Zone Partitioning of Pins Amidst Rectangular Blocks

A dissertation submitted in partial fulfillment
of the requirements of M.Tech.(Computer Science)
degree of Indian Statistical Institute, Kolkata
by

Sujit Biswas

under the supervision of

Dr. S. Sur-Kolay and Dr. S. C Nandy
Advanced Computing & Microelectronics Unit

Indian Statistical Institute
203, Barrackpore Trunk Road
Kolkata-700 108.

July 9^{th}, 2002

Indian Statistical Institute

203, Barrackpore Trunk Road,

Kolkata-700 108.

Certificate of Approval

This is to certify that this thesis titled **"Zone Partitioning of Pins amidst rectangular blocks"** submitted by **Sujit Biswas** towards partial fulfillment of requirements for the degree of M. Tech in Computer Science at Indian Statistical Institute, Kolkata embodies the work done under my supervision.

Sur Kolay
9.7.2002

9/7/2002

Dr. S. Sur-Kolay,
Dr. S. C. Nandy,
Advanced Computing and Microelectronic Unit,
Indian Statistical Institute,
Kolkata-700 108.

Acknowledgements

I take pleasure in thanking Dr. S. Sur-Kolay and Dr. S. C. Nandy for their friendly guidance throughout the dissertation period. Their pleasant and encouraging words have always kept my spirits up.

I take the opportunity to thank my classmates, friends and my family for their encouragement to finish this work.

Sujit Biswas 9/7/2002

Abstract : *Topological routing has gained importance in state-of-the-art VLSI layout synthesis. This report poses an important problem in VLSI physical design called MinZone partitioning where the objective is to partition the routing area of a given placement of circuit modules into minimum set of zones such that in each zone, no two pin belong to the same net and no two zones are overlapping. Thus topological routing can be completed among the zones where as within the zone the only wires are from a pin to the boundary of the zone. Hence such partitioning is likely to accelerate routing by reducing the problem size. The related problem MaxZone, which pertains to identifying a zone with maximum number of distinct pins is also considered here. Both the problems are observed to be NP-hard. For the first problem a special type of visibility graph is formed, by considering the maximum empty rectangle among the circuit modules. Then the graph is partitioned into minimal number of Cliques using Most Common Neighbor heuristic. Further we use a greedy heuristic for avoiding the mutual overlapping of the zones. Once the zone partitioning is done MaxZone can be obtained in linear time.*

Contents

1 Introduction

The problem of k-way partitioning for reasonably large k is useful in decomposing problem instances into smaller tractable ones. For interconnecting the pins on a given placement of circuit modules, multiway partitioning or clustering of heavily connected net in a close neighborhood have been proposed earlier, to reduce the size of original problem. Previous work on clustering have been mostly at circuit level and are either bottom up or top down[1]. Density based clustering algorithm has also been suggested[2, 9]. As a cluster forms a single node in the reduced problem, these have to be re-expanded later for detailed routing which may result in failure in routing completion due to congestion.

The objective of this report is to study the problem of partitioning the routing space for efficient routing where pins in geometric proximity but belonging to distinct nets are grouped together into a zone. As an example, topological routing [5, 8] is expected to be completed much faster on a smaller instance. For proper decomposition into independent sub problems, the zones to be defined are required to be non-overlapping. Another significant problem which can be accelerated after zone processing, is of area(or number)-balanced partitioning of a given floorplan[6]. The related problem of identifying a zone with maximum number of pins from distinct net may be useful in prioritizing completion of routing within the locality. The above issues are formulated as graph-theoretic optimization problem in Section 2. Efficient heuristic methods for solving these problems are presented in Sections 3, 4 and 5. Experimental results and concluding remarks appear in Section 7 and 8 respectively.

2 Problem formulation

Consider a placement of blocks along with positions of pins of the nets (Figure 1). Pins of each distinct net may be designated a unique color. We may consider a point set P corresponding to the position of the pins along the block boundaries, and the blocks as obstacles. In the problem of zone partitioning, the primary goal is to group the point set into zones of distinctly colored points(distinct net), such that any point in a zone is visible to all other points in its zone. *The notion of visibility is defined below.*

In order to formulate the problem of zone partitioning of the pins described above, we assume the boundary of the placement to be rectangular and henceforth refer to it by floor. The following definitions are required and used later on for the proposed method.

Maximal Empty Rectangle(MER). An empty rectangle on the floor which is not fully contained in any other empty rectangle present on the floor. Thus each side of an MER is bounded by either the boundary of the floor or the boundary of the circuit module present on the floor.

MER-visible pair. A pair of points is said to be an MER-visible pair, or simply visible if they appear on the boundary of an MER. Further in the context of zone partitioning the two points of an MER-visible pair must have different colors. Henceforth, we refer to such a pair as 'visible pairs'.

We formulate our problem of zone partitioning by using a graph, called *MER-visibility graph.*

MER-visibility graph. The nodes of the MER-visibility graph are the points(pins) present on the floor. Two nodes have an edge if they are MER-visible. Any given placement has an unique MER-visibility graph.

Thus, we formally state our problem.

MinZone. The minimum set of cliques (fully connected subgraphs) that partition G such that in each subgraph no two vertices have same color.
Another closely related problem is as follows.

MaxZone. A connected subgraph, with maximum number of vertices such that all of them have distinct colors.

Two points a and b are said to be visible, if they are having different color and the rectangle with a and b at diagonally opposite corners of a rectangle does not intersect

any obstacles. For example, if a and b lie on the boundary of two blocks A and B (A and B may be same block) respectively, then a and b are visible if there exist a maximal empty rectangle with blocks A and B appearing along it boundary. In Figure 3 the pin d of block A is visible to pin c and e of block A, pin c of block C, and pin a and b of block B.

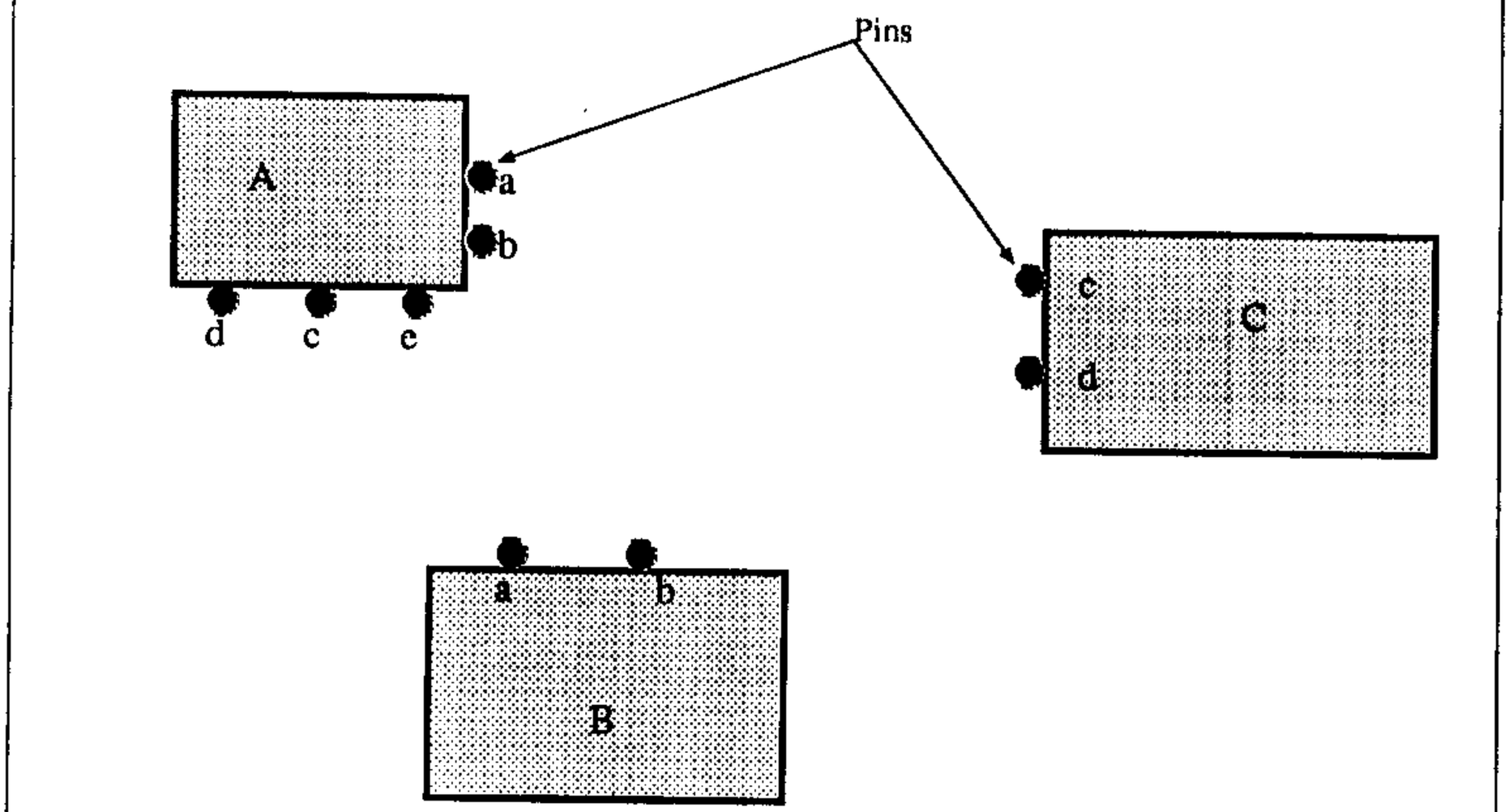


Figure 1 placement of blocks along with pins

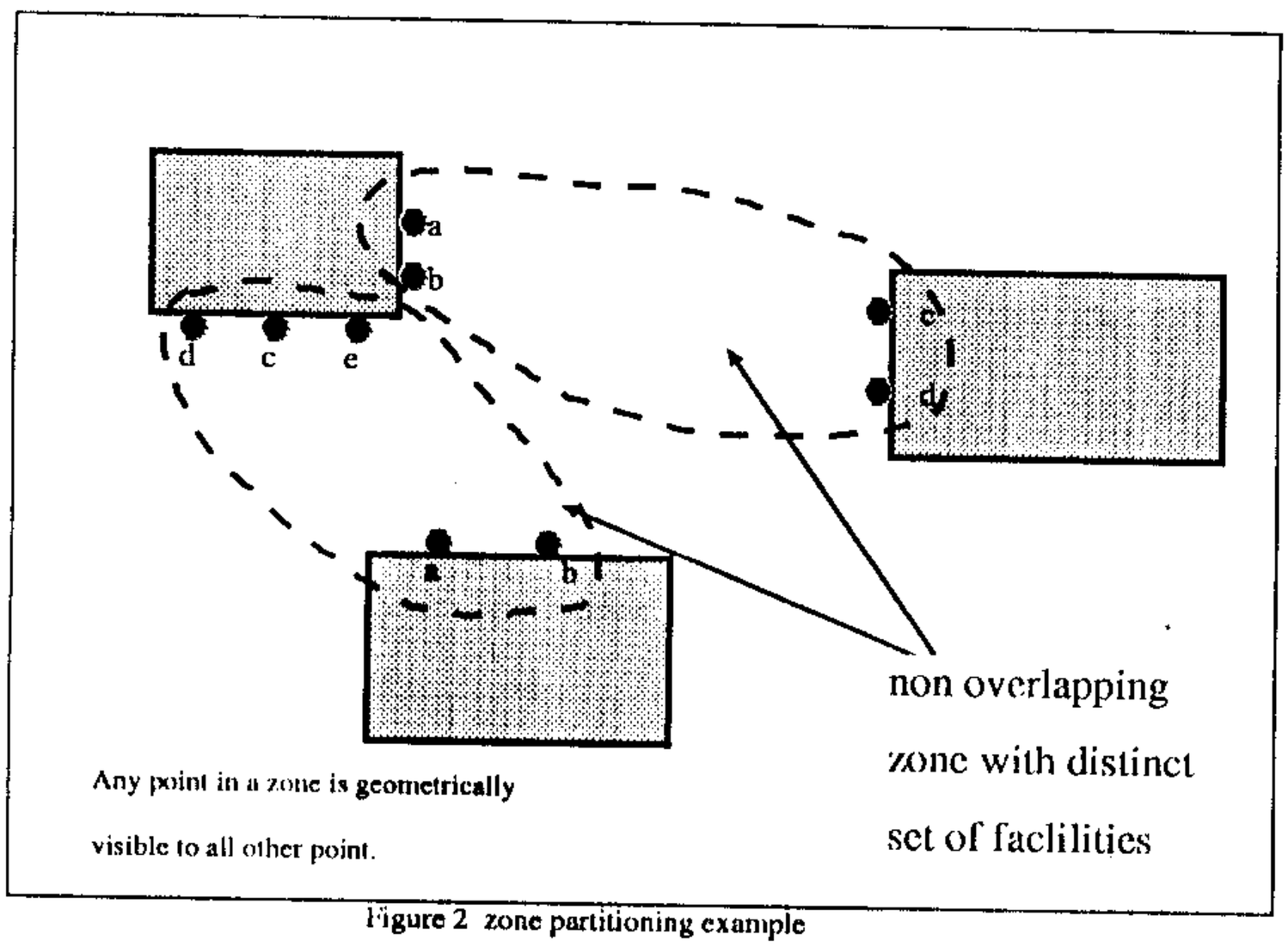


Figure 2 zone partitioning example

3 MinZone Problem : An Overview

This problem was addressed in [10] and conjectured to be NP-complete. A Genetic Algorithm approach was presented, but it does not consider the issue of overlapping zones. Our proposed algorithm here tackles this issue and is based on greedy heuristics. In this section we present the overview of an algorithm for computing the MinZone of a given placement. The algorithm consists of the following steps.

1. Mer: Finds all MER's on the floor and computes the MER-visibility graph.

2. CliquePartitioning: Partitions the MER-visibility graph into minimum number of disjoint cliques. Each clique represents a zone

3. Refinement: Refines the partition such that the corresponding zones are geometrically non-overlapping.

The subsection below discuss the details of the above three steps.

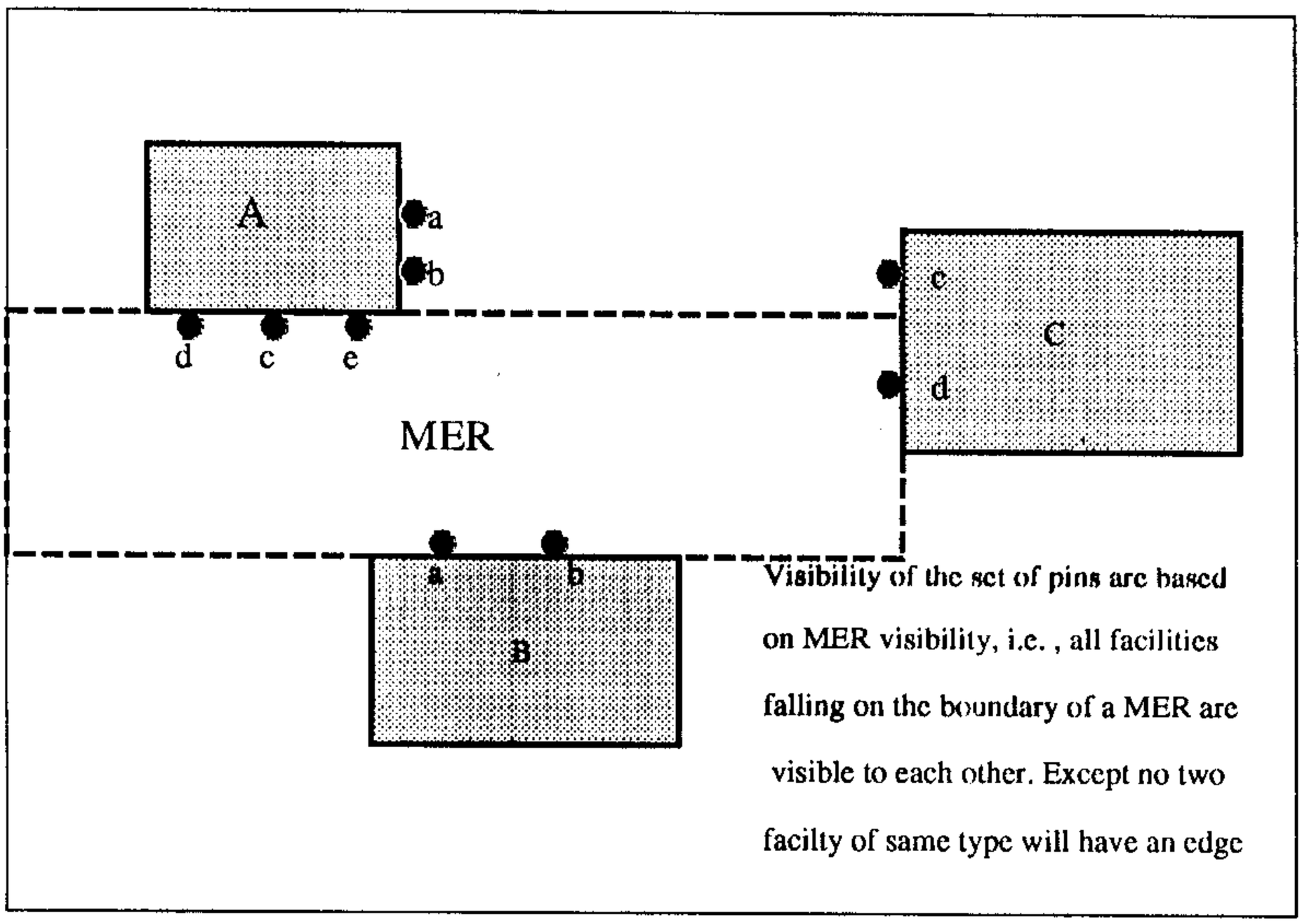


Figure 3 MER-visibility

3.1 MER

The algorithm MER is based on classical line-sweep paradigm. The algorithm uses an interval tree as the underlying data structure. The purpose of algorithm MER is to generate all maximal empty rectangle given a placement of blocks, as the MER's are generated MER-visibility graph for the given set of pins are updated. In Figure 3 all the pins on the boundary of MER are visible to each other. Note that no two pins of the same net will have an edge in the visibility graph even if they have MER visibility. Below we describe the method of generating MER's.

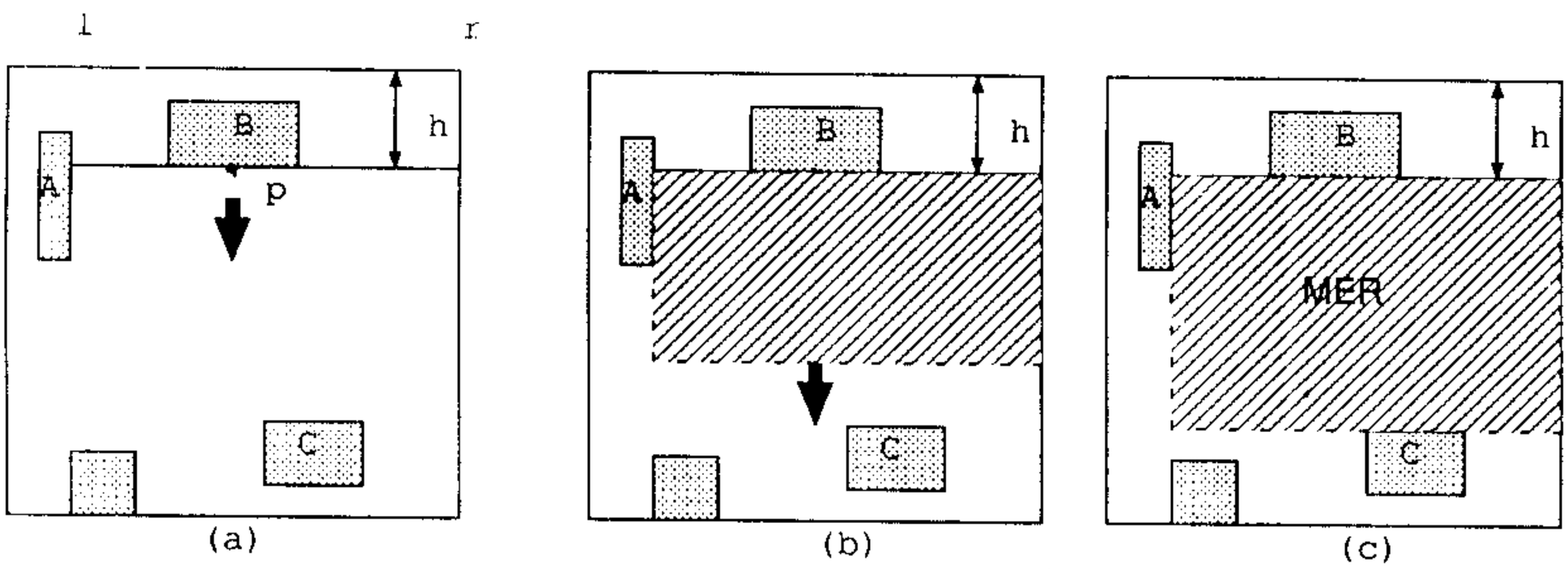
The key idea behind the identification of all MER's is to scan a horizontal sweep line across the rectangular floor. This leads to the concept of a *window* , which is somewhat analogous to a horizontal peeping slot. Windows are generated and deleted dynamically

when solid blocks in the floor are processed and their database is managed by interval tree. Two types of windows, primary and secondary, are generated while processing the bottom and top of solid blocks respectively.

Processing the bottom of solid block

Consider a solid block whose bottom s lies at height h see Figure 5. Consider a point p on s, and draw a line from p to the left parallel to the x-axis, till it hits the east side of a solid block or the west boundary of the floor. Similarly, extend the line from p to the right. Let l and r denotes the x-coordinate of the above hit-points. A primary window is said to be generated in this event and is represented by a ordered tuple $([l, r], h)$, where $[l, r]$, is a horizontal interval and h denotes the height from where it is originated(see Figure 5). The roof of the floor is treated as a bottom of a dummy solid block.

Insertion of a window $S([l, r], h)$ in the interval tree is solely dictated by the interval $[l, r]$. When a interval is to be inserted in T (interval tree), a top down scan is made starting from the root of T till it finds a node w satisfying the condition $l \leq d(w) \leq r$ for the first time. Window S is then attached to node w, i, e the values of l, r, h are inserted into the appropriate node of the secondary structure(*linked list* $w.L$) attached to node w, if it is not already there. Similarly, while deleting a window, the corresponding node is removed from the secondary structure(*linked list* $w.L$) where it is attached.



(a):Generation of new window ([l,r],h)while processing the bot of block B.
(b):A curtain coinciding with the interval [l,r] cotinues to fall from ht h.
(c):The window remains active until the falling curtain first hits a solid block C. The corresponding MER is identified.

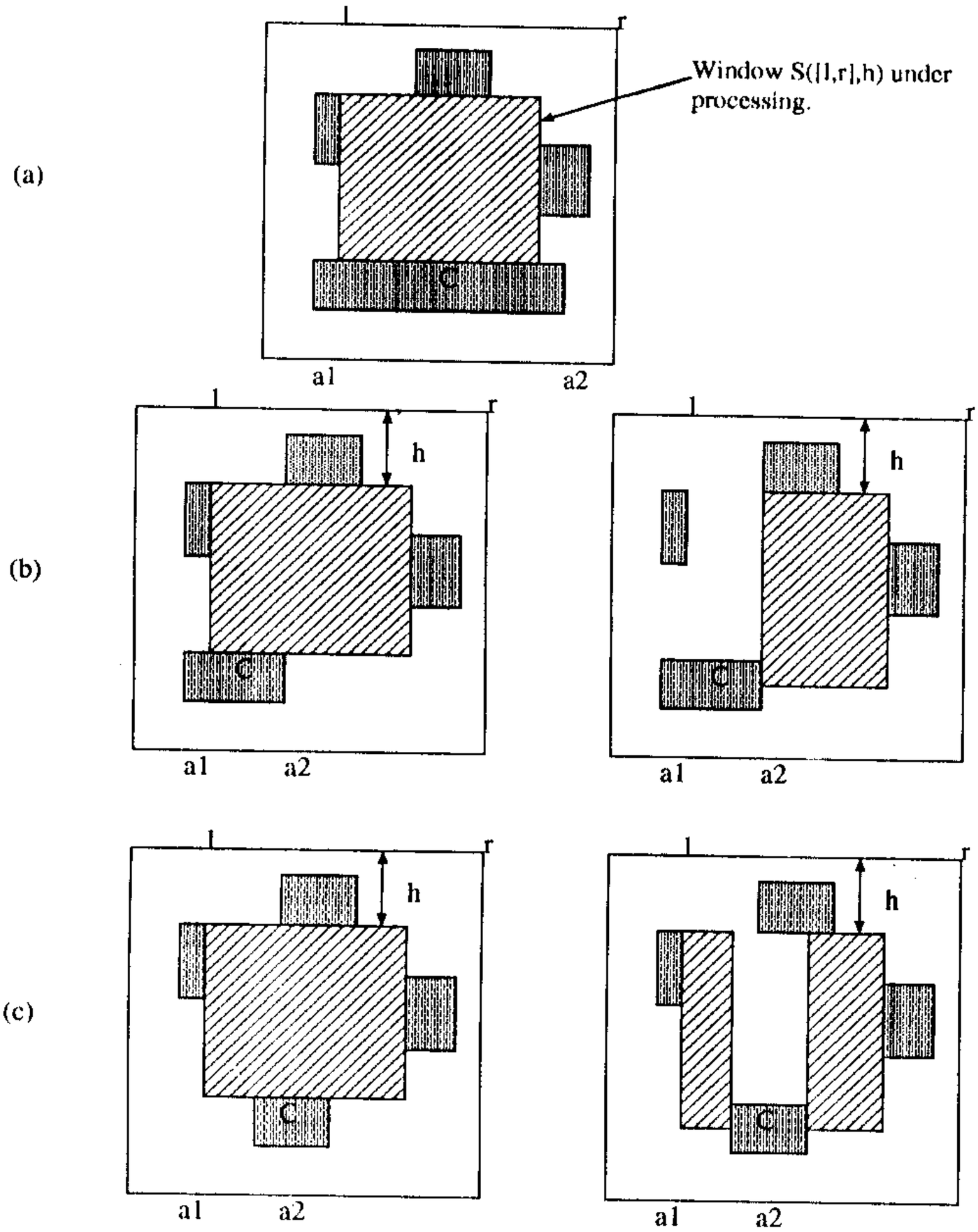
Figure 5 : Processing the bottom of solid block B

Processing the top of solid block

Consider a window $S([l,r],h)$ which strikes the solid block $C[(a_1,b_1),(a_2,b_2)]$, as in Figure 6. Before it becomes inactive, it returns an MER $[(l,h),(r,b1)]$ and splits itself to generate zero, one or two windows whose intervals are subsumed by the interval $[l,r]$ of the parent window S and whose height are inherited from S. These windows are called *secondary windows*, which in turn may generate other secondary windows. In Figure 6 we describe the generation of MER, splitting a window into secondary window(s) while processing the top of a solid block.

Constructing the MER-visibility graph

The MER-visibility graph is represented by a n x n matrix, where n is the number of points(pins). The MER-visibility graph or the matrix is updated as soon as an MER is detected.



The following Cases can occur:

Case1. If $l \geq a1$ and $r \leq a2$, then S becomes inactive leaving no offsprings

Case2. If $l \geq a1$ and $r > a2$, then S becomes inactive and a secondary window ([a2,r],h) is generated.

Case3. If $l < a1$ and $r \leq a2$, then S becomes inactive and a secondary window ([l,a1],h) is generated.

Case4. If $l < a1$ and $r > a2$, then S becomes inactive and a secondary window ([l,a1],h) and ([a2,r],h) is generated.

Figure 6 : Processing the top of solid block C (a) Case 1. (b) Case 2. (c) Case 4

3.2 Clique Partitioning

Our next task is to partition the MER-visibility graph $G(S, E)$ into a minimum number of disjoint cliques. We use the heuristic approach proposed by Tseng and Siewiorek[TsSi86] for this purpose. The heuristic approach is described below.

A supergraph $G'(S, E')$ is derived from the original graph $G(V, E)$. Each node $s_i \in S$ is a supernode that can contain a set of one or more vertices $v_i \in V$. Note that the vertices attached to the supernodes are mutually disjoint. Initially each supernode $s_i \in S$ contains a single node $v_i \in V$. E' is identical to E except that edges in E' now link supernodes in S. During the execution of algorithm, pair of supernodes are merged to form bigger supernodes at each stage. A supernode $s_i \in S$ is a common neighbor of the two supernodes s_j and $s_k \in S$ if there exist edges $e'_{i,j}$ and $e'_{i,k} \in E'$. We use two function in the algorithm Clique Partitioning described in section 5.2.

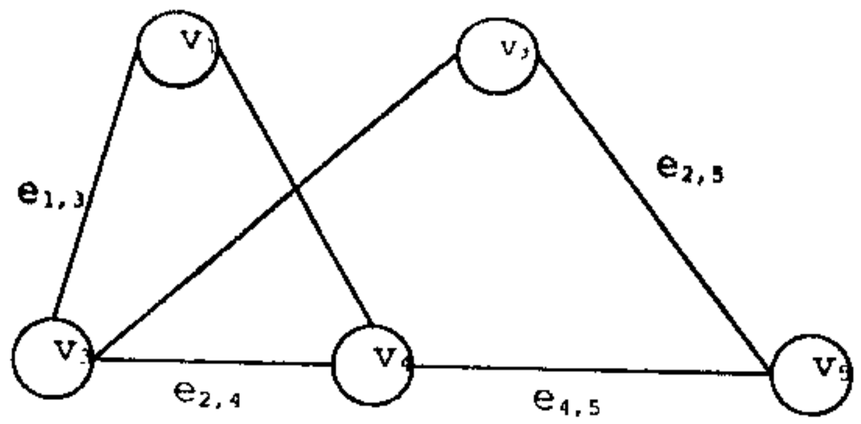
1. COMMON_NEIGHBOR (G', s_i, s_j): returns the set of super-nodes that are common neighbors of s_i and $s_j \in G'$.
2. DELETE_EDGE(E', s_i): deletes all edges in E' which are incident to s_i.

At each step the algorithm finds the super-nodes s_{Index1} and $s_{Index2} \in S$ such that they are connected by an edge and have maximum number of common neighbor. The common neighbors are stored in *CommonSet*. These are referred to as the common neighbor of the edge $e'_{Index1,Index2} \in E'$. These two super-nodes are merged into a single super-node, $s_{Index1Index2}$, which contain all the vertices of s_{Index1} and s_{Index2}. All edges originating from s_{Index1} and s_{Index2} in G' are deleted. New edges are from $s_{Index1Index2}$ to all the super-nodes in the *CommonSet*. Also the weight of $s_{Index1Index2}$ is computed, which is number of vertices contained in it. The above steps are repeated until there is no edge left in the graph. The vertices contained in each super-nodes $s_i \in S$ forms the clique of graph G.

Figure 4 illustrates the above algorithm. In the graph of Figure 4a, $V = \{v_1, v_2, v_3, v_4, v_5\}$ and $E = \{e_{1,3}, e_{1,4}, e_{2,3}, e_{2,5}, e_{3,4}, e_{4,5}\}$. Initially each vertex is placed in a separate super-node (labeled s_1 through s_5. In the first iteration (see Figure 4b) the three edges $e_{1,3}, e_{1,4}$ and $e_{3,4}$ of supergraph G' have maximum no of common neighbors among all edges. The first edge $e_{1,3}$ is selected and the following steps are carried out.

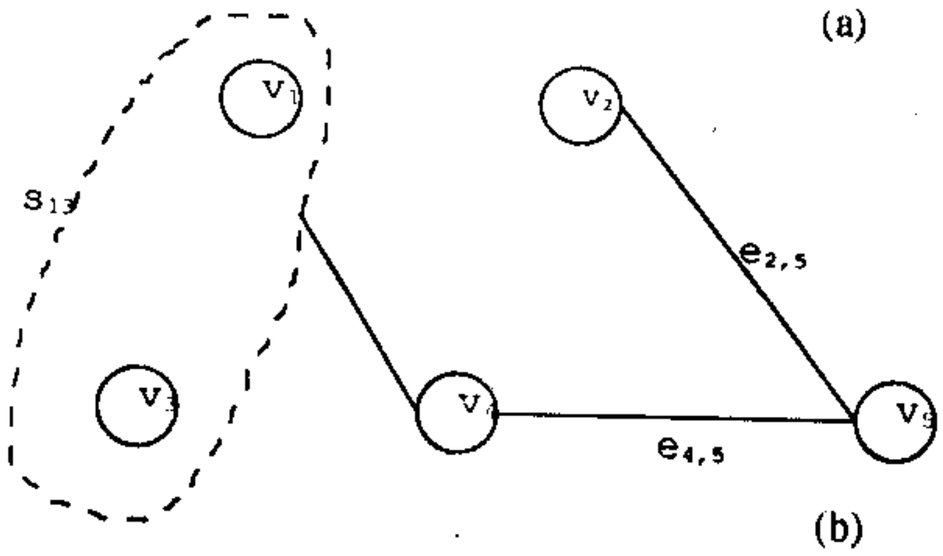
1. s_4, the only common neighbor of s_1 and s_3 is put in *CommonSet*.
2. All edges are deleted that link either super-nodes s_1 or s_3.
3. Super-nodes s_1 and s_3 are combined into a new super-node s_{13}
4. An edge is added between s_{13} and each super-node in *CommonSet* i.e., the edge $e_{13,4}$ is added.

The iteration continued until all the edges are deleted (see Figure 4c and 4d).



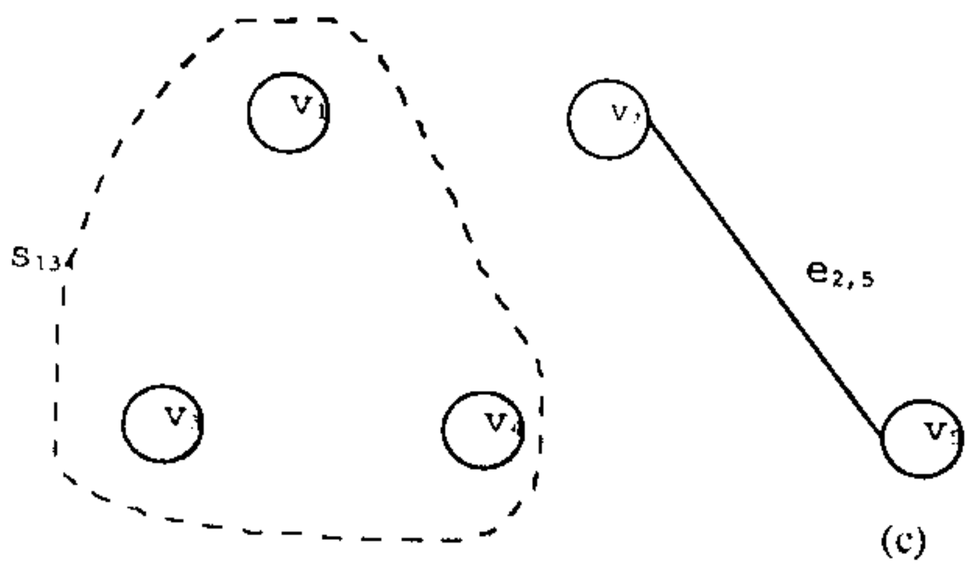
(a)

edge	common neighbor
$e_{1,3}$	1
$e_{1,4}$	1
$e_{2,3}$	0
$e_{2,5}$	0
$e_{3,4}$	1
$e_{4,5}$	0



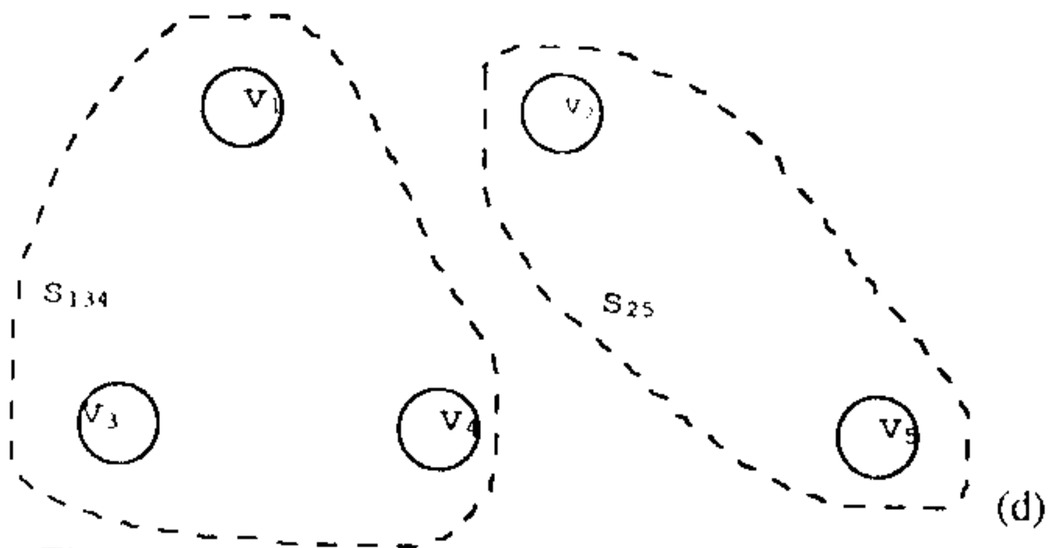
(b)

edge	common neighbor
$e_{13,4}$	0
$e_{2,5}$	0
$e_{4,5}$	0



(c)

edge	common neighbor
$e_{2,5}$	0



(d)

cliques

$s_{134}=\{v_1, v_3, v_4\}$

$s_{25} =\{v_2, v_5\}$

Figure 4: (a) calculating common neighbour for edges of graph. (b) supernode s_{13} is formed by cosidering edge $e_{1,3}$ (c) supernode s_{134} formed by cosidering edge $e_{13,4}$ (d) supernode s_{25} formed by cosidering edge $e_{2,5}$.

3.3 Refinement

After Clique Partitioning, each clique corresponds to a zone which is actually the convex hull of the set of points(pins). Since we have worked on the visibility graph, and ignored the geometry(position of nodes on the floor), the zones may overlap. So we use Refinement algorithm for making the zones non-overlapping. The algorithm Refinement checks whether the zone formed are non-overlapping. If any two zone are overlapping the smaller zone is broken up into zones with single pin.

It needs to be mentioned that, initially we have attached a unique *id* with each points(pins). For a given block, the numbering of pins are done in the order *top*, *left*, *right*, and *bottom*. The blocks are traversed according to their top edge y coordinate. Numbering of zones are done in increasing order. A *Zone* is identified with the pin with lowest *id* in it.

After breaking down overlapping zones (if any) a greedy heuristic is run for further refinement. The heuristic starts from the first zone and traverses to the last zone. If Z_i is the zone currently being processed, it picks up pin p from the remaining higher numbered zones such that p is visible to all existing pins in Z_i. While the pin p is being picked a check is done whether the resulting zone $(Z_i + p)$ is overlapping with any of the existing zone. If no overlap is there p is added to zone Z_i and zone Z_i is redefined. i.e., convex hull for zone Z_i is updated. Also the weight of zone Z_i is updated. The process is continued till all the remaining pins are exhausted. Then the heuristics proceeds with the next zone.

4 MaxZone Problem

In this problem we find a connected subgraph with maximum number of vertices in G such that all of them have distinct colors.

The zone with the maximum number of distinct points(pins), where no two pins are same can be found out in linear time after the Refinement Algorithm. The supernode $S(i)$ with maximum weight represent the zone with maximum number of pins.

5 Implementation Details

5.1 MER

Data Structures

To find all MER's, our algorithm also needs the following data structures:

(a) A list Y, whose element have three fields as follows:
1. $Y(i).y$: the Y-coordinate of the top or bottom of solid block.
2. $Y(i).d$: the dimension of the corresponding solid block.
3. $Y(i).ind$: a tag t or b to indicate top or bottom of the associated solid block.

(b) Two AVL trees : TEMP-TREE-L and TEMP-TREE-R.

(c) Interval tree : An interval tree (T) is a leaf-oriented balanced binary search tree where leaf nodes from left to right holds distinct x-coordinates of the left and right sides of solid blocks(whose extremes are are set $\{x_1, x_2, ...x_{2n}\}$) sorted in ascending order. Each internal node w will have the following information:

1. Let T' be a subtree with a set of leaf nodes $\{x'_1, x'_2, ...x'_m\}$. The root of the subtree T' has discriminant $d(w)$. The discriminant value of the leaf node is assumed to be x-coordinate attached to it.

$$d(w) = \frac{(x'_{\lfloor m/2 \rfloor} + x'_{\lfloor m/2 \rfloor + 1})}{2}$$

2. The left subtree of w in the interval tree with leaf nodes $\{x'_1, x'_2, ...x'_{\lfloor m/2 \rfloor}\}$, and right subtree in the interval tree with leaf nodes $\{x'_{\lfloor m/2 \rfloor + 1}, ...x'_m\}$.
3. A secondary list ($w.L$) of nodes with three fields $L.l$ $L.r$ and $L.h$, sorted in increasing order with respect to $L.h$, is attached to each node w of T in the form of a doubly linked list with and additional forward link from w to the last node in the list.

(d) Visibility Matrix M: for the set of pins.

A node in T is active if it contains non-empty secondary list or it has active nodes in both of its subtrees. The active nodes in the interval trees are also linked using two different pointers, w.LPTR and w.RPTR, in the form of a binary tree.

Scheme of Algorithm MER

Initially an empty interval tree T is created with left and right side of the solid blocks in sorted order. A window $([0, a], 0)$ corresponding to the roof of the floor [(0, 0), (a, b)], is inserted in the interval tree. We start processing solid block in proper sequence. Let $P[(a_1, b_1), (a_2, b_2)]$ be the current solid block whose top is to be processed.

Search the interval tree from the root to get node $w*$ such that $a_1 \leq d(w*) \leq a_2$. Let P_{IN} be the set of nodes traversed from the root to $w*$ (excluding $w*$). From $w*$ scan upto leaf level to get a_1 and a_2. Let P_L and P_R be the set of nodes along these two paths. It is obvious that all active windows in T that intersect the interval $[a_1, a_2]$, will appear with the nodes in P_{IN}, P_L and P_R. As a matter of fact all windows associated with $w*$ will intersect $[a_1, a_2]$. The window associated with nodes in P_{IN}, P_L and P_R other than w* may or may not intersect $[a_1, a_2]$. The search for such nodes can be efficiently accomplished using pointers $w.LPTR$ and $w.RPTR$. All windows intersecting the interval $[a_1, a_2]$ will generate corresponding MER's and can be found in $O(nlogn)$ plus number of reported intersections with no additional search overhead.

While processing the top of solid block $P[(a_1, b_1), (a_2, b_2)]$ two pointers POINTER_L and POINTER_R are maintained. Initially these pointers point to the root of the T. Subsequently, POINTER_L will point to the current node of the interval tree which contains the last inserted secondary window with right extremity at a_1, and POINTER_R will point to the current node of the interval tree which contains the last inserted secondary window with left extremity at a_2.

Let w be a node whose associated windows may return an MER. The windows in $w.L$ are processed from end of the list (i.e., the windows with largest interval first and then in decreasing order) until a window, non-intersecting the interval $[a_1, a_2]$ is found. Note that each time a secondary window is to be inserted in the interval tree, the search for its position is to be initiated from the node currently pointed by either POINTER_L or POINTER_R, down the tree.

Algorithm MER

Input: A rectangular floor [(0, 0), (a, b)] with n solid blocks. Each block with their given set of pins.

Output: 1. All MER's. 2. Visibility Graph for the given set of pins

Method:

Begin

1. create an interval tree with x-coordinate of the left and right sides of solid blocks.

2. insert a window $([0, a], 0)$ corresponding to the roof(top side) of the floor.

3. create a sorted list Y, i, e., arrange the top and bottom sides of solid blocks in p.oper sequence.

4. insert a in TEMP-TREE-R and 0 in TEMP-TREE-L (which are initially empty);

5. **repeat**(process the list Y in increasing order of Y[i].y)
   let $P[(a_1, b_1), (a_2, b_2)]$ be the current solid block;
   if $Y[i].ind = t$ perform 5.1 thru 5.4 else perform 5.5;

   5.1. scan T from root to find the first node $w*$ such that $a_1 \leq d(w*) \leq a_2$. form three sets of node P_{IN}, P_L and P_R in the interval tree;

   5.2. find all windows attached to $w*$ and nodes in P_{IN}, P_L, P_R whose intervals intersects $[a_1, a_2]$. Let these set of windows be be A.

   5.3. for each member A $\in A$

   (a) return the corresponding MER, update Visibility Matrix M and and generate secondary windows.

   (b) delete A from T and insert the secondary windows found in 5.3(a) using POINTER_L or POINTER_R.

5.4. insert a_1, a_2 in TEMP-TREE-R and TEMP-TREE-L respectively;

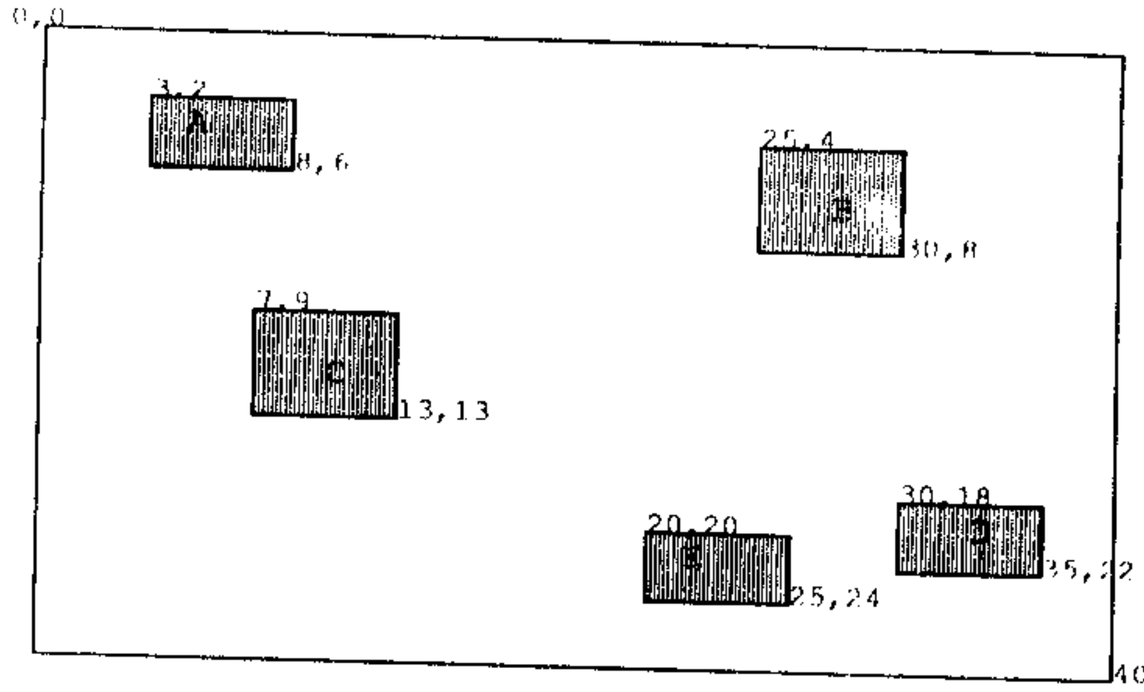
5.5.(a) find the left (right) extremity of the primary window associated to the bottom of current solid block from TEMP-TREE-R (TEMP-TREE-L) and generate corresponding window;

(b) insert the window found in 5.5(a) and delete
a_1, a_2 from TEMP-TREE-R and TEMP-TREE-L respectively;

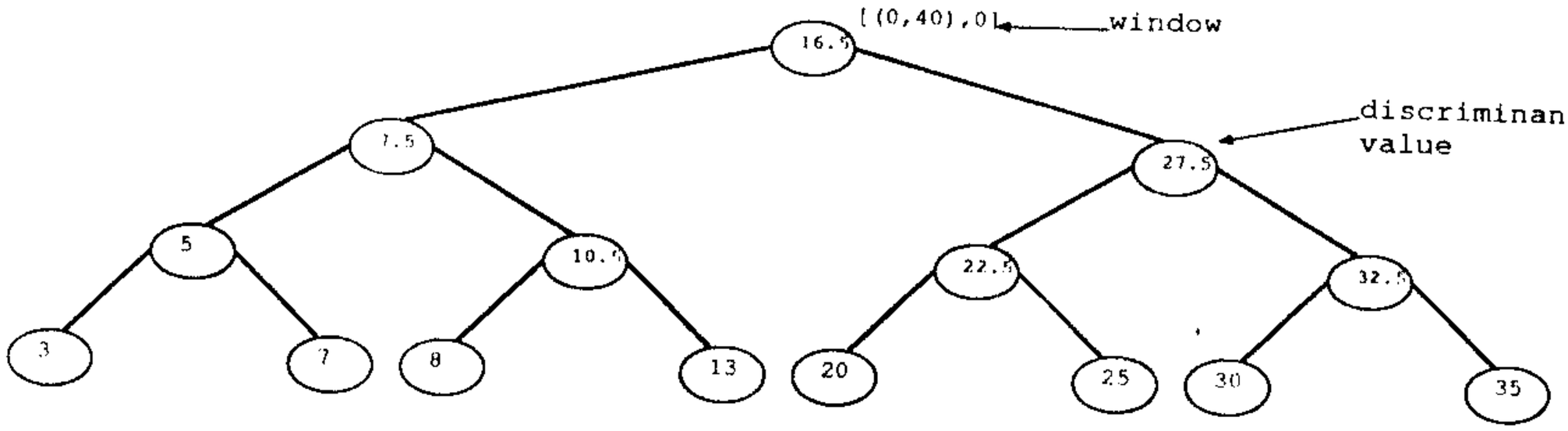
until(top and bottom of all solid blocks are processed);

6. process the bottom of the floor as the top of a dummy solid block.

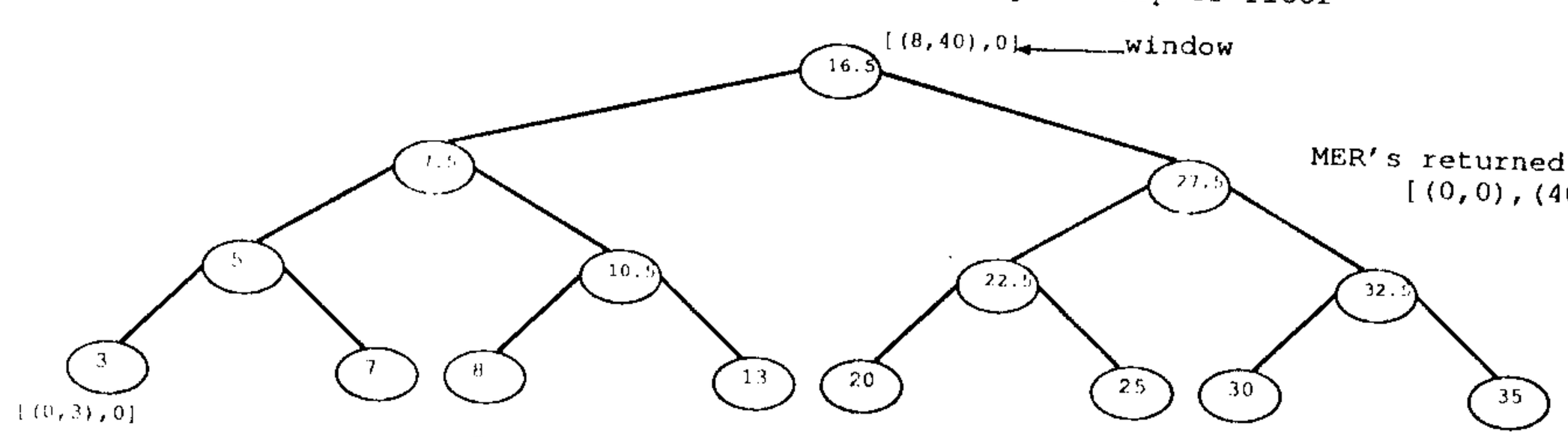
end.



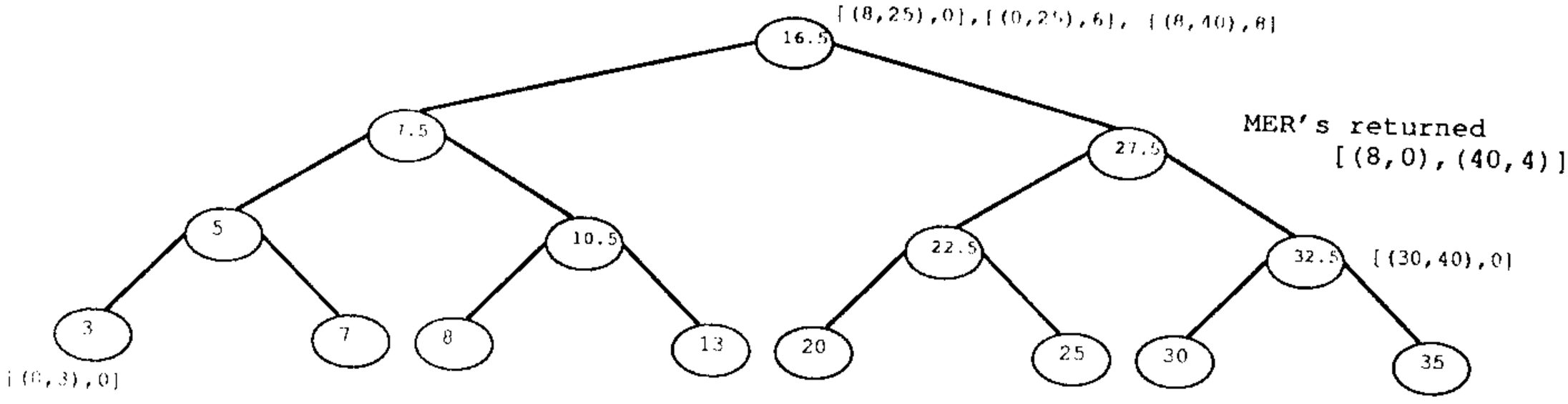
Example floor.



Interval tree after processing the top of floor



Interval tree after processing the top of solid block A



Interval tree after processing the top of solid block B and bottom of A and B.

Figure 7 Execution trace of Algorithm MER

5.2 CliquePartitioning

Data Structures

(a) A Supernode S, whose element have three fields as follows:

1. $S(i).wt$: the wt of a supernode is the number of vertices it contains.

2. $S(i).list$: the list of vertices in the supernode.

3. $S(i).zone$: the convex hull of the set of vertices.

The pseudo code of the algorithm using the above data structures is presented below.

Algorithm : CliquePartitioning //create a super graph $G'(S, E')$

$S = \phi$; $E' = \phi$;
for each $v_i \in V$ do $s_i = v_i$; $S = S \cup \{s_i\}$ endfor
for each $e_{i,j} \in E$ do $E' = E' \cup \{e'_{i,j}\}$ endfor

while $E' \neq \phi$ do

//find s_{Index1}, s_{Index2} having most common neighbor
$MostCommons = -1$
for each $e_{i,j} \in E'$ do
  $c_{i,j} = \mid COMMON\_NEIGHBOUR(G', s_i, s_j) \mid$
  if $c_{i,j} > MostCommon$ then
    $MostCommons = c_{i,j}$
    $Index1 = i; Index2 = j;$
  endif
endfor

$CommonSet = COMMON\_NEIGHBOUR(G', s_{Index1}, s_{Index2})$;

//delete all edges linking s_{Index1}, s_{Index2}
$E' = DELETE\_EDGE(E', s_{Index1})$;
$E' = DELETE\_EDGE(E', s_{Index2})$;

//merge s_{Index1} and s_{Index2} into $s_{Index1Index2}$
$S = S - s_{Index1} - s_{Index2}$;
$S = S \cup \{s_{Index1Index2}\}$;
Update weight of $s_{Index1Index2}$;

//add edge from $s_{Index1Index2}$ to super-nodes in $CommonSet$
for each $s_i \in CommonSet$ do
$E' = E' \cup \{e'_{i,Index1Index2}\}$;
endfor

endwhile

5.3 Refinement

Algorithm: Refinement

Input:1. An array of supernodes obtained from CliquePartitioning algorithm.
Each supernodes representing a zone, which may be overlapping
2. z_count : number of zones

Output:Array of supernodes each representing a zone which are non-overlapping.

Method:

Begin

1.for i=1 to z_count
Define_Zone : convex hull of the set of pins in supernode $S(i)$

2.Break_Overlapping_Zones.
break the smaller of any two overlapping zones, into zones
with single pins.

3.Let z'_count be number of zones after splitting.
for i=1 to $z'\_count - 1$

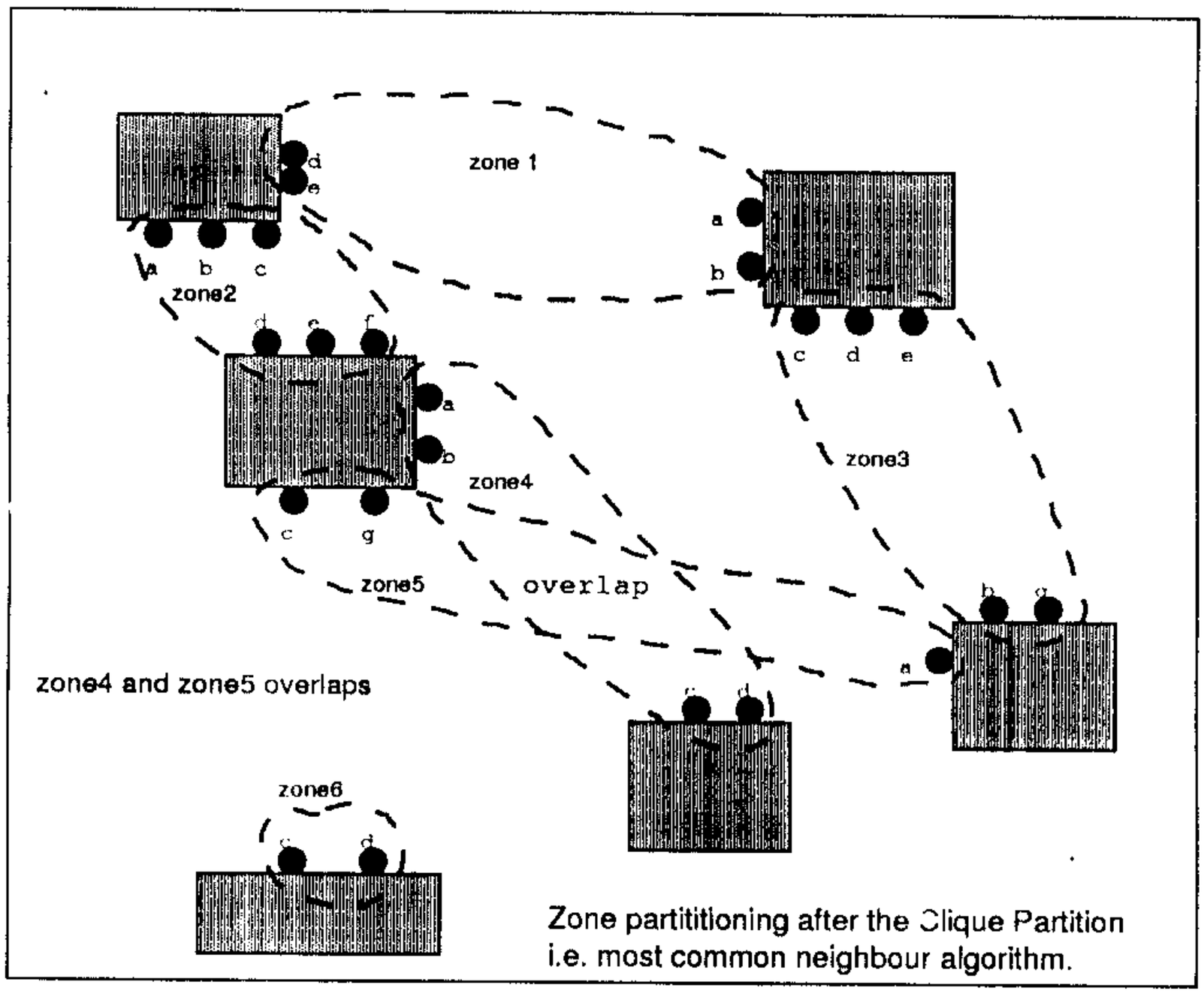
for j=i+1 to z'_count

let pin $p \in S(j)$
if pin p is visible to all pins in $S(i)$ and zone $S(i) + p$ does not overlap with any existing zone, attach p with $S(i)$, Update convex hull of $S(i)$, Update weight of $S(i)$.

endfor

endfor

end



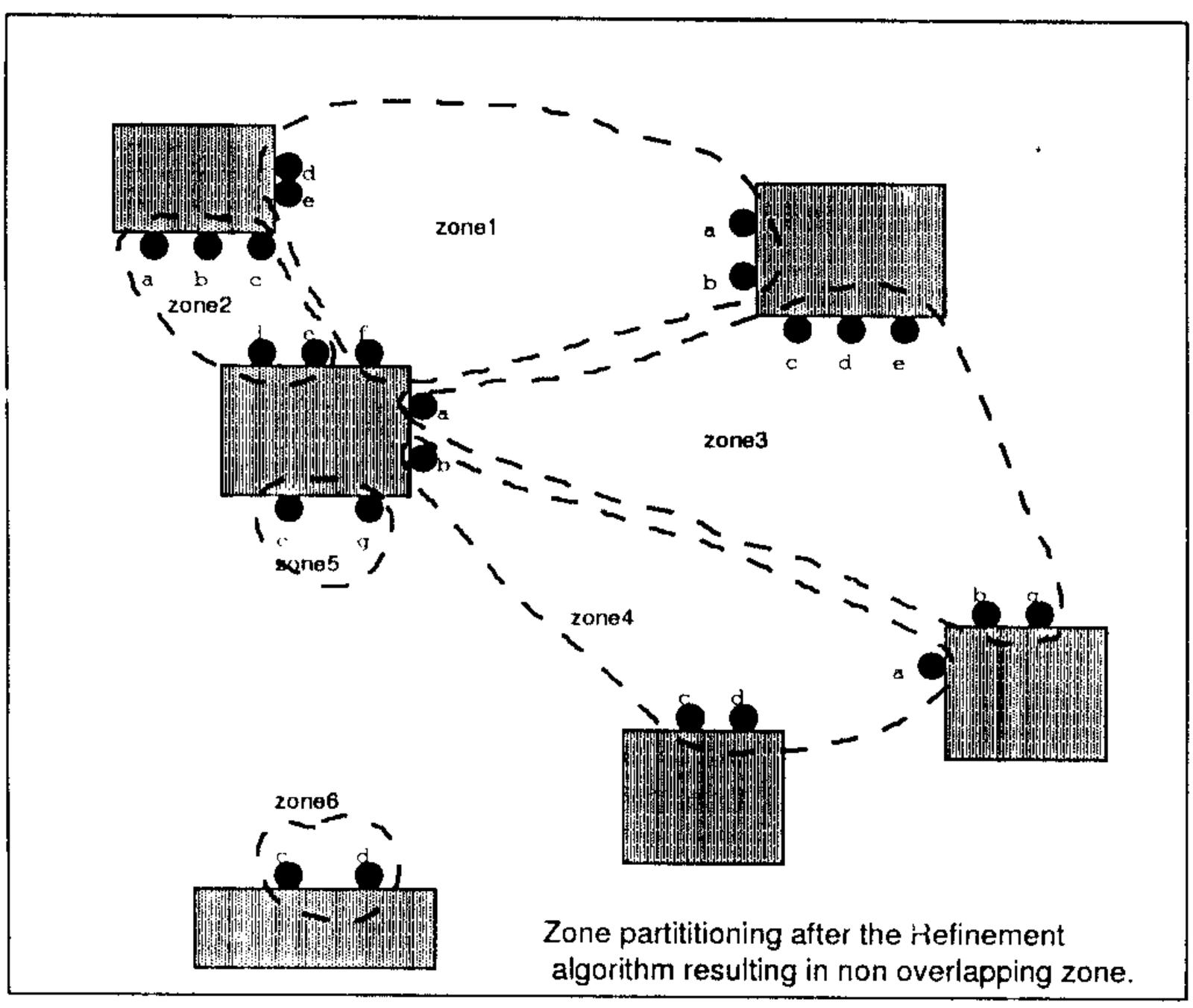


Figure 8 Zone partionionig before and after Refinement Algo.

5.4 MaxZone

Algorithm MaxZone

Input: An array of supernodes obtained after the Refinement Algorithm.

Output: Supernode $S(index)$ with the max number of vertices.

Method:

Begin

$index = 0; wt = 0;$

for $i = 1$ to z_count

if $wt \leq S(i).wt.$

$index = i$;$wt = S(i).wt.$

endif

endfor

end.

6 Complexity

6.1 MER

The algorithm MER runs in $O(nlogn + \mathcal{R})$ time in the worst case, where n is the number of solid blocks and $\mathcal{R}$ denotes the number of reported MER's.The space complexity of the algorithm is $O(n)$. Visibility matrix takes $O(t^2)$, where t is the number of points(pins).

6.2 CliquePartitioning

The time complexity of the algorithm is $O(t^3K^2/2)$[TsSi86], where K is the number of Cliques obtained and t is the number of nodes in the graph. It is reported that the space complexity of the algorithm is $O(t + e)$, where t is the number of nodes and e is

the number of edges. It may be pointed that the MER-visibility graph being dense, the number of edges is $O(t^2)$. Hence in our case the space complexity is $O(t^2)$.

6.3 Refinement

Finally the Refinement algorithm runs in $O(Kt)$ time, where t denotes the number of pins and K denotes the number of zones. The space complexity of the algorithm is $O(t)$.

6.4 MaxZone

After running the MinZone algorithm the extra time needed to compute MaxZone is $O(K)$. The space complexity of the algorithm is $O(K)$, where K denotes the number of zones.

6.5 Overall Complexity

Both time and space complexity of the MinZone algorithm is dominated by CliquePartitioning Algorithm. Hence time complexity of MinZone is $O(t^3K^2/2)$, where K is the number of Cliques obtained. The space complexity is $O(t^2)$, where t is the number of nodes in the Visibility graph obtained by MER algorithm.

7 Experimental Results

The proposed algorithms were implemented in C in UNIX environment on SUN SOLARIS workstation(CPU 296 Mhz) . The input file contains all the blocks, their position($[x, y]$ coordinate), *blocknumber*. Each pin is uniquely defined by its *blocknumber*, *location*, *type* and which *edge* of the block(*top*, *bottom*, *left* or *right*) it is located.

The algorithm when run in Linux environment was taking much less time compared to SUN SOLARIS. Configuration of the Linux machine: Compaq Deskpro EN, CPU Pentium III(750 Mhz), RAM 64 mb.

Table 4.1 : results for problem MinZone

example	vertices	zones obtained	CPU time(secs) Solaris	Linux	MaxZone size
1	15	5	. 01	. 001	5
2	27	8	. 01	. 002	6
3	31	8	. 02	. 004	6
4	35	8	. 02	. 006	6
5	40	10	. 03	. 009	6
6	57	11	. 08	. 010	8
7	67	12	. 12	. 010	10
8	75	18	. 14	. 020	7
9	100	23	. 22	. 040	10
10	122	27	. 43	. 070	10
11	138	31	. 55	. 080	10
12	146	36	. 61	. 100	14
13	168	40	. 65	. 120	9
14	184	44	1.15	. 170	11

8 Conclusion

In this report we proposed a new method of partitioning a set of pins on a given placement of blocks such that all pins in a zone belongs to distinct nets and no two zones are overlapping. A graph theoretic algorithm, followed by a greedy heuristic is adopted for minimum zone partitioning problem, which works on a special type of visibility graph of the set of pins, called MER-visibility graph. Detailed experiments are performed on a randomly generated floorplan. We initially fix the number of blocks and the number of nets on the floor. Then the blocks were generated randomly and the pins attached to block are also generated randomly. Experimental results are observed to be encouraging with respect to time and quality of output.

In Clique Partitioning Algorithm we are using the Most Commom Neighbour heuristic. When the edge with Most Common Neighbors is chosen, ties are resolved arbitrarily. If we can use the information regarding positions of the pins (of MER-visibility graph) while resolving the ties, we might be able to come up with better solutions.

9 References and Tools

References

1. C.J. Alpert and A.B. Kahng, "Recent development in netlist partitioning": A Survey, " Integration: the VLSI journal, 19(1-2), pp, 1-81, 1995.

2. J. Cong and M. Smith "A parallel bottom up clustering algorithm with application to circuit partitioning in VLSI design, " Proc DAC, pp, 755-760, 1993.

3. M.R. Garey and D.S. Johnson, Computer and Intractability

4. S.C. Nandy(Indian Statistical Institute) PhD Thesis

5. S. Bhunia, S. Majumder, S. Sur-Kolay, "Topological routing among polygonal obstacles". Proc Int'l Conf on VLSI design.

6. S. Sur-Kolay, S.C. Nandy, S. Majumder and B. B. Bhattacharya " Area(number)-Balanced Hierarchy of staircase Channels

7. Preparata and Shamos, Computational Geometry : An Introduction. Springer Verlag, NY, 1985

8. N. Sherwani, Algorithms for VLSI Physical Design Automation.

9. A. Singh and M.M. Sadowska, "Circuit clustering using graph coloring, " Proc. Int'l Symp on Phisical Design, 1999.

10. K. Sinha, S. Sur-Kolay, P. S. Dasgupta and B. B. Bhattacharya "Partitioning routing area into Zones with Distinct Pins"