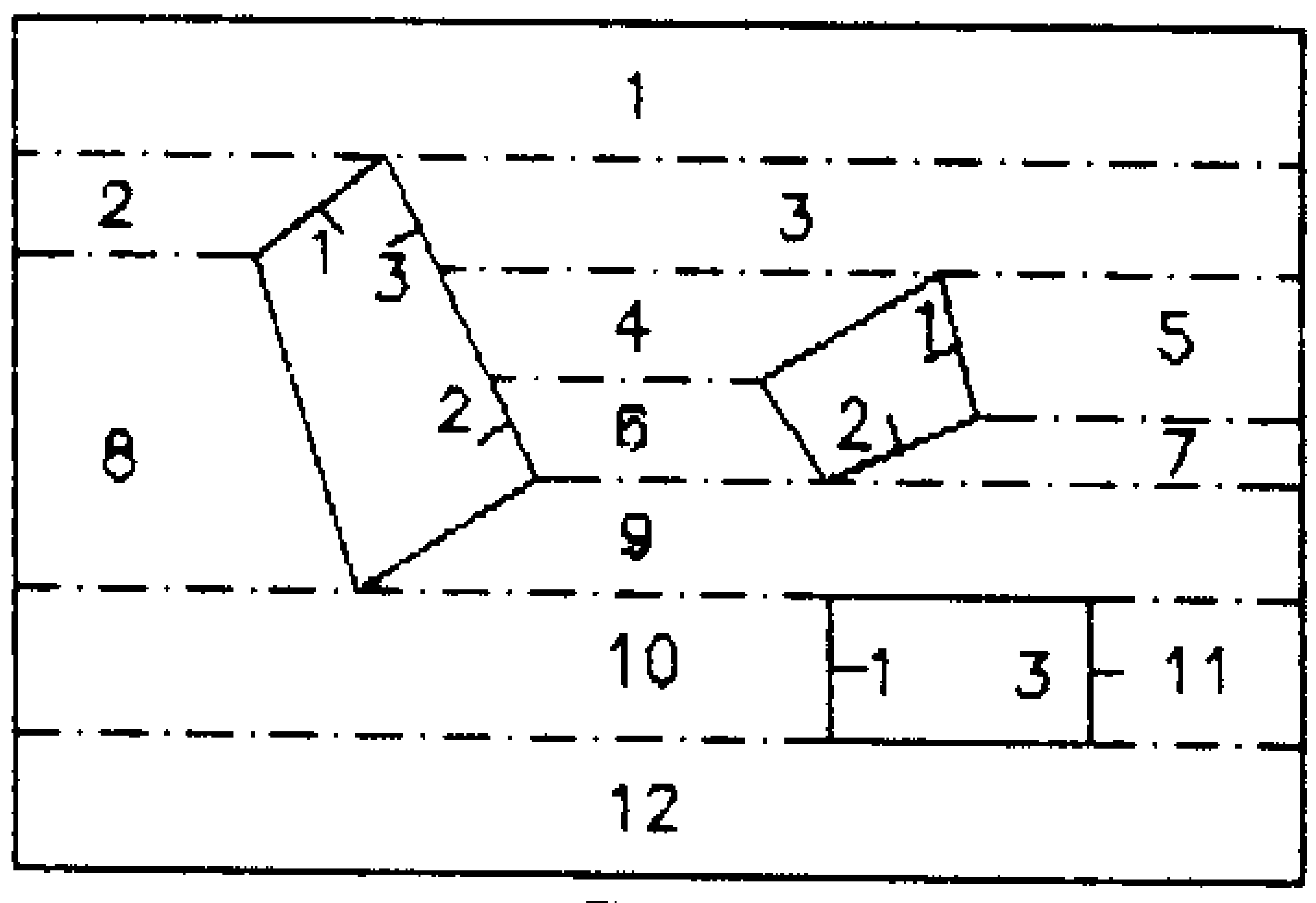

Fig 34

Chapter 7

Routing

We shall route all the nets concurrently using the line sweep algorithm i.e. taking all the nets together, we need not consider the ordering of the nets.

THEME :

Consider the graph 'G' with the trapeziums as its vertices. An edge is present between two trapeziums if they share a common edge or a part of it. Needless to say the graph will be planar. Figure - (34) gives one such example.

Initially all the nets appear along the edges of the convex polygons, and they will be attached to their adjacent empty trapezium, now we proceed to work with the trapeziums only and may forget the polygonal obstacles.

Our main aim is to route the nets. First we present an algorithm which uses a naive method of routing, without optimising the path lengths, we then progress to suggest certain methods improving upon the path lengths. For implementation purposes we have adopted the first algorithm. We present the examples later.

7.1 Main Algorithm :

The idea will be to do a sort of breadth first search of the graph 'G'. Each node has its own original net list as already mentioned, besides that it inherits the unrouted nets from its ancestors. In each search a set of trapeziums' net list is compared, when a common net is found it is routed through its common ancestors.

This is carried out in two passes, a top-down sweep and a bottom-up sweep. In the top-down sweep the nets are pairwise routed and in the bottom-up sweep interconnections are made in case of multi- terminal nets. Details and Examples are discussed a little later. Whatever is left unrouted in the two passes (say due to capacity failure) is done over repeated iterations.

Before proceeding further we explain the concept of inheritance.

In case a net is left unrouted its neighbour inherits it i.e. the neighbour's new netlist will be the set theoritic union of its original netlist and the unrouted nets of its ancestor's netlist.

When a trapezium has more than one neighbour then the neighbour through which the median of unrouted concerned net passes inherits that net.

Each time before inheriting the capacity of the edge is checked, in case the capacity has been used up it is left for the next iteration.

Also for each net there are pointers pointing to the topmost and bottommost net. The bottommost (topmost) net is not inherited by the trapeziums below (above) it.

Steps involved in our algorithm may be summarised as :

- 1. Find the original net list.

• 2. Form the queues [@] .

• 3. Find the topmost and bottommost pin for each net.

The next few steps are done iteratively.

• 4. Perform a Top to Down sweep. (Here pairwise routing is completed).

• 5. Perform a Bottom to Up sweep. (Here connections for multiterminal nets are completed).

Whenever capacity of an edge is exhausted find some other path is found by backtracking till we reach a forking (a trapezium with more than one neighbour) vertex in the next iteration, this time we let some other vertex inherit the net.

[@] : Queues are sequences of trapeziums whose net lists are to be compared.

In a certain sense trapeziums in a queue are at a same horizontal level.

The trapeziums formed have a neat ordering that helps in the formation of the queues. Given a queue the next queue would be found by finding the minimum element in the given queue and finding its neighbours and including them in the new queue along with those trapeziums of the previous queue not having their neighbour in the new queue.

Algorithm - 6 :

```
Input : Given a queue[k]
Output : A new queue[k+1]
Process : queue[k+1] = $\emptyset$
minimum = minimum element of queue[k]
while($\exists$ minimum's neighbour)
queue[k+1] = queue[k+1] $\cup$ minimum's neighbour
$\forall$ elements x in queue[k]
```

```
if any of x's neighbour $\notin$ queue[k+1]
    queue[k+1] = queue[k+1] $\cup$ x
else
    queue[k+1] = queue[k+1] $\cup$ all neighbours of x
```

Detailed explaination of sweeps :

Top to Down Sweep :

Compare the elements in the queue after inheriting nets from the previous queue. If we find a match, i.e. same net in more than one element in the sequence, we route it, by tracing back through the path of inheritance. Once routed its flags are marked and is not inherited by its descendants.

In case of inheritance, if a trapezoid has more than one descendant, the descendant through which the *Median** of UNROUTED nets pass through inherits the net.

In case capacity of a trapezoid is exhausted, the net is not inherited and marked UNROUTED.

No net is inherited by the trapezoids below the bottom-most net. An Example is explained in details in the next section.

Bottom to Up Sweep :

The nets left (or *considered*#) unrouted in the previous sweep are routed here. We find *Median** of ROUTED nets in this path of inheritance, whenever a trapezoid has more than one neighbour (bottom). The neighbour through which this median passes inherits the net. The path of inheritance continues till we reach a trapezium which is already present in routed path of the net.

* :

In case median passes through left (right) of the leftmost (rightmost) neighbour, the leftmost (rightmost) neighbour inherits the net. Example.2 illustrates it.

:

In case of multi terminal nets (say four nets) the sweep routs them pairwise say A, B and C, D have been connected respectively. The connection needed between these pairs is unconsidered unrouted.

Figure 40 and Example 3 Illustrates it.

Iterations :

Those left unrouted have flags marked indicating the sweep which left them unrouted, accordingly we trace back to the last ancestor which has more than two neighbours, the other neighbour will be inheriting it this time.

7.2 Illustrative examples :

We achieved hundred percent routing has been achieved on most of the input data.

In fact majority of the cases the routing was completed after one iteration itself.

Example : 1 (Refer Figure - 34)

Its graph : (Refer Figure - 35)

Its original net list : (Refer Figure - 36)

Its queue :

1.	2	3	
2.	8	3	
3.	8	4	5
4.	8	6	5
5.	8	6	7

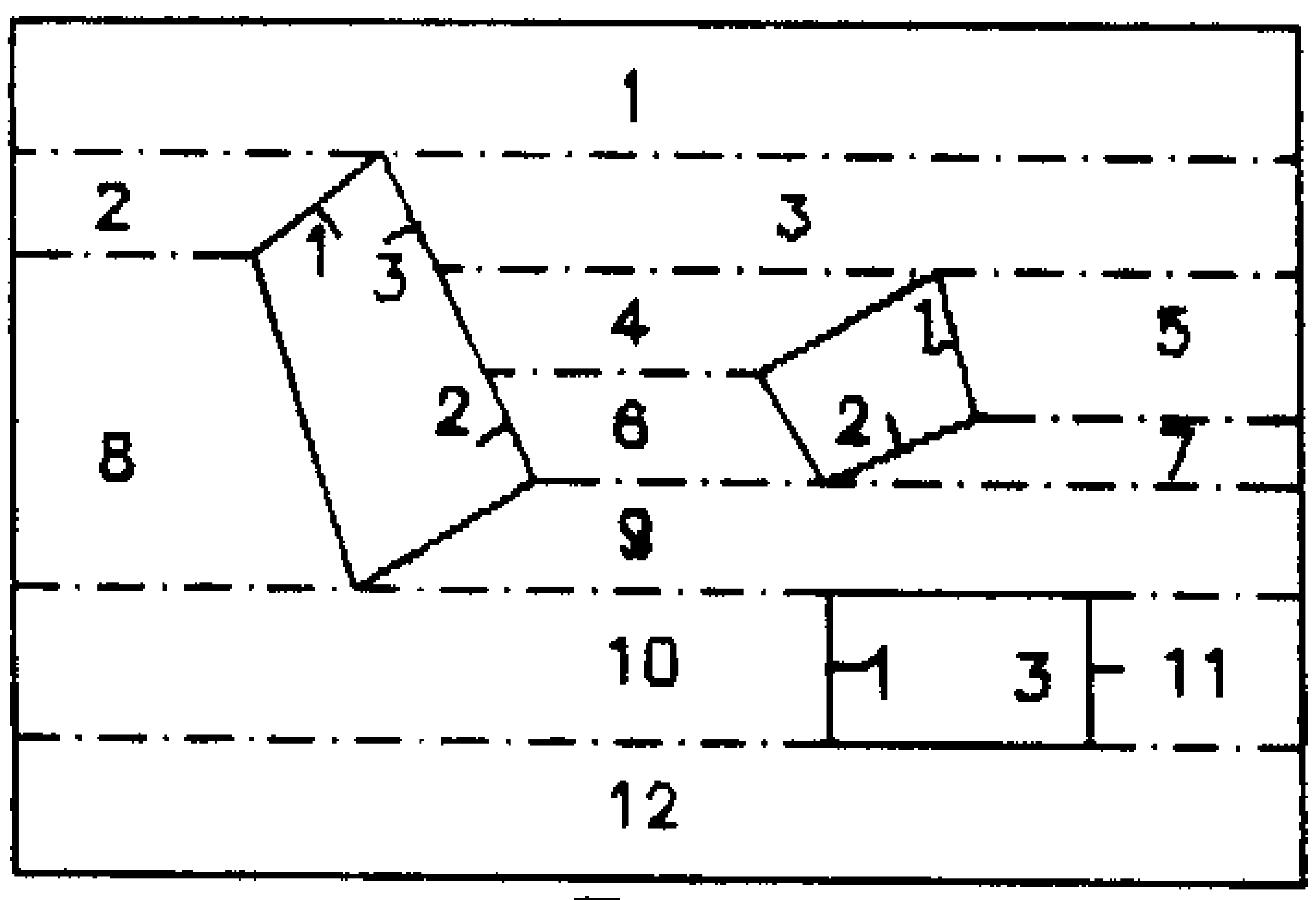

Fig 34

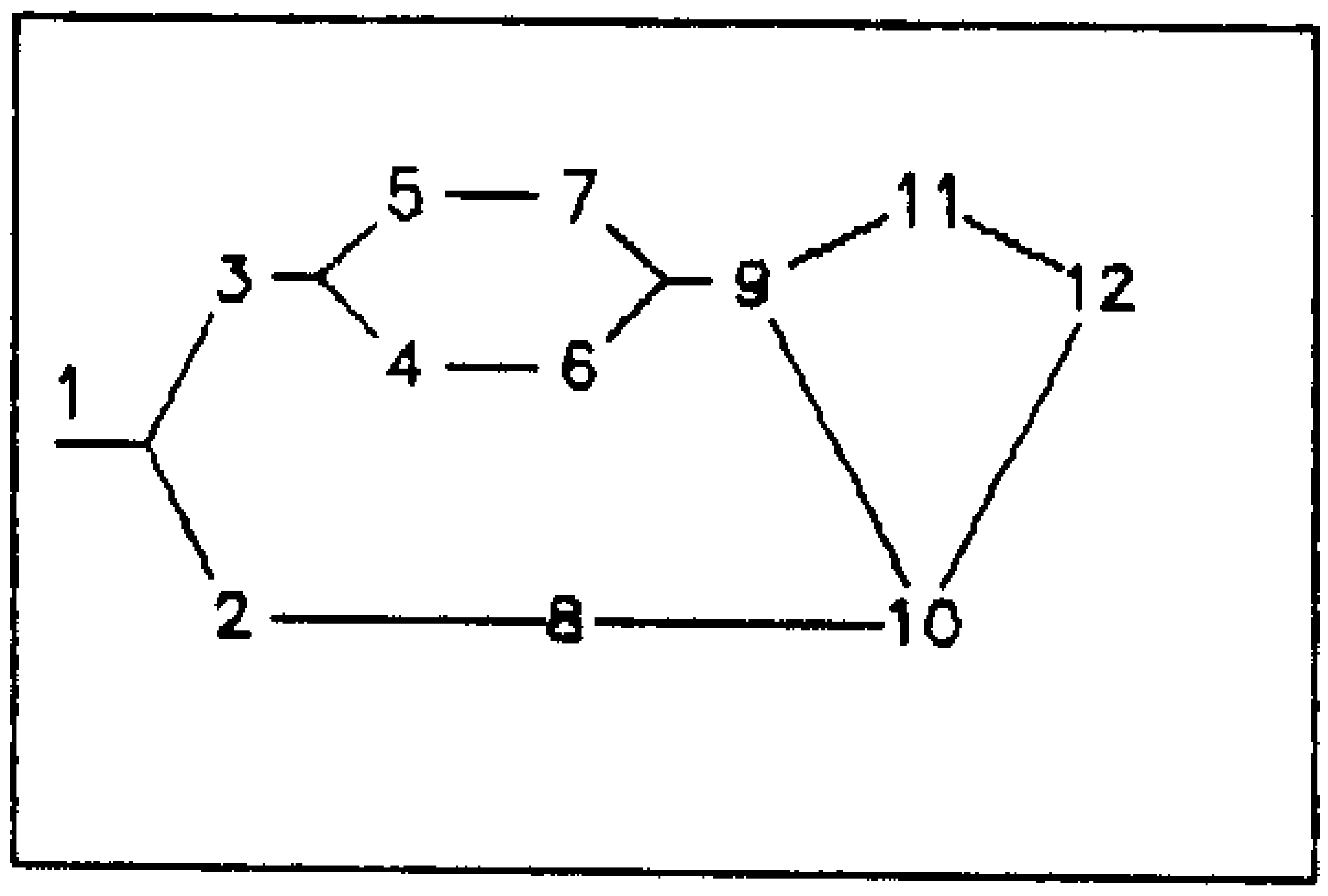

Fig 35

```
1
2 —  3
8 —  3
8 —  4  — 5
8 —  6  — 5
8 —  6  — 7
8 —  9
10— 11
12
```

Fig 36

6.	8	9
7.	10	11
8.	12	

Its course of completion :

<u>*Top to Down*</u>

Queue 1 : $2 \rightarrow 1$; $3 \rightarrow 3$

Queue 2 : $8 \rightarrow 1$; $3 \rightarrow 3$

Queue 3 : $8 \rightarrow 1$; $4 \rightarrow 3$; $5 \rightarrow 1$

- Supposing median of B and G passes through '3' or left of '3'.

As 'one' is common to eight and five their common ancestor is searched for and resulting route is thus

{ 8 - 2 - 1 - 3 - 5 }

'1' being present in 2's original list 8 is redundant and hence deleted.

Path for net # 1 : { 2, 1, 3, 5 }

The flags of 8 and 5 for net 1 are marked routed. To indicate they are not to be considered for inheritance in the next step, shown by bullet superscript henceforth.

Queue 4 : $8 \rightarrow 1^\bullet$; $6 \rightarrow 2,3$; $5 \rightarrow 1^\bullet$

Queue 5 : $8 \rightarrow 1^\bullet$; $6 \rightarrow 2,3$; $7 \rightarrow 2$

- Net 2 appears more than once. The common ancestor of 6 and 7 is 3.

Path for net # 2 : { 6, 4, 3, 5, 7 }

Queue 6 : $8 \rightarrow 1^\bullet$; $9 \rightarrow 3$

Queue 7 : 10 *rightarrow* 1 ; $11 \rightarrow 3, 3$

- Assuming the median of B and G passes through 11.

As 3 is repeated in same trapezoid simply the path of inheritance is traced back.

Path for net # 3 : { 11, 9, 6, 4, 3 }

Queue 8 : 12 $\rightarrow \emptyset$

- 1 is not inherited as 12 is below the bottom most net 1.

Bottom to Up

Unconsidered nets left is F (present at 10). The trapezium through which the median for routed net 1 passes inherits it. Say here the median of A and C passes through 9 and 6. Then 1 is inherited by them and inheritance continues till we reach 3 which is present in path 1.

So the appended path is { 10 - 9 - 6 - 4 - 3 }

The path for net # 1 is finally : { 2, 1, 3, 5, 10, 9, 6, 4 }

Routing is over.

Net # 1 : { 2, 1, 3, 5, 10, 9, 6, 4 }

Net # 2 : { 6, 4, 3, 5, 7 }

Net # 3 : { 3, 4, 6, 9, 11 }

Note : Here routing has been completed in one iteration.

If any net remains unrouted we iterate the steps to route till completion.

Why we need the bottom to up sweep is clear from the discussed examples..

Next we ponder over the reasons for unrouted nets even after these two passes.

Reasons for unrouted nets remaining :

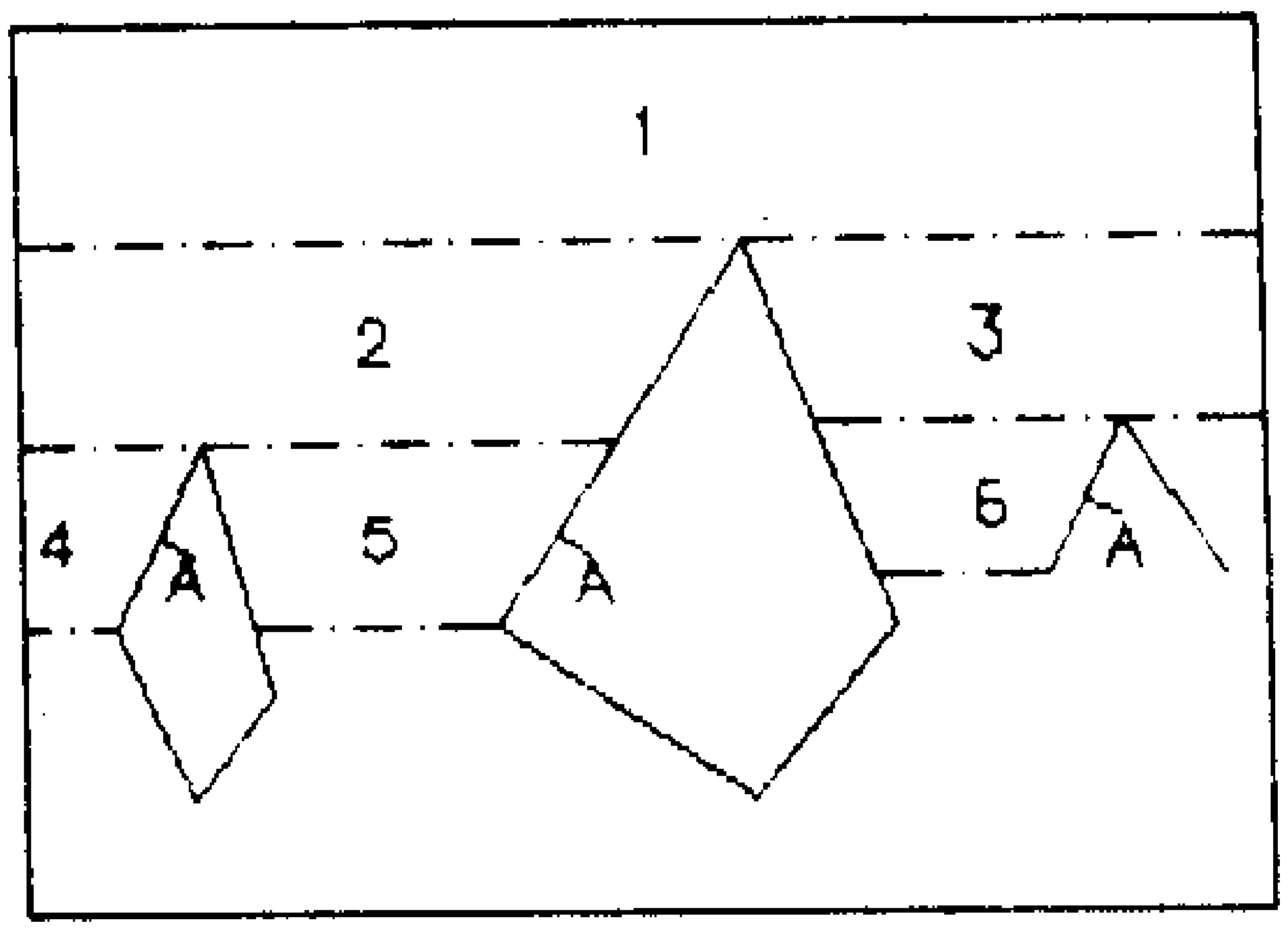

Fig 37

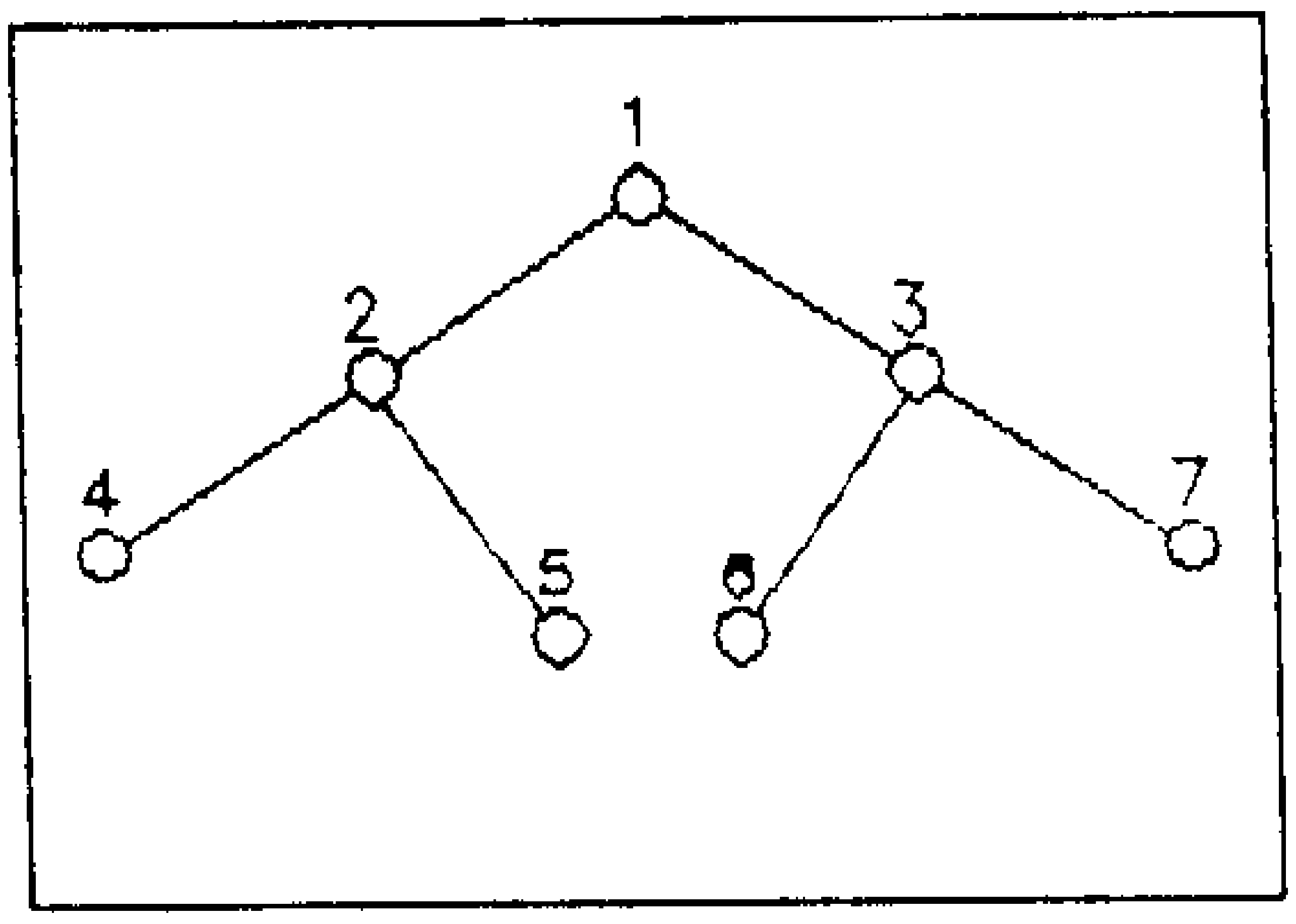

Fig 38

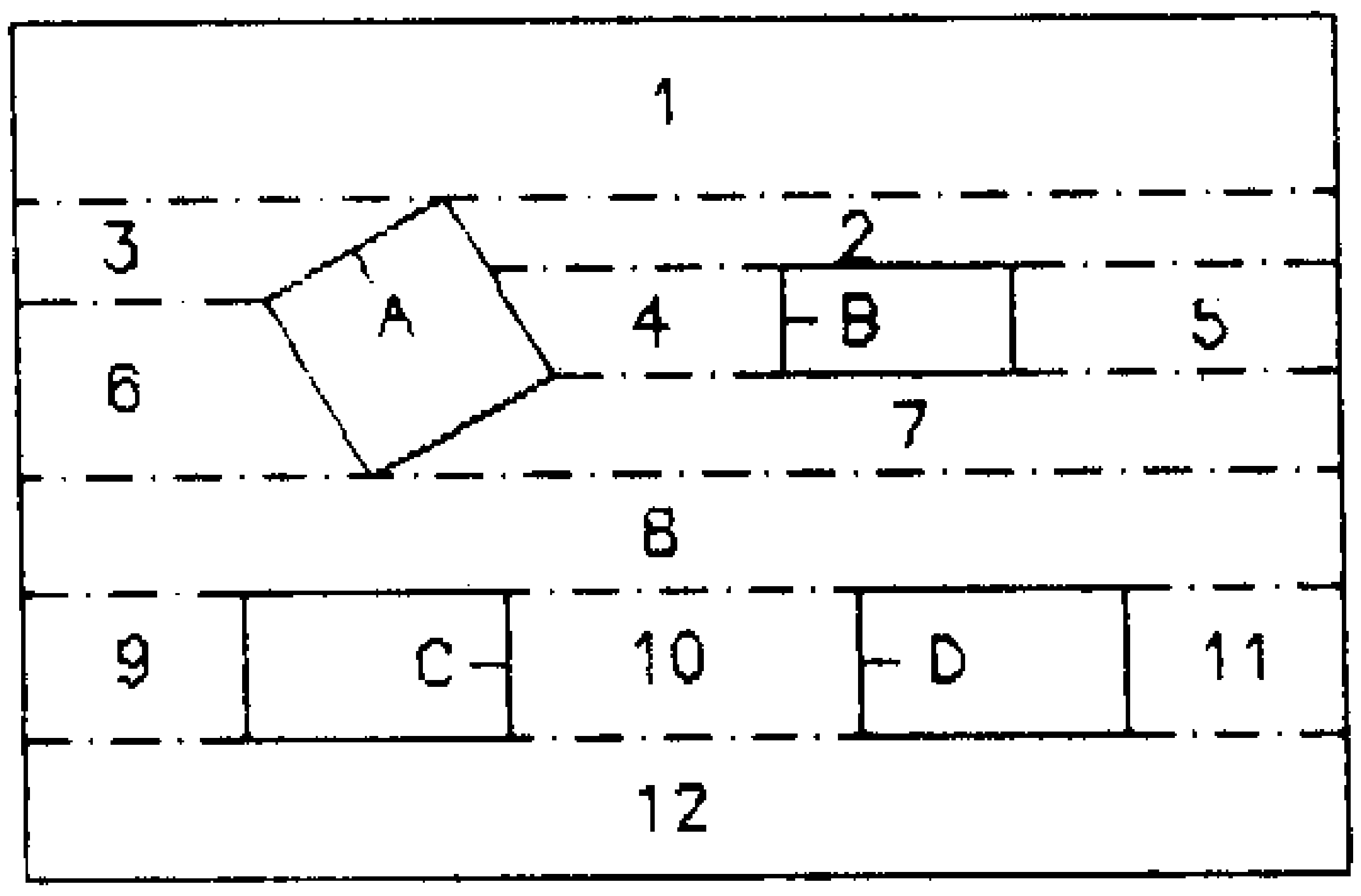

Fig 39

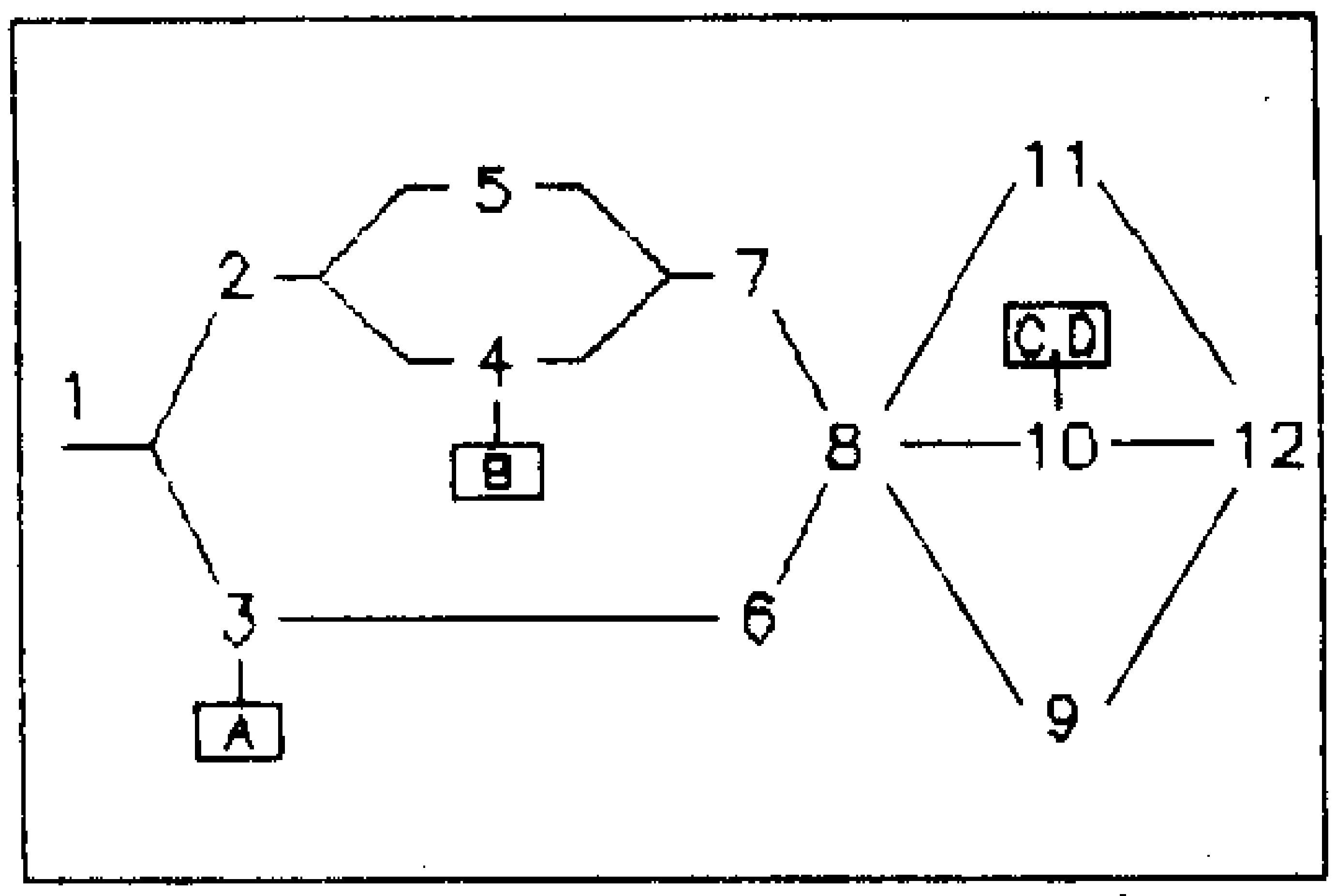

Fig 40

1. Capacity constraint disallowing inheritance (both for top and bottom).

2. While tracing back (by checking the median in the bottom sweep) we may reach the end without any desired result. As shown in Example 2.

Example 2 : (Refer Figures - 37, 38)

When the queue 4, 5, 3 is considered repeated occurance of net 1 is detected.

Resulting Path for net 1 : { 4 2 5 }

Net 1 at 6 remains unrouted in the top to down sweep also it is not inherited by any trapezium as it is the bottommost net 1.

In the first bottom to up sweep the path encoutered is 3 1 none of them being present in the constructed Path 1.

As the end has been encountered without the completion of routing.. this has to be taken care in the next iteration.

Example 3 : (Refer Figures - 39, 40)

As the nets are multi terminal, say four nets A, B, C, D are all net 1. A and B, C and D are pairwise routed in the same order. Then the lower C and D are considered unrouted for bottom to up sweep to have inter connection.

Top To Down :

AB { 3 1 2 4 }

CD { 10 }

Path for net # 1 : Path for AB $\cup$ Path for CD

Bottom To Up :

The lower element among C and D is selected and considered

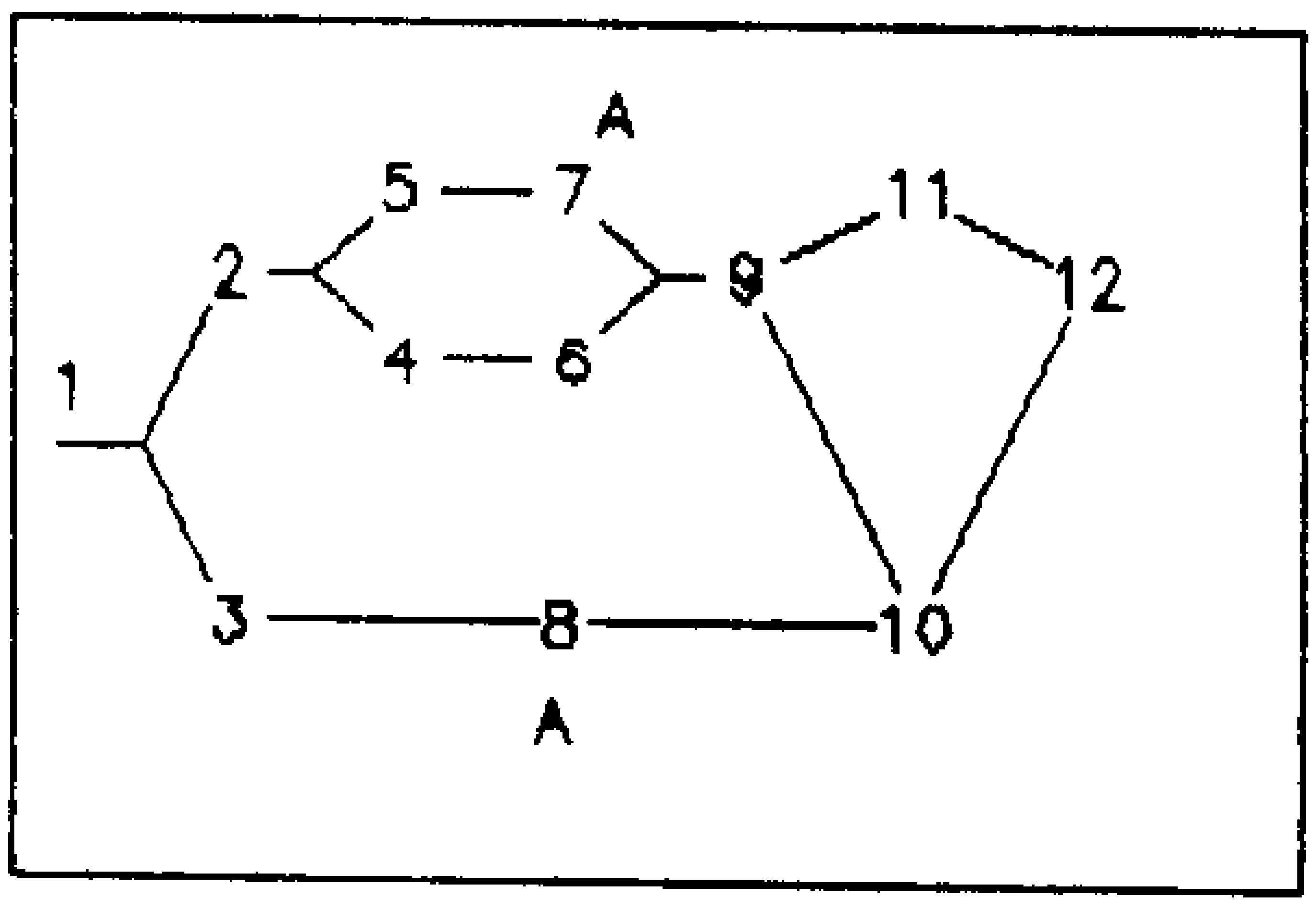

Fig 41

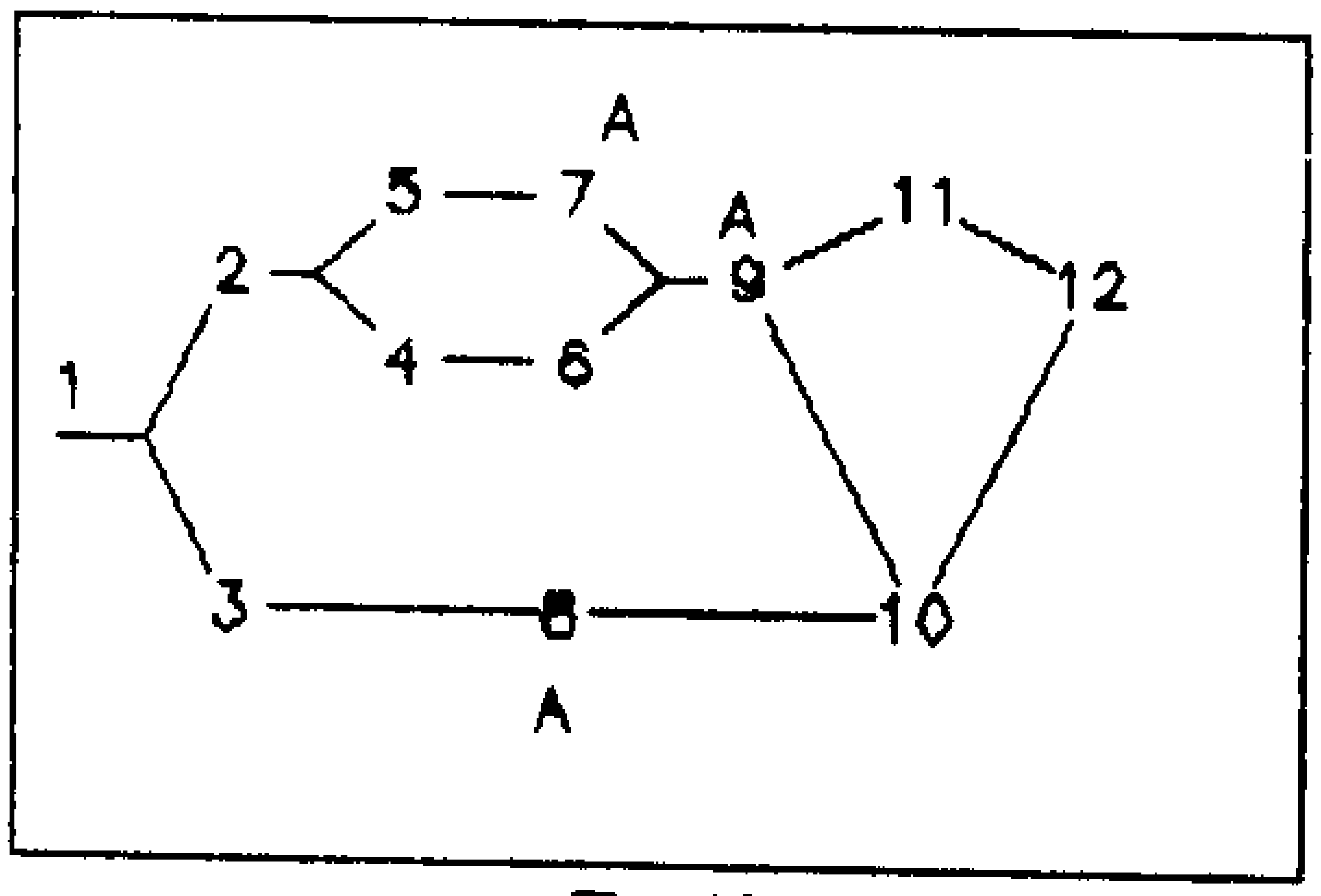

Fig 42

unrouted to trace path. say median of A and B passes through '7'. then path traced is

{ 10, 8, 7, 4 }

Ultimately the path for net # 1 is :

{ 3, 1, 2, 4, 10, 8, 7 }

7.3 Better Techniques

In this method its obvious that we have overlooked any method of optimising the route. Keeping this in mind we may approach to find better (i.e. in terms of wire length) route.

• On finding a matched net instead of always backtracking till its common ancestor, we could calculate the smaller distance (distance from either of it to its common ancestor or descendant) and route it accordingly. This will lead to smaller routes. Example (Figure - 41).

• Other criteria for routing would be to take density of the pins of this net to decide whether to trace the path till the common ancestor or descendant. This could be done by simply checking the y coordinates of the nets. This will ensure the connections from the remaining pins of this net may have a shorter path. Example (Figure - 42).

• In subsequent passes we may allow inheritance even after a match is found to study its performance.

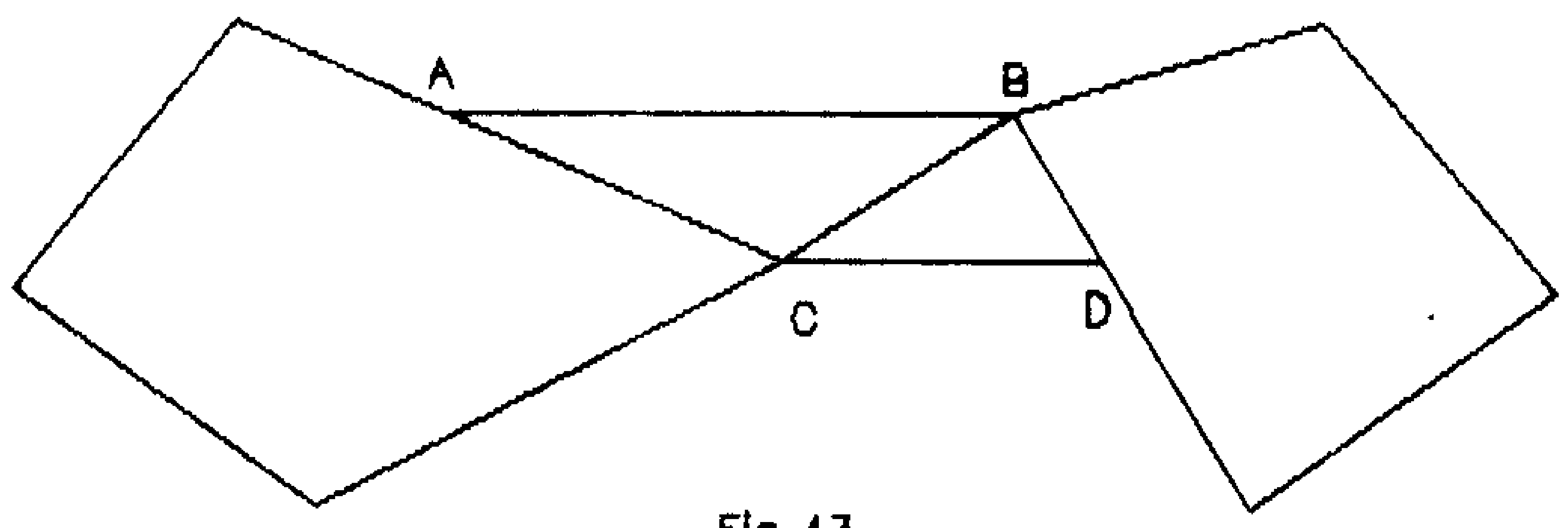

Fig 43

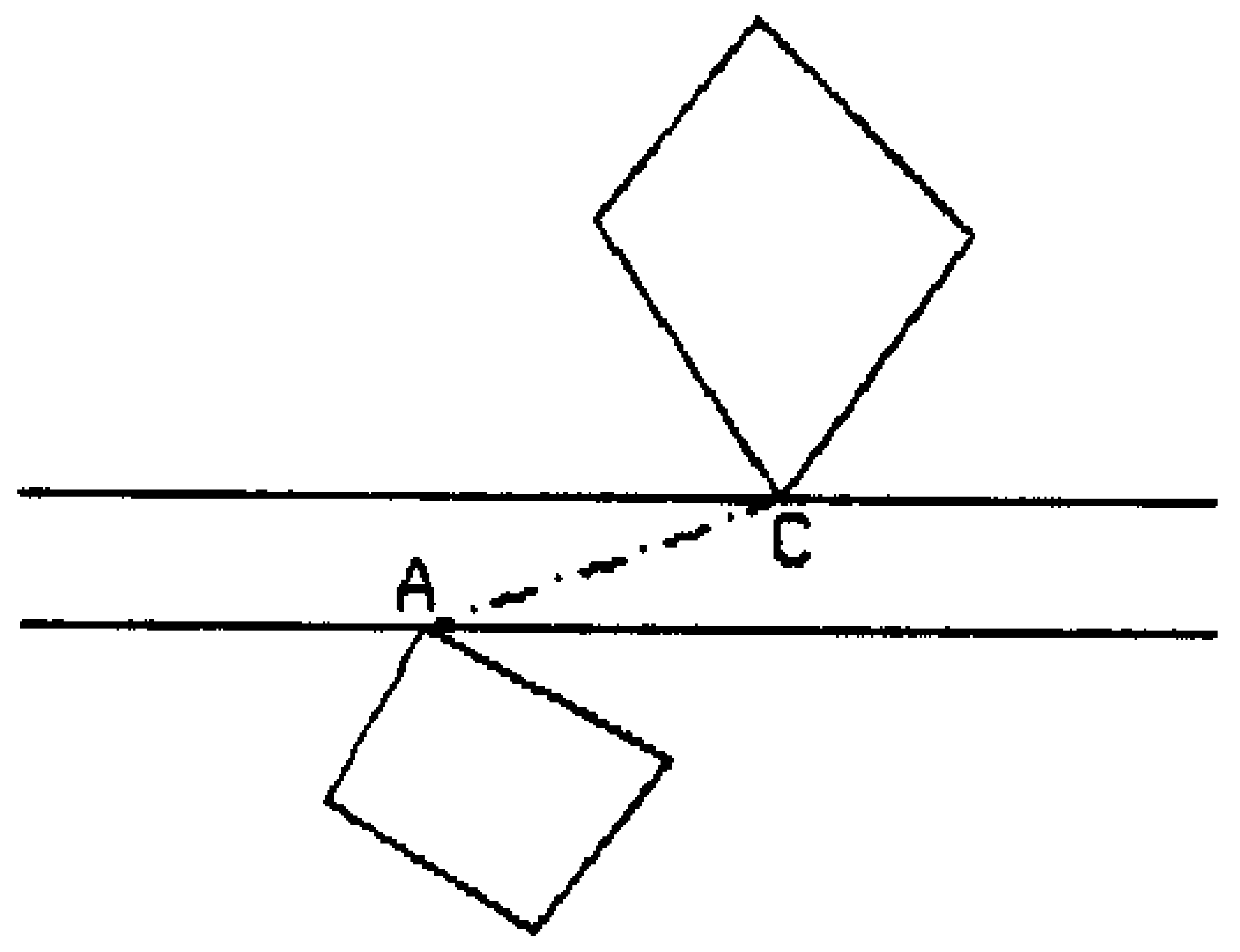

Fig 44

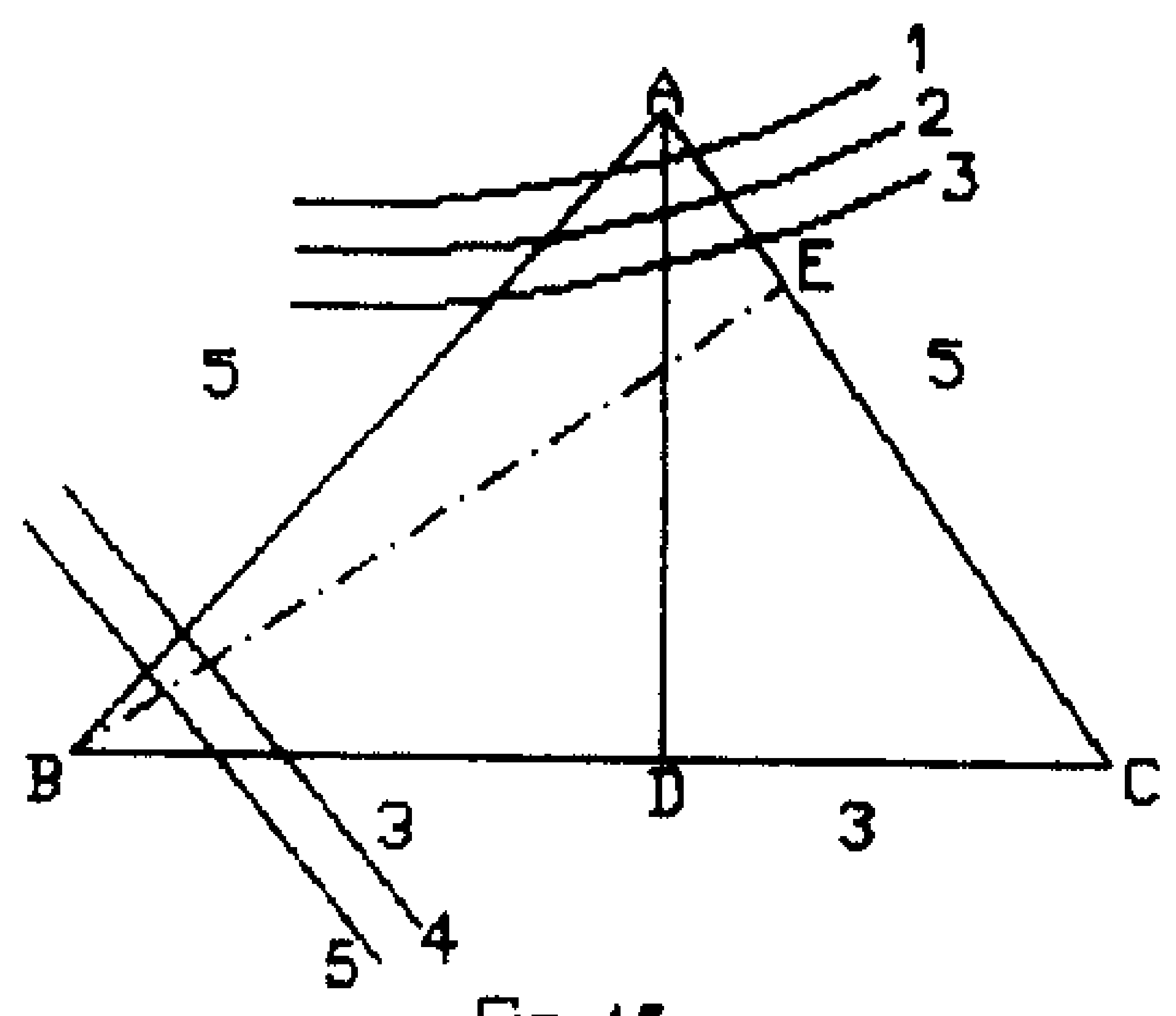

Fig 45

Chapter 8

Conclusion

Our approach involves breadth first search considering all the nets at the same time. The nets placed higher up have a higher priority of being routed before.

This does not guarantee the shortest path, but guarantees an obstacle free path.

As all nets are considered simultaneously without any bias to its ordering.

For most of the input data the routing is over in the first iteration itself because many capacity constraints have been overlooked.

Examples : (To show how the capacity constraint considered here is inadequate).

(Reference Figure - 43)

Definitely AC allows lesser number of wires to pass through than either AB or CD considered.

(Reference Figure - 44)

Constraint AC has been overlooked .

(Reference Figure - 45)

Though AB, AC allow five wires to pass through AD actually restricts it, which has not been considered.

To overcome the above mentioned problem if we consider the altitudes drawn from the intersection point of the sides through which the wires pass are considered.
(i.e. for wires 1, 2, 3 AD is considered
and for wires 4, 5 BE is considered)

it is insufficient. BE or AD can allow more wires to pass through but AB cannot.

So we may have to consider a combination of both of these.

Adding a subroutine ***test-capacity*** which will test for all crossing of wires within the trapezoidal area will make this algorithm complete in true sense.

Bibliography

[1] T.G.Szymanski," Dogleg Channel Routing is NP-complete, " *IEEE Transactions on Computer-Aided Design* , CAD - 4 pp. 31-41, January 1985.

[2] H. Bollinger,"A Mature DA System for PC Layout ", *Proceedings of first International Printed Circuit Conference* , 1979.

[3] W.A. Dees, and P.G.Karger,"Automated Rip-up and Reroute Techniques,"*Proceedings of Design Automation Conference,* 1982.

[4] W.A. De,*A Heuristic Global Router for Polycell Layout,* PhD thesis, Duke University, 1986.

[5] C.Y.Lee,"An Algorithm for Path connections and Its Applications," *IRE Transactions on Electronics Computers,* 1961.

[6] K. Mikami, and K. Tabuchi,"A Computer Program for Optimal Routing of Printed Circuit connectors," *IFIPS Proc.,* H47, pp. 1475-1478, 1968.

[7] D.W.Hightower,"A Solution to the Line Routing Problem on a Continous Plane,"*Proc. 6th Design Automation Workshop,* 1969.

[8] E.W.Dijkstra,"A Note on Two Problems in Connexion with Graphs," *Numeriche Mathematik* , 1, pp. 269 - 271, 1959.

[9] J.Heisterman, and T.Lenguar,"The Efficient Solution of Integer Programs for Hierarchial Global Routing," *IEEE Transactions on Computer - Aided Design,* CAD 10(6), pp. 748 - 753, June 1991.

[10] P.K.Agarwal "Ray and other applications of spanning trees with low stabbing number",*Proc. 5th ACM Symposium on Computational Geometry,*(1989) 315-325.