LOCATION OF LARGEST EMPTY STAIRCASE POLYGON AMONG POINT OBSTACLES

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

By

DEBASHREE GHOSH

under the supervision of

DR. BHARGAB B. BHATTACHARYA

and

MR. SUBHAS C. NANDY

JULY 1992

INDIAN STATISTICAL INSTITUTE

203 B.T. ROAD, CALCUTTA - 700 035

ACKNOWLEDGEMENTS

I am very grateful to my guide, Dr. B. B. Bhattacharya who gave me the opportunity to work in the interesting field of computational geometry. I would also like to thank Mr. Subhas C. Nandy for his help and guidance during this period. Last but not the least, I would like to thank my friends Arani Sinha and Kaushik Dasgupta for their moral support.

Debashree Ghosh

LOCATION OF LARGEST EMPTY STAIRCASE POLYGON AMONG OBSTACLES

Abstract

This dissertation outlines an algorithm for identifying an empty staircase polygon of maximum area, among a set of points distributed on a euclidean plane. The problem is formulated using the concept of permutation graphs and an $O(n^3)$ time and $O(n^2)$ space algorithm is proposed.

Keywords: Computational geometry, algorithms, search trees, permutation graphs.

1: INTRODUCTION

The problem of finding the largest convex k-gon in a simple polygon[1] is a classical problem in computational geometry. A related problem of recognizing a convex k-gon of maximum area or perimeter, amidst a set of point obstacles whose vertices are members of the point set, is reported in [2,3]. In VLSI layout design, isothetic polygons play a major role; the sides of these polygons are parallel to co-ordinate axes. In context to VLSI layout design, the problem of finding an empty rectangle of maximum area in an ensemblement of a set of randomly distributed points, was first proposed in [4], and an algorithm of time complexity O(min (n^2, Rlogn)) was suggested, where n is the number of points and R is the number of maximal-empty-rectangles present on the floorplan. Later the complexity was improved to $O(n\log^3 n)$ in

[5] and then to $O(n\log^2 n)$ in [6]. In [7], an algorithm is proposed for finding the maximum empty rectangle among a set of randomly distributed isothetic rectangles of arbitrary size and aspect ratio. The problem of locating isothetic empty orthoconvex polygons of required size amidst a set of obstacles, is also very important in VLSI layout design. An isothetic polygon is said to be **orthoconvex**, if any line parallel to X or Y axis and not overlapped with any of the bounding lines of the polygon, intersects the polygon either at exactly two points or at no point. In [8], several classes of such empty-orthoconvex-polygons (EOP) are defined among a set of points obstacles,and algorithms for locating such polygons of maximum area are discussed. The classes are (i) the "L" shaped EOP's, (ii) the " + " shaped EOP's, (iii) the EOP's with a nonempty kernel, (iv) the edge visible EOP's and (v) the line visible EOP's. The worst-case time complexities of these algorithms are $O(n^3)$, except for the maximum-area-edge- visible-polygon which is $O(n^2)$ [8].

In this dissertation, a new class of orthoconvex polygon, namely the **staircase polygon**, is introduced and an algorithm for finding an empty staircase polygon of maximum size is suggested. The algorithm is designed with the help of the concept of permutation graph. The worst-case time complexity of the algorithm is $O(n^3)$ and the space complexity is $O(n^2)$.

2: FORMULATION OF THE PROBLEM

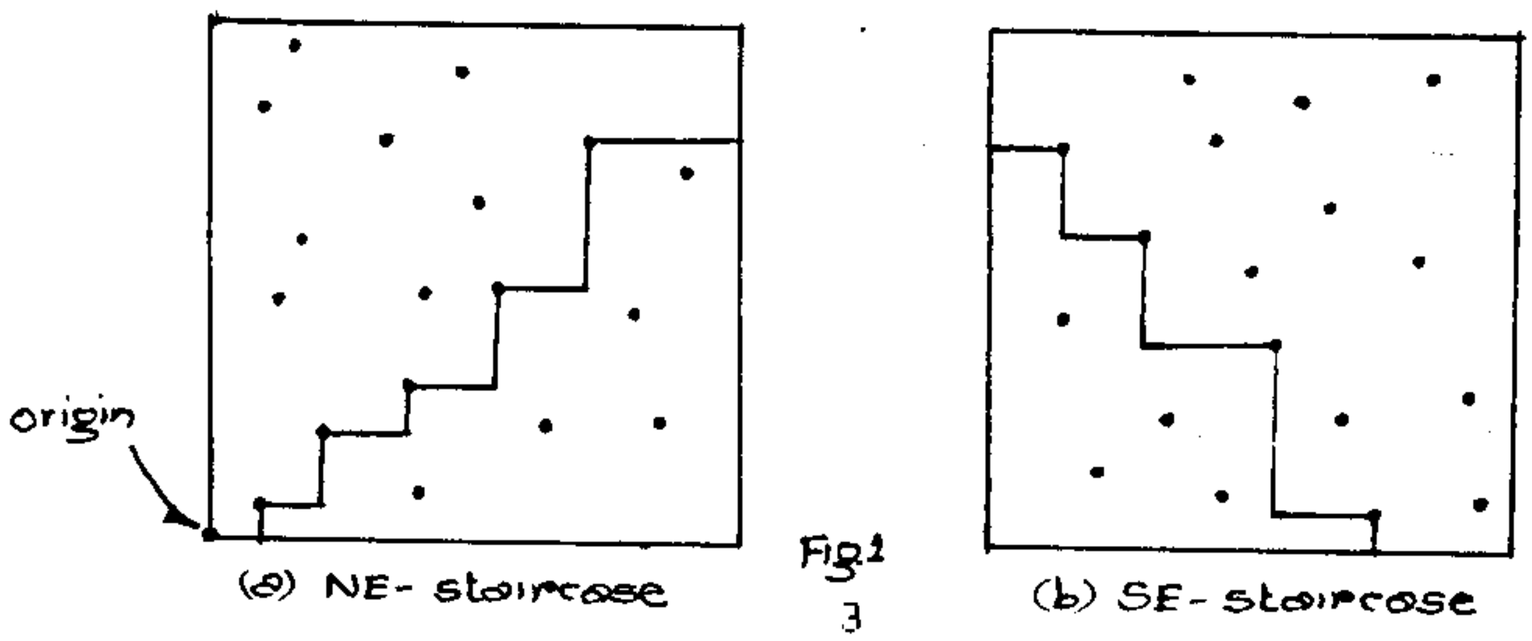
To illustrate the concept of an isothetic staircase polygon, let us define the following terms.

Definition : A simple polygon is said to be an isothetic polygon if all its edges are parallel to either X-axis or Y-axis.

Definition : An isothetic simple polygon is said to be an orthoconvex polygon, if any straight line drawn parallel to the X-axis or Y-axis and non-aligned to any of the boundaries of the polygon, either intersects exactly two sides of the polygon or does not intersect the polygon at all.

Definition : A non-crossed isothetic curve is a staircase if for all points p_i (x_i, y_i) and p_j (x_j, y_j) on the curve, where $x_i <= x_j$, either one has $y_i <= y_j$, or, $y_i >= y_j$. Let p be the top-most and q be the bottom-most points of a staircase.

If the staircase is monotonically increasing in the north east direction, it is called a north-east (NE) staircase. If the staircase is monotonically decreasing in the south east



(a) NE- staircase

Fig 3

(b) SE- staircase

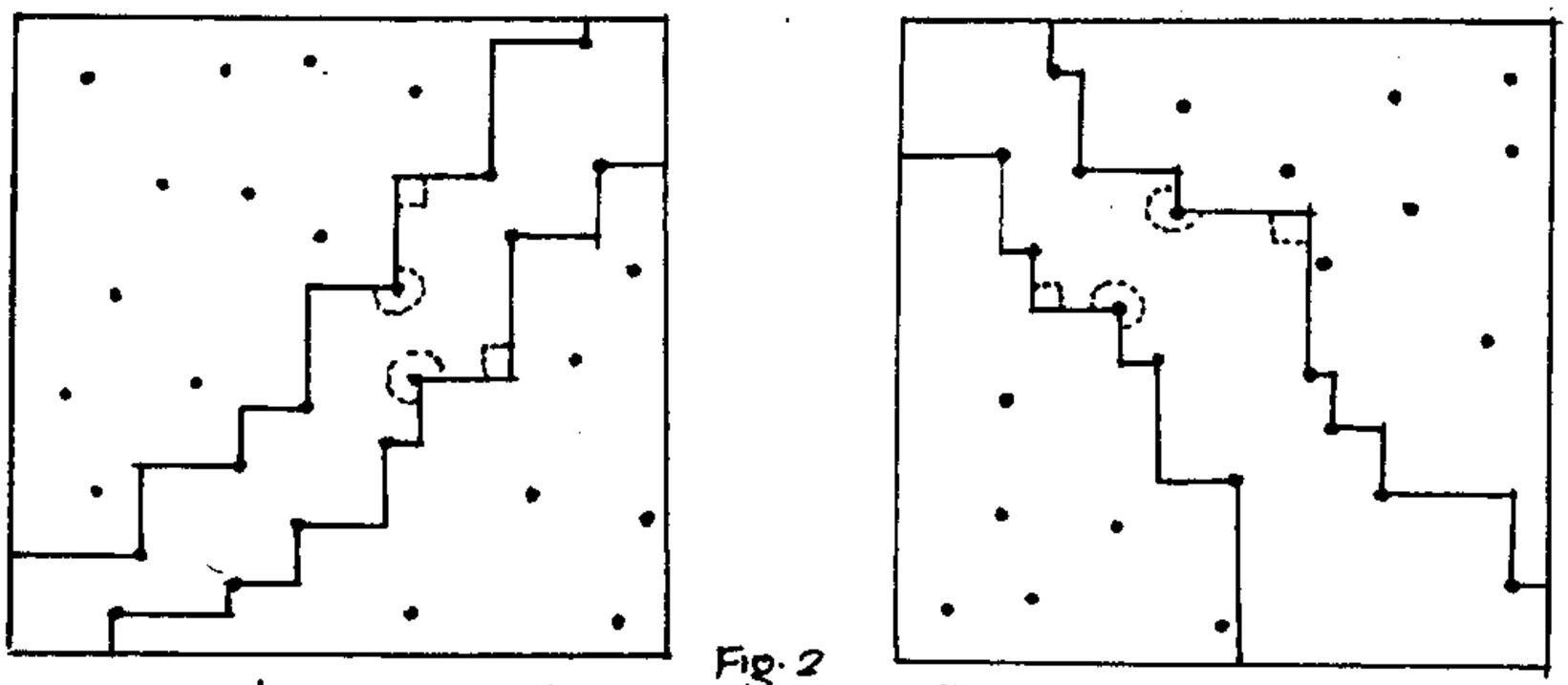
direction, it is called a south-east (SE) staircase. In Fig. 1 examples of north-east and south-east staircases are shown.

Let P be a set of points distributed on the floor. This set includes the top-right,top-left,bottom-right and bottom-left corners of the floor.

Definition : An empty staircase polygon in the floor is an isothetic polygon either bounded by two NE-staircases, or by two SE-staircases, such that no point in P lies inside the polygon.

An empty staircase polygon is said to be maximal if it cannot be embedded in a larger empty staircase polygon.

For maximal NE-staircase polygons, the topmost point of both the staircases is the top-right corner of the floor and bottommost point is the bottom-left corer of the floor. Similarly, for maximal SE-staircase polygons, the topmost point of both the staircases is the top-left corner of the floor and bottom most point is the bottom-right corner of the floor. Fig.2 shows examples of NE and SE staircase polygons.



(a) NE-staircase polygon

Fig. 2

(b) SE-staircase polygon

Fact 1 : The sides of a staircase polygon are alternately horizontal and vertical line segments. Two adjacent sides of a staircase meet at a corner point.

Definition : A corner point p of a polygon is said to be convex (concave) if the internal angle at p is 90^{o} (270^{o})

Observation 1 : In a maximal empty staircase polygon, all the concave corners coincide with a member in P.

The proof follows from Fig. 2.

Observation 2: For a maximal empty staircase polygon, if one staircase is fixed, the other staircase becomes unique.

The proof follows from Fig. 3.

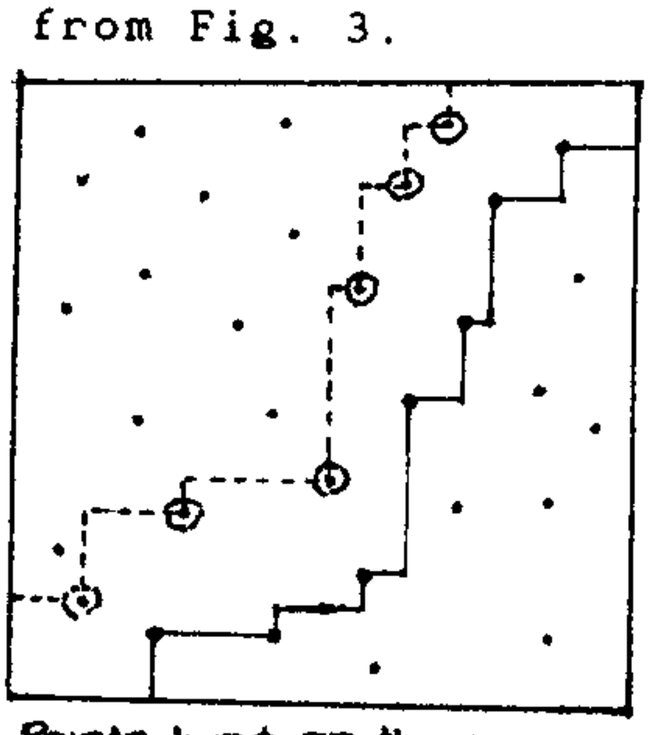


Fig 3. Points lying on the upper stair for the fixed lower stair, are encircled.

To find the the maximum area empty staircase polygon, we consider the NE-staircase polygons and the SE-staircase polygons separately. Here, we shall discuss the method of finding the maximum area NE-staircase empty polygon. The SE-staircase empty polygons can be located similarly. Let us

consider the NE-staircase polygons. On the basis of observation 2, we find an appropriate lower NE-staircase such that its corresponding polygon is empty and the area is maximum.

Henceforth we shall denote the x-coordinate of a point p P by p.x, and the y-coordinate by p.y.

Observation 3 : Consider a pair of concave corners p_i and p_j of the lower stair of an empty NE-staircase polygon. If

$p_i.x < p_j.x$ then $p_i.y \leq p_j.y$.

On the basis of observation 3, let us construct a graph G(V,E) where

V = { p | p P }

E = { (p_i,p_j) | p_i,p_j P and ($p_i.x <= p_j.x$) and ($p_i.y <= p_j.y$)}

It can be easily observed that the graph G is a permutation graph [10].

Observation 4 : Any path in the graph G corresponds to the lower NE-staircase of a maximal empty NE-staircase polygon.

In our algorithm, we shall select an appropriate path from the bottom-left corner to the top-right corner point in the permutation graph as the lower stair; the upper stair will be automatically decided as discussed in Observation 2.

Definition : An L-path from a point x to another point y is a manhattan path with at most one corner.

Definition : An L-Polygon(x,y) is a staircase polygon whose lower stair is an L-path from x to y, but the upper stair is a staircase path from x to y.Note that the upper stair has at least one corner. (see Fig. 4)

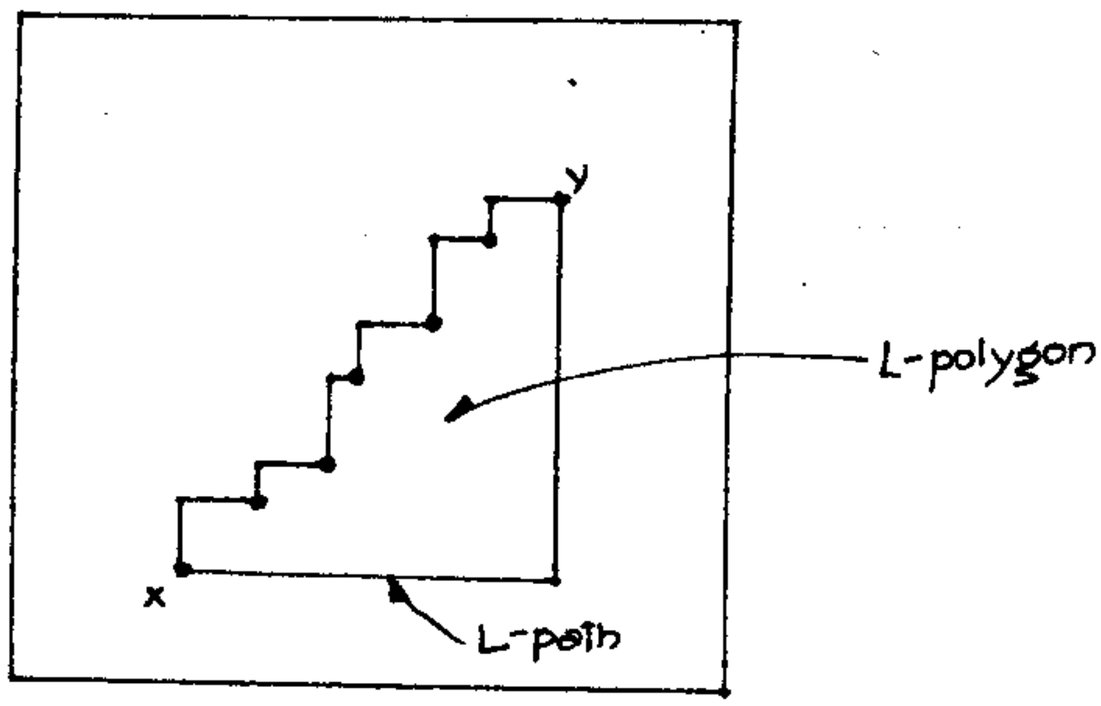


Fig. 4

Definition : A S-polygon(x,y) is an empty staircase polygon of maximum size from the bottom-left corner of the floor to the point y and whose lower stair has an L-path from point x to the point y. (see Fig. 5)

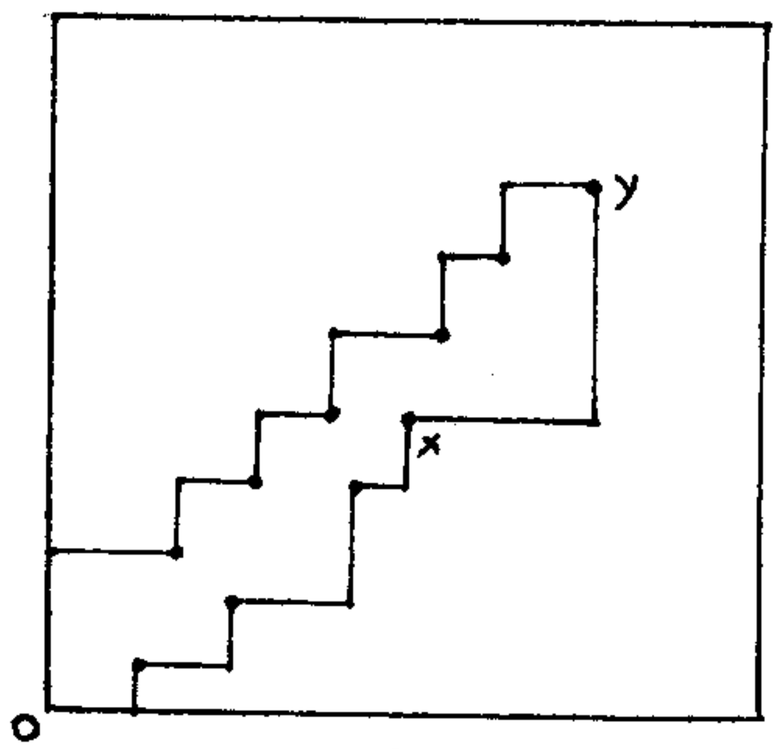


Fig. 5

Consider an edge of the permutation graph G from bottom-left corner O to a point x P on the floor as an L-path of the lower stair. It can be easily observed that

S-polygon(O,x) = L-polygon(O,x).

Definition : The merging line of a point x P is a horizontal line from left boundary of the floorplan to the point x.

Definition : Consider an edge (O-x) and the corresponding S-polygon(O,x). Let u^* be the point, where the topmost vertical edge of the upper stair meets the merging line of x.Then u^* is called the merging point of the S-polygon(O,x).

The area of the S-polygon(O,x) along with the point u^* is to be preserved along with point x for processing the outgoing arcs from point x.

Now consider the next L-path from the point x to the point y.

Observation 5 : The area of the S-polygon(x,y) = S-polygon(O,x) + L-polygon(u,v) + L-polygon(x,y). where u is the merging point of the S-polygon(O,x), and v is the point of intersection of the bottom-most horizontal edge of the upper stair of the L-polygon(x-y) with the vertical line at the point x.Fig. 6 demonstrates this fact.

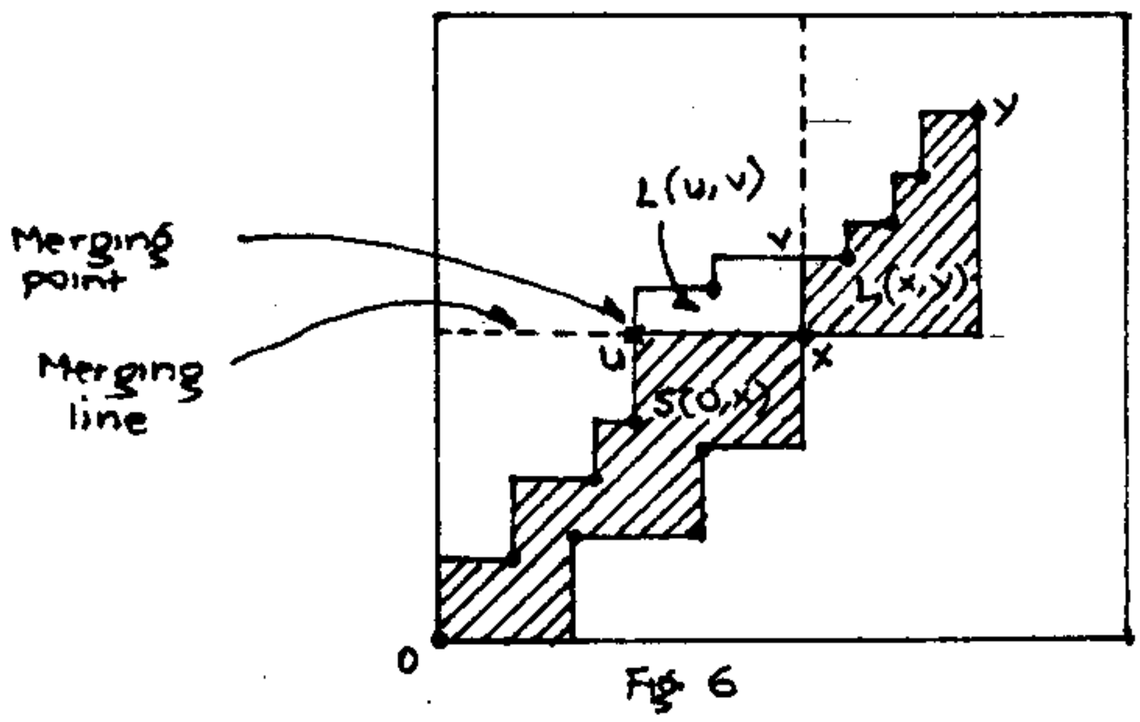


Fig 6

Next consider a vertex of G corresponding to a point y on the floor. Let x_1, x_2, ..., x_k be the points from each of which there is a directed edge incident on y. All these edges correspond to different S-polygons(x_i,y). Let u_i be the merging points of the S-polygon(x_i,y) and let A_i denote area(S-polygon(x_i,y)). Without any loss of generality, let us assume that $u_1 > u_2 > \ldots > u_k$. Consider the L-polygon(y,z) the bottom-most horizontal line of whose upper stair meets the vertical line at y at v.It is obvious that

$L(u_1,v) < L(u_2,v) < \cdots\cdots\cdots\cdots < L(u_k,v)$

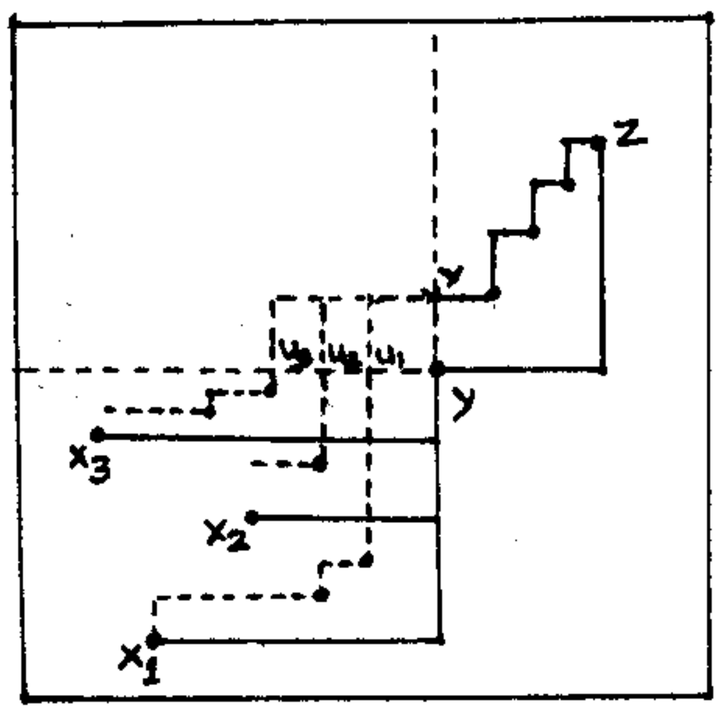


Fig. 7

Now if $A_i < A_j$ with $x_i > x_j$,

then $A_i + L(u_i,v) < A_j + L(u_j,v)$.

Thus the point u_i ($> u_j$) need not be considered further if $A_i < A_j$. Thus we conclude that the points $u_1, u_2, \ldots, u_k$ are to be retained with y only if $A_1 \rightarrow A_2 > \ldots > A_k$. (see Fig 7)

Let us now consider the method of finding the S-polygon(y,z) for an edge (y-z) in G. The point v, as stated above, is one, where the topmost vertical edge of the upper stair of the L-polygon(y,z) meets the vertical line at y and u_1, u_2, ..., u_k are the merging points of the S-polygons corresponding to y.One has to join one of the points (u_1, u_2, ...,u_k), say u_i, to v for the upper stair whose contribution to the area, as obtained from Observation 5, is maximum. If this area exceeds the area of the S-polygons incident on z,which have already been computed, or the new merge point is smaller than that corresponding to any S-polygon with a larger area, already computed, this (maximum) area, the point y and the merge point u_i are to be associated with the point z. Note that, if the upper stair of L-polygon(y,z) has only one vertical edge, more than one of the points {u_1, u_2,...,u_k} may have to be joined to v, leading to more than one S-polygon(y,z). All the S-polygons whose lower stairs pass through z, are found in a similar way. This process is repeated for all points in increasing order of their y-

coordinates until the upper right corner is reached. Next, a backward trace is executed to decide finally the lower stair from the top right corner to the bottom left corner. By observation 2, the upper stair is automatically decided by a line sweep. The theme of the algorithm and the data structure is described in the next section.

3: THEME OF THE ALGORITHM

3.1: Data structure

In this algorithm, two different data structures are to be maintained with the points in S.

(i) A priority search tree [9]

(ii) A list of points sorted in increasing order of y-coordinates. With each point,apart from it's coordinates, a pointer to the list of merge points is associated. Each merge point corresponds to an edge incident on x. If more than one edge incident to a point has the same merge point, the merge point will correspond to the edge which contributes the maximum area. The fields associated with each merge point are as follows.

(a) The x-coordinate of the merge point;

(b) Area of the S-polygon(x_i,x);

(c) The tail vertex x_i corresponding to the edge (x_i,x);

(d) The position of the merge point of the vertex x_i with

reference to which the area is calculated for the edge (x_i,x). Fields (c) and (d) are required for retracing the path.

Priority Search Tree

Let (U) be the universe of x-coordinates of the given point-set (P). Then a priority search tree for (P) is a search tree for (U) s.t. it consists of (i) a single node labelled by a unique element,if (P) contains only one point.
or (ii) A root,with fields priority and split .

The priority field contains the point (p) with minimum y-coordinate in the current set. The split field contains the x-coordinate of the vertical line that divides the set (U-p) into two equal halves,say (U_1) and (U_2).The left and right subtrees of the root are priority search trees for (U_1) and (U_2) respectively. Points lying on the separating line are put in a sorted sequence associated with the root.

It is needless to mention that in this priority search tree with the points P in the floorplan, the root corresponds to the bottom-left corner point.

In the graph G, each point (x,y) corresponds to a vertex. In the algorithm we shall use the terms vertex and point

interchangeably. The point (x,y) of G has outgoing edges to all the points $((x_i,y_i) \mid x \leq x_i$ and $y \leq y_i)$. Notice that we are not explicitly constructing the permutation graph. The edges of the graph are recognized during execution.

3.2: Method

The points are processed from bottom to top and if more than one point are in the same horizontal level, they are processed from left to right. A point is said to be completely processed if all its outgoing edges are processed. It is noted earlier that each edge in G corresponds to an L in the lower stair of a feasible staircase polygon. Thus when a point is selected for processing, it is completely processed before selecting the next point for processing. Consider an edge $e[P_1\text{-}p_2]$. The coordinates of the points p_1 and p_2 are (x_1,y_1) and (x_2,y_2) respectively. The processing of the edge e is done in the following way.

Let $u_1 < u_2 < \ldots < u_k$ be the merge points corresponding to the point p_1 with area $A_1 < A_2 < \ldots < A_k$. Consider a sweep line with horizontal span $I[u_1$ to $x_2]$ and set the vertical level $Y = y_1$. The sweep line starts an upward sweep from the vertical level Y. The priority search tree is searched to get the first point $p^*(x^*,y^*)$, such that $x^* \in I[u_1$ to $x_2]$ and the difference between y^* and y_1 is minimum. Here, three cases may arise.

Case 1: $y^* > y_2$ or no such point exists (Fig. 8).

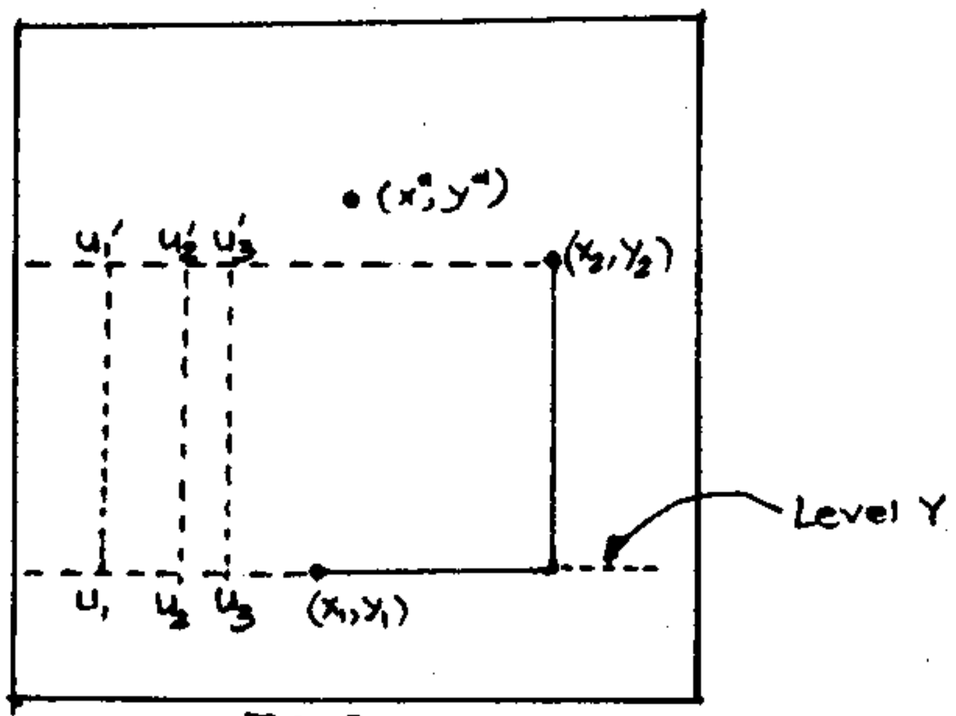


Fig. 8

Here, for each edge $e(p_1 - p_2)$, a list of merge points for the point p_2, is to be created from the merge points associated with p_1. The area corresponding to these merge points are to be calculated from the beginning of the merge list for p_1. The area corresponding to the merge point u_1 is a^* = (A_1 + Area of the rectangle $[(u_1,y_1), (x_2,y_2)]$. The merge point u_1' (= u_1) is inserted in the newly created merge list along with its associated fields as described in the data structure.

Let u_1', u_2', ..., $u_j'(j' < i)$ be the merge points stored in the newly created list at the time of processing u_i of the merge list corresponding to p_1. Now, $A^* = A_i$ + Area of the rectangle $[(u_i,y_1),(x_2,y_2)]$ is calculated and compared with A_j'. If $A^* > A_j'$, then u_i is added in the new merge list as u_{j+1}' otherwise u_i is ignored. This process is repeated for all merge points in the list of p_1. This newly formed merge list is now merged with the merge list associated with

p_2 maintaining the monotone increasing property with respect to area among the merge points associated to p_2.This completes processing of $e(p_1,p_2)$.

Case 2: $y^* < y_2$ and $u_{i-1} < x^* < u_i$ and

Case 3: $y^* < y_2$ and $x^* > u_k$ (Fig. 9).

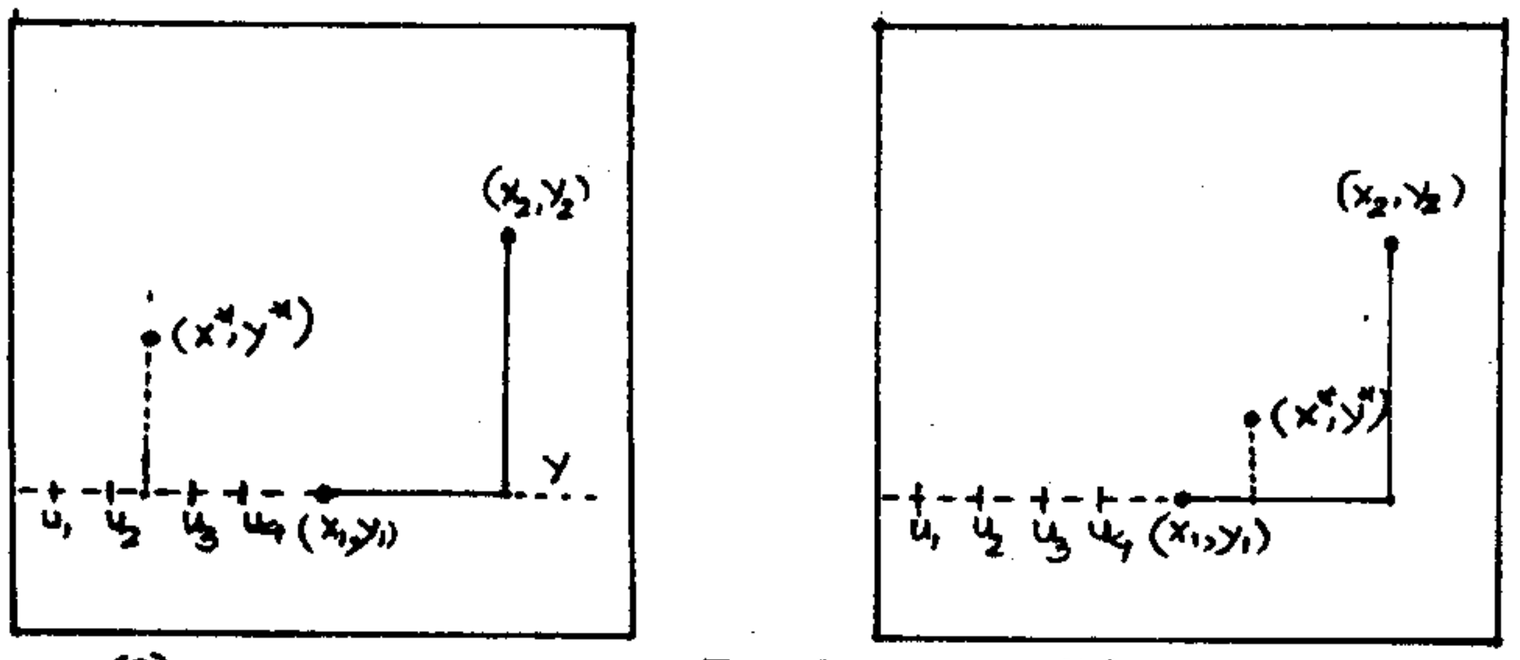


Fig. 9

For cases 2 and case 3, a new list of merge points is to be created for the edge $e[p_1-p_2]$ using the following procedure. Copy the list of merge points associated to p_1 in a temporary list. This list is to be updated to get the merge list for the edge e.

Consider case 2. In this situation calculate $A_j' = A_j +$ Area of the rectangle $[(u_j,y_1), (x^*,y^*)]$ for all $j = 1,2, \ldots, i-1$. Let A^* = maximum (A_1', A_2', ..., A_{i-1}') corresponding to the merge point u^* associated to p_1. Note that u^* is any one of $\{u_1, u_2, \ldots, u_{i-1}\}$. Delete $\{u_i, i=1, \ldots, i-1\}$ from the temporary list. Insert u^* at the begining of the list along with the associated fields. —

For case 3, all the merge points in the temporary list will be deleted after u^* is determined and u^* will be the only

merge point in the list.

This temporary list now becomes the merge list for $e(p_1,p_2)$. Reduce the horizontal span of the sweep line to $I[u^*$ to $x_2]$. The sweep is again initiated with horizontal span I from the vertical level Y.

In this way all the edges in G are processed. Note that at each vertex, only one tail vertex is stored corresponding to a particular merge point in the merge list. After complete processing of the top right corner point of the floor, a backtrack is done through the tail vertices to decide the lower stair.It is at this stage that the information about the merge point corresponding to the tail vertex is used to determine the next lower point on the lower stair.The upper stair is simultaneously decided by moving a sweep line downwards.

4: ALGORITHM MAX STAIR POLYGON

Input: A rectangular floor[a,b] with n point obstacles distributed on the floor(the pts include the four corner points of the floor as mentioned earlier.

Output:A maximum empty staircase polygon.

```
pair = record
 x,y:real;
end;

pt-&-merge-list =record
    x,y:real ;
    mglist:ptr  to record  merge_cell;
   link: ptr to record  pt-&-merge-list;
end;

merge_cell =record
    merge_pt:real;
    tail:pair;
    last_merge_pt:real;
    area:real;
end;
```

```
L,pi,pj:ptr to record pt-&-merge-list;
phigh,plow:pair; next_u:real;

Begin
  1:  Sort the n points into a list L such that for two
       points (xi,yi) and (xj,yj),
      (xi,yi)lies ahead of (xj,yj) in the list if (yi <  yj)
      or(yi=yj) and (xi < xj).
      This sorting corresponds to a topological ordering  of
      the underlying permutation graph.

  2:call FORM_PRIORITY_SEARCH_TREE(L);

  3:pi = L;
    while pi<>nil do
    pj=pi->link;
    while pj<>nil do
    if(pi->x <= pj->x) and
        (pi->y <= pj->y)then
        call FORM-S_POLYGON(pi,pj);
          pj = pj->link;
    enddo;
```

```
            p_i = p_i->link;
 enddo;

4:Scan the merge list of the top-right corner  point(a,b)
  to find  the  merge  point  for  which  the  area  is
  maximum.Let u be the merge point entry so determined.
  phigh =(a,b);
  plow =u->tail;
  next_u = u->last_merge_pt;
call FIND_RT_STAIR(plow,phigh,next_u);
 repeat
       u = next_u;
       phigh = plow;
       Scan  the  merge list of phigh to  find  the  merge
       point u.
       plow = u->tail;
       next_u = u->last_merge_point;
    call FIND_RT_STAIR(plow,phigh,next_u);
until plow = the bottom left corner point;

This determines the maximum NE-staircase polygon.

5:Rotate  the floor through 90^0 clockwise and tranform  the
 points accordingly so that the top left corner coincides with
 the bottom left corner.
 Repeat steps (1)-(4).

   This determines the SE-staircase polygon.
end;


ALGORITHM FORM_PRIORITY_SEARCH_TREE(L)

Input : A sorted list of points L;

Output :A priority  search tree for the points in L;

 treenode=record
   priority:pair;
   split:real;
   leftptr,rtptr:ptr to treenode;
   midptr:ptr  to  record
   pt-&-merge-pts;
 end;

p;pair;
med-x:real;

Begin
```

```
1: Let t-node be the tree node being processed.
   Search the list L to find the point with minimum y-
   coordinate.Let the point be p.
   Find the median of the x-coordinates of the remaining
   points.Let it be med-x.Let the two sublists obtained
   be  L1 and L2.

   t-node->priority=p;
   t-node->split =med-x;
   t-node->leftptr = call FORM_PRIORITY_SEARCH_TREE(L1);
   t-node->rtptr = call FORM_PRIORITY_SEARCH_TREE(L2);
   end;
```

ALGORITHM FIND_S_POLYGON(p_i,p_j)

Input:Two points p_i and p_j.

Output:The updated merge list of the point p_j
corresponding to the S-polygon(p_i,p_j)

```
 TMP_L:ptr to merge_cell;
 xright,xleft,ylow,yhigh:real;
 done:boolean;

 Begin
  1:Copy  the merge points and the associated fields into  a
    temporary list TMP_L;

 2:xleft = minimum merge_point for p_i;
   xright = p_j.x;
   ylow = p_i.y;
   yhigh = p_j.y;
   (x*,y*) = call MIN_Y_IN_XRANGE(xleft,xright,ylow);

 3:done = false;
 repeat
    if y* > yhigh or y* is undefined then
       for each merge point u in TMP-L do
             u->area = u->area + (xright - u->merge_pt)*
                                          (yhigh - ylow );
       enddo
       callMERGE(TMP-L,merge_list of point p_j);
         done = true;
    else
    for each merge point u_i in TMP_L such that
      u_i < x* do
            u_i->area = u_i->area + (xright - u_i)*
                                          (yhigh -   ylow);
    enddo;
```

```
Let u* be the merge point corresponding to which
the area is maximum.
Delete all merge points u_i < x* except u*
from TMP_L;
u* ->merge_pt = x*;
Place u* at the beginning of list TMP_L;
retain the merge points > x* if the associated
area is larger then the area associated with u*.
xright = x* ;
      endif;
   until done;
end;


min-y=record
   x,y:real;
   valid:boolean;
end;

ALGORITHM FIND_MIN_Y_IN_XRANGE(t_node,x1,x2,y):min-y;

Input:An x interval (x1,x2) and a horizontal line Y=y
          t_node is the tree node which is scanned.

Output:The point p if one exists,such that
           (i)x1 < p.x <x2;
           (ii)p.y > y
           (iii) (p.y -y) is minimum among all the points
                 satisfying conditions (i) and (ii)

cand1,cand2,cand3:min_y;


Begin
   1:if(xleft < t-node->priority.x < xright) and
    (t-node->priority.y > y) then
        cand1.x = t_node->priority.x;
        cand2.y = t_node->priority.y;
        cand2.valid = true;
   else
      if( xleft < t_node->split)then
          if t_node->leftptr <>nil then
              cand2=MIN_Y_IN_XRANGE(t-node->leftptr,x1,x2,y);
          endif
   endif
      if xleft = t-node->split then
      if t-node->midptr<>nil then
          cand3 = the first point in the median list
                  whose y-coordinate exceeds y;
```

```
    endif
endif
if t-node->split < xright then
    if t-node->rtptr <> nil then
          cand3 = FIND_MIN_Y_IN_XRANGE(t-node->rtptr,
                                        x1,x2,y);

     endif
endif
          if (not cand1.valid) or ((cand2.valid) and
             ((cand2.y < cand1.y) or (cand2.y = cand1.y) and
             (cand2.x > cand1.x))) then
                 cand1 = cand2;
           endif
          if (not cand1.valid) or ((cand3.valid) and
            ((cand3.y < cand1.y) or (cand3.y = cand1.y) and
            (cand3.x > cand1.x))) then
                cand1 = cand3;
          endif
            MIN_Y_IN_XRANGE = cand1;
      endif
   end;
```

```
ALGORITHM FIND_RT_STAIR(p1,p2,u)

Input:Points p1 and p2,and a merge point u corresponding to
      p1.

Output:The right stair corresponding to the L-polygon(p1,p2)

 xleft,xright,ylow,yhigh:real;
 done:boolean;

   Begin
    1:xleft = u;
      xright = p2.x;
      ylow = p1.y;
      yhigh = p2.y;
      (x*,y*) = call MIN_Y_IN_XRANGE(xleft,xright,ylow);

    2:done = false;
    repeat
       ify* > yhigh or y* is undefined then
            done = true;
       else
            Output(x*,y*);
            xright = x*;
       endif;
    until done;
end;
```

5: COMPLEXITY OF THE ALGORITHM

5.1: Time complexity

It is stated earlier that the construction of the priority search tree takes O(nlogn) time . The permutation graph representing the search domain is not physically constructed. At the time of processing a vertex, all its outgoing edges are considered. Corresponding to each such edge, the best L-polygon can be found in O(logn + a) time where a is the number of points lying on the upper stair of the L-polygon corresponding to that edge. Thus the total time complexity for the forward pass requires $O(|E|.\log n + \sum a_i)$ time, where E is the edge set of the underlying permutation graph. Let a^* be the maximum value of a_i's,then the time complexity of the algorithm will be $O(|E|.(\log n + a^*))$ which is output sensitive. It is also to be noted that |E| and a^* can be $O(n^2)$ and O(n) respectively in the worst case. So the worst case complexity of the algorithm is $O(n^3)$.

5.2: Space complexity

(i)The priority search tree takes O(n) space.

(ii)The merge-list associated with each point in the sorted list of points, may contain atmost n points.Since the merge list has to be retained till the backward scan determines the maximum stair,one concludes that the merge lists take $O(n^2)$ space. Thus the total space complexity is $O(n^2)$.

6: CONCLUSION

In this dissertation we have designed an algorithm for identifying an empty staircase polygon of maximum area, among a set of points distributed on a rectangular floor. The worst case time complexity of the algorithm is $O(n^3)$ and uses $O(n^2)$ space. A program based on this algorithm has been implemented in the C language on the VAX-8650 machine, and runs quite satisfactorily.

7: REFERENCES

1. Chang, J.S. and Yap, C.K., "A polynomial solution to potato-peeling and other polygon inclusion and enclosure problems", Proc. 25th IEEE Symp. on Foundation of Computer Science, 1984.

2. Boyce, J.E., Dobkin, D.P., Drysdale, R.L. and Guibas, L.J., "Finding extremal polygons", SIAM Journal on Computing, Vol. 14, 1985, pp. 134-147.

3. Aggarwal, A, Klawe, M., Moran, S., Shor, P. and Wilber, R., "Geometric applications of a matrix-searching algorithm", Algorithmica, Vol. 2, 1987, pp. 195-208.

4. Naamad, A., Lee, D.T. and Hsu, .L., "On the maximum empty rectangle problem", Discrete Applied Mathematics, Vol. 8, 1984, pp. 267-277.

5. Chazelle, B., Drysdale, R.L. and Lee, D.T., "Computing the largest empty rectangle", SIAM Journal of Computing, Vol. 15, 1986, pp. 300-315.

6. Aggarwal, A. and Suri, S., "Fast algorithm for computing the largest empty rectangle", Proc. 3rd Annual ACM Symposium on Computational Geometry", 1987, pp. 278-290.

7. Nandy, S.C., Bhattacharya, B.B. and Ray, S., "Efficient algorithms for identifying all maximal isothetic empty

rectangles in VLSI layout design", Proc. FSTTCS - 10, 1990, pp. 255-269.

8. Datta, A. and Ramkumar, G.D.S., "On some largest empty polygons in a point set", Proc. FSTTCS - 10, 1990, pp. 270-285.

9. Mccreight,E.M., "Priority search tree",SIAM J.Computt,Vol 14,1985,pp 257-276

10. Golumbic, M. C., Algorithmic Graph Theory and Perfect Graphs, Academic Press, 1980.

----------*----------