M.Tech.(Computer Science) Dissertation Series

Recognition of Largest Empty Orthoconvex Polygon in a Point Set

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

By

Humayun Kabir

Roll No : CS0606

Under the supervision of

Prof. Subhas C. Nandy

Indian Statistical Institute
203, B.T. Road
Kolkata-700108

INDIAN STATISTICAL INSTITUTE
203, B.T.Road
Kolkata - 700108

Certificate of Approval

This is to certify that the dissertation thesis titled "**Recognition of Largest Empty Orthoconvex Polygon in a Point Set**" submitted by Mr. Humayun Kabir, in partial fulfillment of the requirements for the M. Tech.(Computer Science) degree of the Indian Statistical Institute, Kolkata, embodies the work done under my supervision.

Prof. Subhas C. Nandy
Advanced Computing and Microelectronics Unit
Indian Statistical Institute, Kolkata - 700108

Acknowledgment

I thank my guide, **Prof. Subhas C. Nandy**, for his constant support, encouragement and many valuable suggestions, without which this report would not have been possible. I would also like to express my sincere gratitude to Dr. Gautam K. Das for his kind help.

I take this opportunity to thank my classmates for their support and help during my course work at ISI.

Humayun Kabir
M.Tech.(Computer Science),
Indian Statistical Institute, Kolkata - 700108

Contents

Chapter 1

Introduction

The objective of this report is to study the algorithm for computing the maximum area empty isothetic orthoconvex polygon ($MEOP$) among a set of n points on a rectangular region in $2D$. A polygon is said to be *isothetic* if its sides are parallel to coordinate axes. An isothetic polygon is said to be *orthoconvex* if the intersection of the polygon with a horizontal or a vertical line is a single line segment. Orthoconvexity has importance in robotic visibility, and VLSI. Datta and Ramkumar [1], proposed algorithms for recognizing largest empty orthoconvex polygon of some specified shapes among a point set in $2D$. These include (i) L-shape, (ii) cross-shape, (iii) point visible, and (iv) edge-visible polygons. The time complexity of these algorithms are all $O(n^2)$. Another variant in this class of problems is recognizing the largest empty staircase polygon among point and isothetic polygonal obstacles, which can also be solved in $O(n^2)$ time and space complexity [2]. But the problem of finding an maximum area orthoconvex polygon $MEOP$ of arbitrary shape is not studied yet. Here, we propose an algorithm to compute an $MEOP$ in $O(n^5)$ time and $O(n^3)$ space.

The thesis is organized as follows. In Chapter 2, we introduce some preliminary concepts and the overview of the algorithm. The algorithm for computing the *maximum area edge-visible* polygon is discussed in Chapter 3. The algorithm for finding the *maximum area empty staircase polygon* is discussed in Chapter 4. Finally the conclusion of the work appears in Chapter 5.

Chapter 2

Preliminaries

In this chapter we will give the algorithm to compute the $(MEOP)$. Before giving the algorithm we will first define some useful terms here.

Let $\mathcal{R}$ be a rectangular region in $2D$ containing a set of n points $P = \{p_1, p_2, \ldots, p_n\}$. The bottom left corner of $\mathcal{R}$ is assumed to be the origin, and the bottom and left boundaries of $\mathcal{R}$ are the $x-$axis and $y-$axis respectively. The coordinates of a point p are denoted as $(x(p), y(p))$. We assume that the points in P are in general positions, i.e., for any two points p_i and p_j, $x(p_i) \neq x(p_j)$ and $y(p_i) \neq y(p_j)$.

Definition 2.1 *A curve is said to be isothetic if it consists of horizontal and vertical line segments only.*

Definition 2.2 *An isothetic curve consisting of alternatively horizontal and vertical line segments is said to be a monotonically rising stair (R-stair) if for every pair of points* α *and* β *on the curve,* $x(\alpha) \leq x(\beta)$ *implies* $y(\alpha) \leq y(\beta)$.

Definition 2.3 *An isothetic curve consisting of alternatively horizontal and vertical line segments is said to be a monotonically falling stair (F-stair) if for every pair of points* α *and* β *on the curve,* $x(\alpha) \leq x(\beta)$ *implies* $y(\alpha) \geq y(\beta)$.

Definition 2.4 *A polygon is said to be* isothetic *if its sides are parallel to coordinate axes. An isothetic polygon is a region bounded by a closed isothetic curve.*

Definition 2.5 *An isothetic polygon* Π *is said to be* orthoconvex *if for any horizontal or vertical line* l*, the intersection of* Π *with* l *is a line segment of length greater than or equal to* 0.

An orthoconvex polygon is *empty* if it does not contain any point of P in its interior. Our objective is to identify the largest empty orthoconvex polygon in $\mathcal{R}$.

Definition 2.6 *An empty orthoconvex polygon* Π *is said to be maximal empty orthoconvex polygon* ($MEOP$) *if it does not contained in any other empty orthoconvex polygon* Π'.

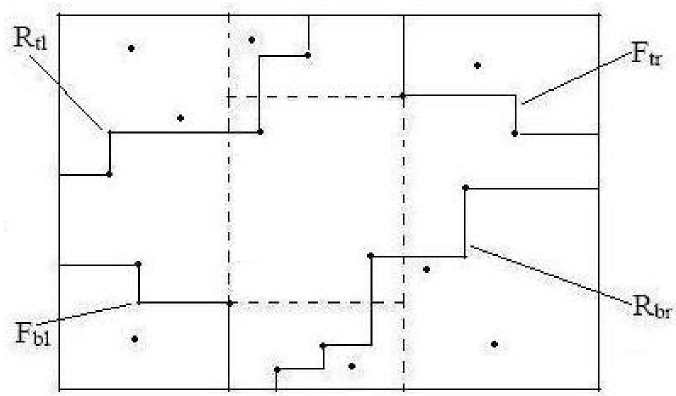

Figure 2.1: MEOP

The ($MEOP$) is bounded by two *R-stairs*, namely R_{tl} and R_{br}, and by two *F-stairs*, namely F_{tr} and F_{bl} (Figure 2.1). The rising stair R_{tl} spans from the left boundary to the top boundary of $\mathcal{R}$. The rising stair R_{br} spans from the bottom boundary to the right boundary of $\mathcal{R}$. The falling stair F_{tr} spans from the top boundary to the right boundary of $\mathcal{R}$. The falling stair F_{bl} spans from the left boundary to the bottom boundary of $\mathcal{R}$. Each concave vertex of these stairs must coincide with a point of P. Any of these stairs can become a degenerate stair and can coincide with a corner point of $\mathcal{R}$.

Definition 2.7 *Let* $\mathcal{R}'$ *be a rectangular region with* a *and* b *as its opposite corner points and let* $\mathcal{R}'$ *contains a point set* P' *and* $a, b \notin P'$. *A maximal empty staircase polygon* ($MESP(a, b)$) *among the points in* P' *is a* $MEOP$ *bounded by either two R-stairs or two F-stairs from from a to b depending on whether a and b are the bottom-left and top-right (resp. bottom-right and top-left) corner points of* $\mathcal{R}'$. *Its each concave corner of the stairs coincides with a point of* P'.

If the $(MESP(a,b))$ is bounded by two *R-stairs* then it is called a *R-staircase polygon* and if it is bounded by two *F-stairs* then it is called a *F-staircase polygon.*

Definition 2.8 *Let $\mathcal{R}'$ be a rectangular region containing a point set P' and a horizontal or a vertical line segment $[a,b]$ and $a,b \notin P'$. A maximal empty edge-visible polygon with the base $[a,b]$ among the points in P' is an $MEOP$ having an edge $[a,b]$ such that each point on its boundary is visible from the edge $[a,b]$. In such a polygon the edge farthest from $[a,b]$ coincides with the boundary of the region.*

We now present an algorithm for computing the maximum area $MEOP$.

2.1 Algorithm

Definition 2.9 *A point $p_i \in P$ is said to be the* bottom-pivot *of an MEOP if it lies on F_{bl} of that MEOP and it is the closest to the bottom boundary of $\mathcal{R}$ among all such points on F_{bl}. Similarly, a point p_j is said to be the* top-pivot *of an MEOP if it lies on F_{tr} of that MEOP and it is the closest to the top boundary of $\mathcal{R}$ among all such points on F_{tr}.*

We will consider each pair of points $p_i, p_j \in P$, and identify the maximum area $MEOP$ with p_i and p_j as the bottom-pivot and top-pivot respectively; the corresponding $MEOP$ is denoted by $MEOP(p_i, p_j)$. We will use H_i and V_i to denote a horizontal and vertical line passing through p_i. Let us denote by P_i (resp. P'_i) the set of points to the left (resp. right) of V_i. Let S denote the vertical slab bounded by V_i and V_j, and P_{ij} denote the set of points inside the vertical slab S. The projections of a point $p_k \in P_{ij}$ on V_i and V_j are denoted by q_k and q'_k respectively. The projections of $p_k \in P_{ij}$ on H_i and H_j are denoted by r_k and r'_k respectively. For a pair of points (p_i, p_j), the following three cases may produce an $MEOP$:
(i) $x(p_i) < x(p_j)$ and $y(p_i) < y(p_j)$,
(ii) $x(p_i) > x(p_j)$ and $y(p_i) < y(p_j)$, and
(iii) $x(p_i) < x(p_j)$ and $y(p_i) > y(p_j)$.

In Case *(i)*, the vertical lines V_i and V_j split the point set P into three parts, P_i, P_{ij} and P'_j (Figure 2.2). Let V_i hit the top and bottom boundaries of $\mathcal{R}$ at t_i and b_i respectively. Also let V_j hits the top and bottom boundaries of $\mathcal{R}$ at t_j and b_j respectively. Then the portion of the $MEOP$ inside the

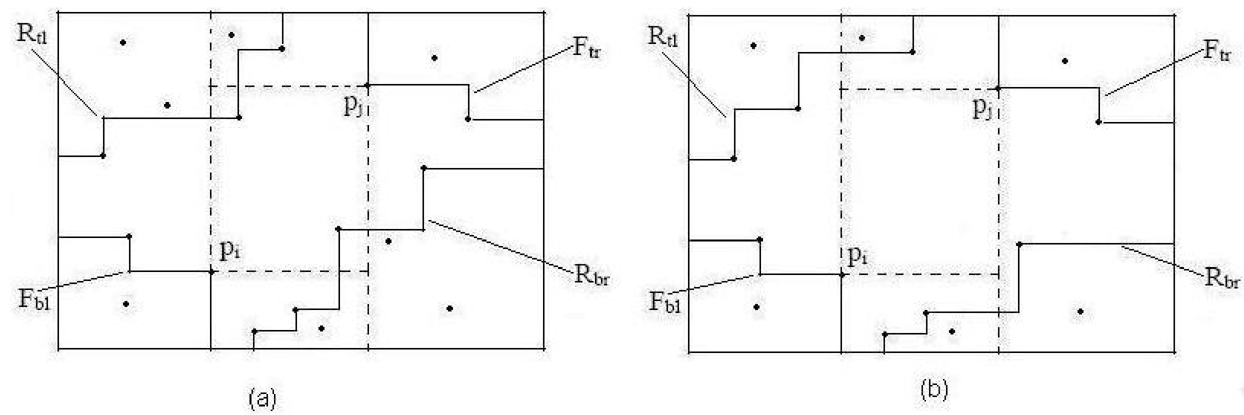

Figure 2.2: MEOP (Case (i))

vertical slab S is the $MESP(b_i, t_j)$ and we denote it by $M_2(p_i, p_j)$. The two stairs of $M_2(p_i, p_j)$ are parts of the rising stairs R_{br} and R_{tl} of $MEOP(p_i, p_j)$ respectively. If R_{tl} hits V_i at q_α (corresponding to a point $p_\alpha \in P_{ij}$), then the portion of the $MEOP$ to the left of V_i, denoted by $M_1(q_\alpha)$, is a maximal empty edge-visible polygon with base $[p_i, q_\alpha]$ among the points in P_i. Similarly, if R_{br} hits V_j at q'_β (corresponding to a point $p_\beta \in P_{ij}$) then the portion of $MEOP$ to the right of V_j is a maximal empty edge-visible polygon with base $[p_j, q'_\beta]$ among the points in P'_j, we denote it by $M_3(q'_\beta)$. Here two cases may arise: Case (i-a): the rectangle with p_i and p_j at its diagonally opposite corners is non-empty (Figure 2.2 (a)), and Case (i-b): the rectangle with p_i and p_j at its diagonally opposite corners is empty (Fig 2.2 (b)). The processing of Case (i-b) for computing $M_2(p_i, p_j)$ is slightly different from that of Case (i-a).

In Case (ii), V_j is to the left of V_i (Figure 2.3). Here the portion of $MEOP(p_i, p_j)$ inside the vertical slab S, $M_2(p_i, p_j)$ is equal to $MESP(p_i, p_j)$. The two stairs of $M_2(p_i, p_j)$ are the parts of falling stairs F_{bl} and F_{tr} of $MEOP(p_i, p_j)$ respectively. If F_{bl} hits V_j at q'_α (corresponding to a point $p_\alpha \in P_{ij}$), then the portion of $MEOP(p_i, p_j)$ to the left of V_j is an edge visible polygon with base $[q'_\alpha, t_j]$, among the points in P'_j (P'_j is the set of points to the left of V_j), we denote it by $M_1(q'_\alpha)$. If F_{tr} hits V_i at q_β (corresponding to a point $p_\beta \in P_{ij}$), then the portion of $MEOP(p_i, p_j)$ to the right of V_i is an edge visible polygon with base $[q_\beta, b_i]$ among the points in P_i (P_i is the set of points to the right of V_i), we denote it by $M_3(q_\beta)$.

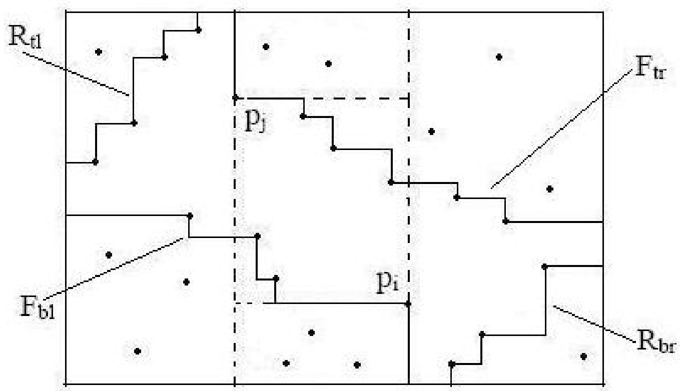

Figure 2.3: MEOP (Case (ii))

In Case (iii), here two cases may arise.

Case $(iiia)$: The rectangle with p_i and p_j at its diagonally opposite corners does not contain any point from P (Figure 2.4). Here the following two MEOPs' are generated depending on the change of role of these two points as p_i and p_j. In the former case, we calculate $M_2(p_i, p_j)$, $M_1(q_\alpha)$, and $M_3(q'_\beta)$ in the same way as that are calculated in Case (ib) (see Fig 2.4(a)). In the latter case, we rename p_i as p_j and p_j as p_i (see Figure 2.4(b)), and use Case (ii) to calculate $MEOP(p_i, p_j)$.

Case $(iiib)$: the rectangle with p_i and p_j at its diagonally opposite corners is non-empty, i.e., it contains some points from P inside it, then we rename p_i as p_j and p_j as p_i (Figure 2.5), and use Case (ii) to calculate $MEOP(p_i, p_j)$.

After fixing p_i as the bottom pivot, and p_j as the top pivot, we need to choose $M_2(p_i, p_j)$, $M_1(q_k)$, and $M_3(q_l)$ for some points p_k and p_l in P_{ij} such that the sum of areas of these three polygons is maximum among all such polygons. We will describe the algorithm for Case (ia) only. The case (ib) is the concatenation of two edge visible polygons, two *L_polygon*s and the rectangle with diagonally opposite corners p_i and p_j. Case (ii) and Case (iii) can be handled using a similar method. The method of computation for $M_1(.)$ and $M_3(.)$ are same. So, for Case (ia), only the methods of computing the desired $M_1(.)$ and $M_2(.,.)$ are explained in the subsequent chapters.

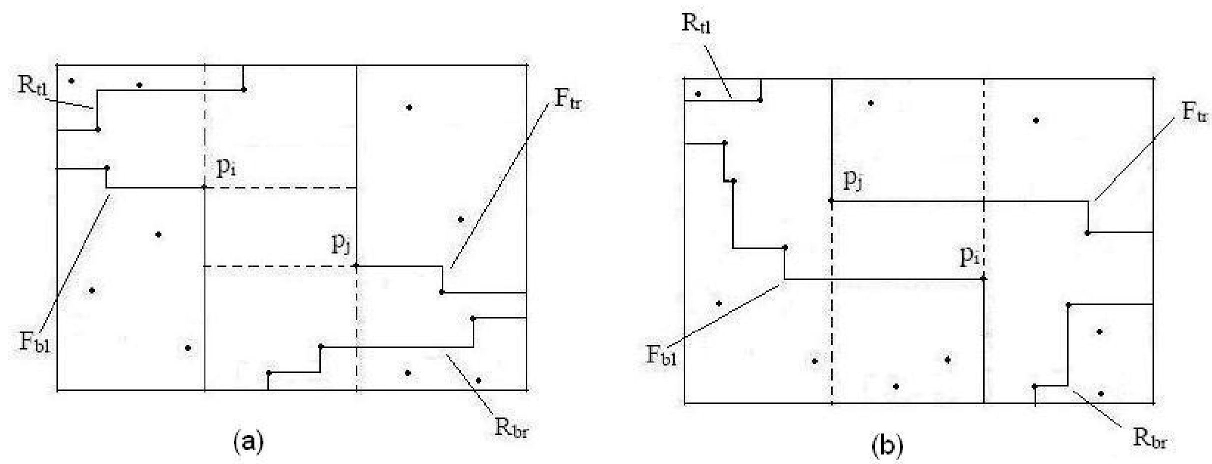

Figure 2.4: MEOP (Case (iiia))

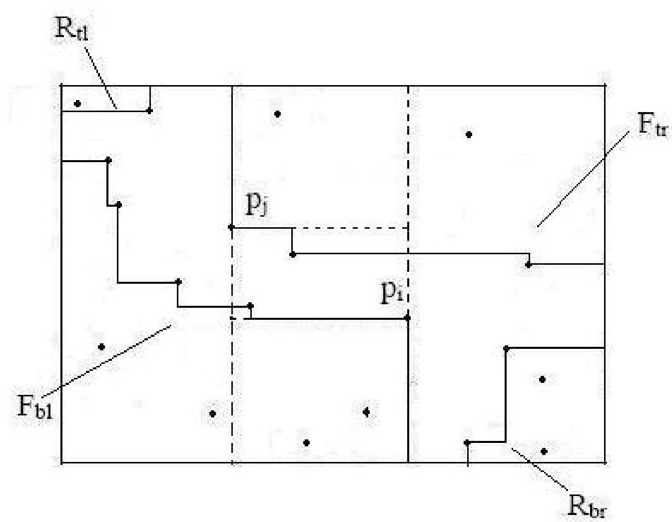

Figure 2.5: MEOP (Case (iiib))

Chapter 3

Computation of edge-visible polygon

In this chapter we will explain the method of calculating the maximum empty edge-visible polygon among points in Pi.

Let us consider a point $p_i \in P$. Let $Q = \{p_k | x(p_k) > x(p_i)$ and $y(p_k) > y(p_i)\}$. Q includes the top-right corner of $\mathcal{R}$, and $|Q| = m + 1$, i.e., Q contains $m + 1$ points. Let the projections of the points of Q on V_i are denoted by $q_0, q_1, \ldots, q_m$. We create an array $EVL(p_i)$ whose elements are the maximum empty edge visible polygons $M_1(q_k)$ with $[p_i, q_k]$ as the base, for all $k = 0, 1, 2, \ldots m$.

We use a vertical line sweep among the points in P_i starting from the position of V_i to create a binary tree $\mathcal{T}$ (Figure 3.1). Each node v of the tree is represented as a 4-tuple $(I, x_{val}, y_{val}, \Delta)$. I is the base of both the edge-visible polygons attached to v. (x_{val}, y_{val}) is the point where the node v is generated, and Δ contains the area of the edge-visible upto the node v, with base I of the root of the tree $\mathcal{T}$.

3.1 Creation of $\mathcal{T}$

For a point $p_k \in P_{ij}$, we compute the maximum empty edge visible polygon with base $[p_i, q_k]$ as follows.

The root r of the tree $\mathcal{T}$ corresponds to the interval $I = [p_i, q_k]$, its x_{val} is set to $x(p_i)$, y_{val} is set to 0, and Δ is also set to 0. A vertical line starts sweeping from $x = x(p_i)$ towards left. When it hits a point $p = (x(p), y(p)) \in$

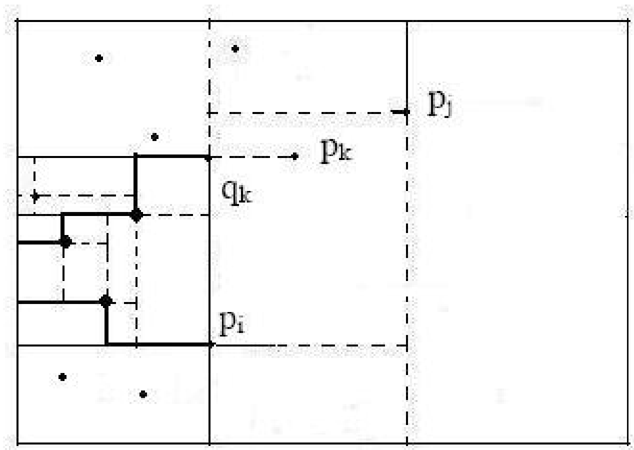

Figure 3.1: M1 computation

P_i, the leaf nodes in $\mathcal{T}$ are searched. If $y(p)$ lies in the interval $[i_1, i_2]$ of a node $v = ([i_1, i_2], x_{val}, y_{val}, \Delta)$, then we compute $\Delta' = \Delta + (x_{val} - x(p)) * (i_2 - i_1)$. Then we create two children of v, namely $v_{left} = ([i_1, y(p)], x(p), y(p), \Delta')$ and $v_{right} = ([y(p), i_2], x(p), y(p), \Delta')$. The sweep continues until the left boundary of $\mathcal{R}$ is reached.

We traverse the tree $\mathcal{T}$, and find the maximum value of Δ among the leaves in the tree $\mathcal{T}$, call it Δ_{mk}, then $M_1(q_k) = \Delta_{mk}$ is the maximum area empty edge-visible polygon with base $[p_i, q_k]$.

For each, $k = 0, 1, 2, \ldots m$, we calculate $M_1(q_k)$ and put it in the array $EVL(p_i)$. The algorithm to compute $M1_(q_k)$ is given below.

Algorithm 3.1 $M_1(q_k)$

1. Declare a structure, treeNode as,

```
typedef struct treenode
{
int I1;
int I2;
int xVal;
int yVal;
float Delta;
struct treenode *lChild;
```

*struct treenode *rChild;*
}treeNode;
2. Find the set, $P_1 = \{p_k | x(p_k) < x(p_i)\}$ *and* $nopP1 = |P_1|$ *and sort* P_1 *in decreasing x-coordinates.*
3. Create the root of the tree, where
root$\rightarrow$ *I1=*$y(p_i)$*;*
root$\rightarrow$*I2=*$y(q_k)$*;*
root$\rightarrow$*xVal=*$x(p_i)$*;*
root$\rightarrow$*yVal=0;*
root$\rightarrow$*lChild=NULL;*
root$\rightarrow$*rChild=NULL;*
4. For $l = 1, \ldots, nopP1$*, do for each* $p_l \in P_1$
search the leaves of the tree, if for some leaf v, $v(I1) < y(p_l) < v(I2)$*, then calculate*
$Delta' = v(Delta) + (v(xVal) - x(p_l)) * (v(I2) - v(I1));$
and create two chilren of v, where left child has I1 = v(I1),I2=$y(p_l)$*, xVal =* $x(p_l)$*, yVal =* $y(p_l)$*, and Delta = Delta′ and right child has I1 =* $y(p_l)$*,I2=v(I2), xVal =* $x(p_l)$*, yVal =* $y(p_l)$*, and Delta = Delta′*
5. Then traverse the tree to find the max-value of Delta among leaves and assign it to $M_1(q_k)$*.*

Lemma 3.1 *The computation of* $M_1(q_k)$ *can be done in* $O(n^2)$ *time.*

Proof 3.1 *The construction of the tree takes* $O(n^2)$ *and then searching for the maximum area node takes* $O(n)$*, in the worst case. So to compute* $M_1(q_k)$*, will take* $O(n^2)$ *time.*

Similarly, for each $p_j \in P$, we create an array $EVR(p_j)$. Let $Q' = \{p_k | x(p_k) < x(p_j) \text{ and } y(p_k) < y(p_j)\}$. Let $|Q'| = m'+1$, and let $q'_0, q'_1, q'_2, \ldots, q'_{m'}$ be the projections of the points of Q' on the vertical line V_j. Then $|EVR(p_j)| = m' + 1$, and the k-th element of $EVR(p_j)$ contains the largest empty edge-visible polygon $M_3(q'_k)$ with base $[p_j, q'_k]$ among the points in P'_j.

Lemma 3.1 states that, $EVL(p_i)$ and $EVR(p_j)$ for any $i, j = 1, 2, \ldots, n$, can be calculated in $O(n^3)$ time.

Chapter 4

Computation of M_2

For a pair of points $p_i, p_j \in P$, satisfying $x(p_i) < x(p_j)$ and $y(p_i) < y(p_j)$, we will explain in this chapter how to calculate the $MESP(b_i, t_j)$ among the points in P_{ij}, where b_i is the point where V_i hits the bottom boundary of $\mathcal{R}$, and t_j is the point where V_j hits the top boundary of $\mathcal{R}$, we call it $M_2(p_i, p_j)$.

First we will define two important terms.

Definition 4.1 *Let a and b be two points in 2D, with $x(a) < x(b)$ and $y(a) < y(b)$. Then an $L\_path(a, b)$ is a rectilinear path from the point a to the point b with exactly one corner at $(x(a), y(b))$. (Figure 4.1)*

Definition 4.2 *The largest empty staircase polygon whose upper stair is the $L\_path(a, b)$ and the lower stair is a rising stair from a to b is denoted as $L\_polygon[a, b]$. (Figure 4.1)*

Here we consider processing of a pair of points $p_i, p_j \in P$, satisfying $x(p_i) < x(p_j)$ and $y(p_i) < y(p_j)$. Let P'_{ij} be the set of points inside the rectangle with p_i and p_j as its diagonally opposite corners. Then $P'_{ij} \subseteq P_{ij}$. Let $|P'_{ij}| = m$. This $M_2(p_i, p_j)$ can be split into three parts: the *L_polygons* inside the slab S below H_i, the *L_polygons* inside the slab S above H_j, and the empty staircase polygon $MESP(p_i, p_j)$. Our objective is to choose the staircase polygon such that the sum of its area along with the area of the corresponding *L_polygons* in S and and the edge-visible polygons to the left of V_i and to the right of V_j is maximum.

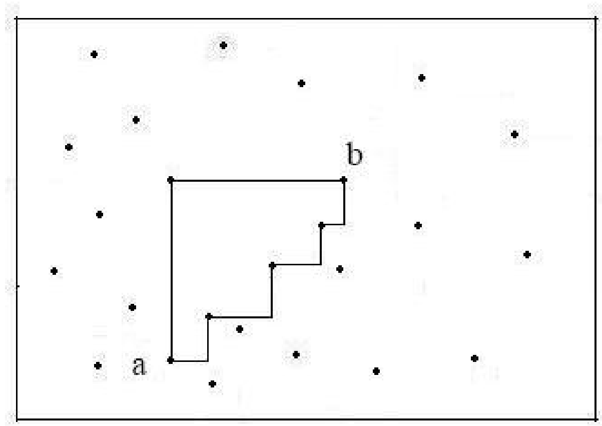

Figure 4.1: $L\_polygon[a, b]$

4.1 Computation of $L\_polygons$

We have $|P'_{ij}| = m$. Let $r_1, r_2, \ldots, r_m$ be the projections of the points in P'_{ij} on the horizontal line H_i in the increasing order of their x-coordinates. Let r_{m+1} be the intersection point of H_i and V_j.

The maximal empty *L_polygons* below H_i are calculated using a horizontal line sweep (Figure 4.2). The horizontal line sweep among the points in P_{ij} starts from the floor of $\mathcal{R}$ and ends at H_i, and computes the maximum empty *L_polygons* $LB(r_k)$ for $k = 1, 2, \ldots, m+1$. The upper stair of $LB(r_k)$ is an *L_path* with p_i at the corner of its *L_path*, and its lower stair is a rising staircase path from b_i to r_k.

Let r_0 be the intersection point of V_i and H_j. Let r_k for $k = 1, 2, \ldots, m$ be the projections of the points in P_{ij} on H_j in decreasing order of their x-coordinates. The maximal empty *L_polygons* above H_j, namely $LA(r_k)$, for $k = 0, 1, 2, \ldots, m$, are calculated using a horizontal line sweep starting from the top boundary of $\mathcal{R}$ up to H_j.

Lemma 4.1 *The computaion of LB and LA takes $O(n)$ time.*

Algorithm 4.1 *L_polygons LB*
1. Find the set $P_{LB} = \{p | y(p) < y(p_i)\}$ sort them in increasing x-coordinates and $nopPlb = |P_{LB}|$.
2. Take the projection of points in P'_{ij} on H_i and sort them in increasing

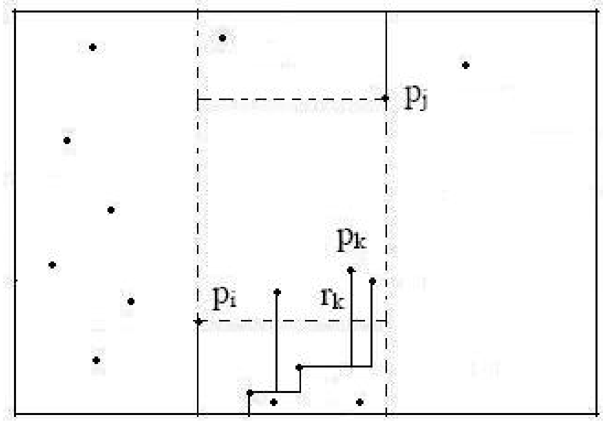

Figure 4.2: LB computation

x-cordinates. Let these be $r_k, k = 1, 2, \ldots, m+1$
3. k=1;
(a)For $l = 1, 2, \ldots, nopPlb$, $p_l \in P_{LB}$ *do*
if (l==1){
while $(x(r_k) < x(p_l))$
{
$LB(r_k) = (x(r_k) - x(p_i)) * y(p_l)$;
k=k+1;
}
$temp = (x(p_l) - x(p_i)) * y(p_l)$
lastused=l;
}
else {
if$(y(p_l) > y(p_{lastused})${
while $(x(r_k) < x(p_l))$
{
$temp1 = (x(r_k) - x(p_i)) * (y(r_k) - y(p_l))$;
$LB(r_k) = temp + temp1$;
k=k+1;
}
$temp1 = (x(p_l) - x(p_i)) * (y(p_l) - y(p_{lastused}))$;
$temp = (x(p_l) - x(p_i)) * y(p_l)$

lastused=l;
}
}
}

(b) while$(k \leq m+1)$
$temp1 = (x(r_k) - x(p_i)) * (y(r_k) - y(p_{lastused}))$;
$LB(r_k) = temp + temp1$;

4.2 Computation of $MESP(p_i, p_j)$

We now describe the last phase of our algorithm, where we compute the maximal empty staircase polygon $MESP(p_i, p_j)$ including the area of the corresponding *L_polygons* and the appropriate edge-visible polygons such that the total area of $MEOP(p_i, p_j)$ is maximum.

Let us first describe the method of computing $MESP(p_i, p_j)$ without considering the *L_polygons* and the edge-visible polygons. It will be bounded by two *R-stairs*, namely, the lower stair and the upper stair. Then we describe the changes needed in the procedure to calculate the $MEOP(p_i, p_j)$.

For any $MESP$, if the lower stair is fixed, the upper stair becomes unique. So to compute $MESP$, we have to find an appropriate lower stair such that the corresponding polygon is empty and its area is maximum, i.e., our task has boiled down to find an appropriate lower stair.

Let $G = (V, E)$ be a directed acyclic graph with vertices $V = \{p_k | p_k \in P''_{ij}\}$ where $P''_{ij} = P'_{ij} \cup \{p_i, p_j\}$. An edge $e_{kl} = (p_k, p_l)$ exists from p_k to p_l if $x(p_k) < x(p_l)$ and $y(p_k) < y(p_l)$. Thus, the edge set $E = \{e_{kl} = (p_k, p_l) | p_k, p_l \in P''_{ij}, x(p_k) < x(p_l) \text{ and } y(p_k) < y(p_l)\}$. The indegree of a node p_k is denoted by $in(p_k)$ and the outdegree is denoted by $out(p_k)$. In the graph G with P''_{ij}, we have $in(p_i) = 0$ and $out(p_j) = 0$.

Any directed path from p_i to p_j in G is called a *complete* path. Every complete path in G corresponds to the lower stair of a $MESP$.

The graph G for the points in Figure 4.3 (a) is shown in the Figure 4.3 (b). Also the $MESP$ in the Figure 4.3 (c) corresponds to the complete path $p_i \rightarrow p_1 \rightarrow p_2 \rightarrow p_j$ in the graph of Figure 4.3 (b).

The number of vertices in G, $|V|$ can be $O(n)$ in the worst case and the number of edges in G, $|E|$ can be $O(n^2)$ in the worst case.

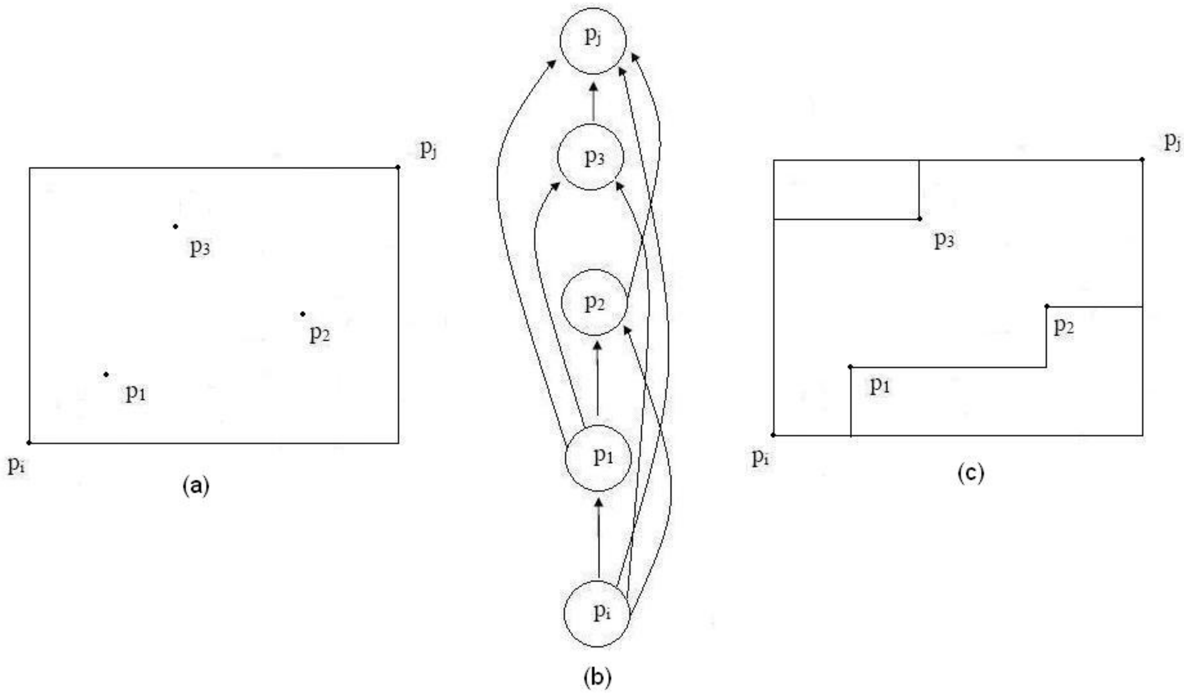

Figure 4.3: MESP

Among the complete paths we are to find the complete path which corresponds to the $MESP$ having maximum-area. So it is very natural to think that if we can assign weights to the edges of the graph G, then our job will be to find the maximum-weight complete path among all the complete paths in G. But we show that such an assignment of weight to the edges of G is not possible. In Figure 4.4, two lower stairs R_1 and R_2 are considered which pass through the points p_k and p_j. Considering R_1 in the lower stair, the weight of the edge (p_k, p_j) should be $Area(L\_polygon(k_1, p_j))$ and considering R_2 in the lower stair, the weight of the edge (p_k, p_j) should be $Area(L\_polygon(k_m, p_j))$. But in an weighted graph, an edge can not assume two different weights.

To overcome the above difficulty, we introduce the concept of $footprints$ obtained from the point in P'_{ij} and define a new weighted directed graph, called the *staircase* graph using the footprints as vertices. In this graph every edge between two nodes will correspond to a unique *L_polygon*, the

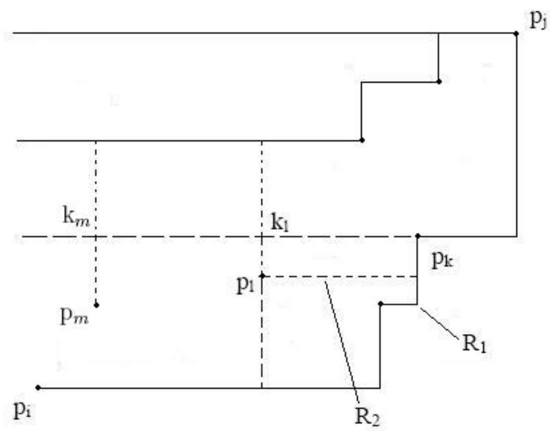

Figure 4.4: Motivation of footprints

weight of the edge will be the area of the *L_polygon* and staircase polygon is the concatenation of an appropriate set of *L_polygons*.

Definition 4.3 *Let p_l and p_k be two points in P''_{ij} such that $x(p_k) < x(p_l)$ and $y(p_k) < y(p_l)$, i.e., $(p_k, p_l) \in E$. Then the point $(x(p_k), y(p_l))$ is called the* footprint *of p_l contributed by p_k, and is denoted by l_k. The $footprint$ of p_i (the bottom-left corner point) is p_i itself.*

The set of footprints of a point p_l is denoted as $FP(p_l)$. The number of footprints of a point $p_l \in P''_{ij} \backslash \{p_i\}$ is $|FP(p_l)| = in(p_l)$, where $in(p_l)$ is the indegree of p_l.

The set of footprints for the example of Figure 4.3 (a), is shown in the Figure 4.6. Now we define the staircase graph below.

Definition 4.4 *The* staircase graph $SG = (V', E')$ *for a given digraph* $G = (V, E)$ *is a weighted digraph with nodes* $V' = \cup_{p_l \in P''_{ij}} FP(p_l) = \{$*the set of footprints of all the points in* $P''_{ij}\}$. *A footprint* $k_m \in FP(x_k)$ *has a directed edge to a footprint* $l_n \in FP(x_l)$, *if* $(p_k, p_l) \in E$ *(i.e.,* (p_k, p_l) *is an L_path), and the upper stair of the L_polygon*$[k_m, p_l]$ *meets the line* $Y = y(p_l)$ *at the footprint* l_n. *The weight of the edge* $(k_m, l_n) \in E'$, *denoted as* $w(k_m, l_n)$, *is equal to the area of the L_polygon*$[k_m, p_l]$.

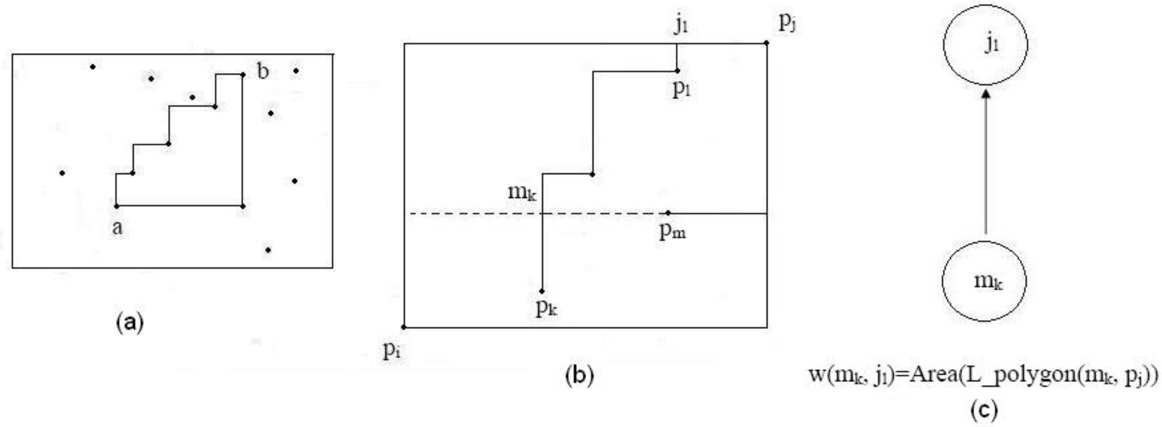

Figure 4.5: SG Edge

The graph SG is acyclic. In the Figure 4.5 (b), we have shown that there will be an edge between the footprints m_k and j_l, and the edge with its weight is shown in the Figure 4.5 (c). The staircase graph without the edge weights for the example of Figure 4.3 (a), is shown in the Figure 4.7.

Lemma 4.2 *The number of vertices in the graph SG, $|V'| = |E|$, and number of edges in the graph SG, $|E'| = O(n|E|)$.*

Proof 4.1 *Every footprint corresponds to an edge in G, so $|V'| = |E|$. The total number of outgoing edges of k_l in the graph SG is equal to $out(p_k)$. Again, the total number of footprints of a point p_k is equal to $in(p_k)$. Thus, the total number of edges $|E'| = \sum in(p_k) * out(p_k)$ which, in the worst case, is $O(n|E|)$.*

Every directed path from p_i to any footprint of p_j wil give a $MESP$. If k_m and l_n are on some path from p_i to some j_l ($\in FP(p_j)$), then the lower stair of the corresponding $MESP$ will pass through the points p_k and p_l and the upper stair of the $MESP$ will pass through the points p_m and p_n. So a directed path from p_i to a footprint of p_j in SG will determine both the upper stair and the lower stair of the corresponding $MESP$. The sum of edge-weights along the directed path will be the area of the corresponding $MESP$. The *maximum-weight* path is a path from p_i to some j_l ($\in FP(p_j)$)

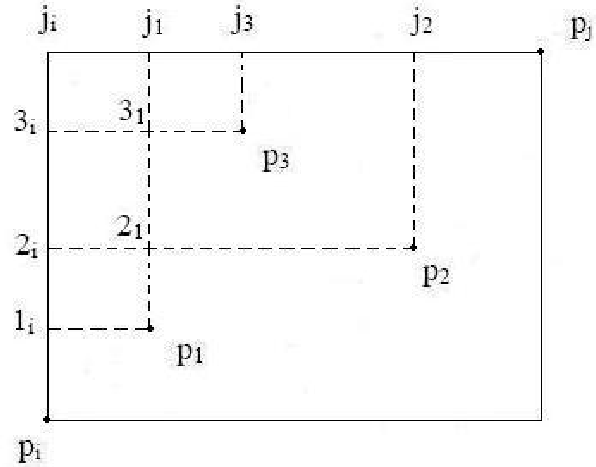

Figure 4.6: Footprints

whose weight is maximum among all such paths. The largest $MESP$ from p_i to p_j can be found by determining the max-weight path in the digraph SG.

Algorithm 4.2 *MESP*
1. Take the points in P''_{ij} *and give them order.*
2. Declare a structure to represent the nodes of the graph G, as
typedef struct linknode
{
int index;
*struct linknode *nextnode;*
}*nodeG;*
3. Construct the graph G and represent it using adjacency list representation.
4. Declare a structure to represent the nodes of the graph SG, as
typedef struct linknodefp
{
int fpOf;
int contributedBy;
float weight;
*struct linknodefp *nextnode1;*
}*nodeSG;*

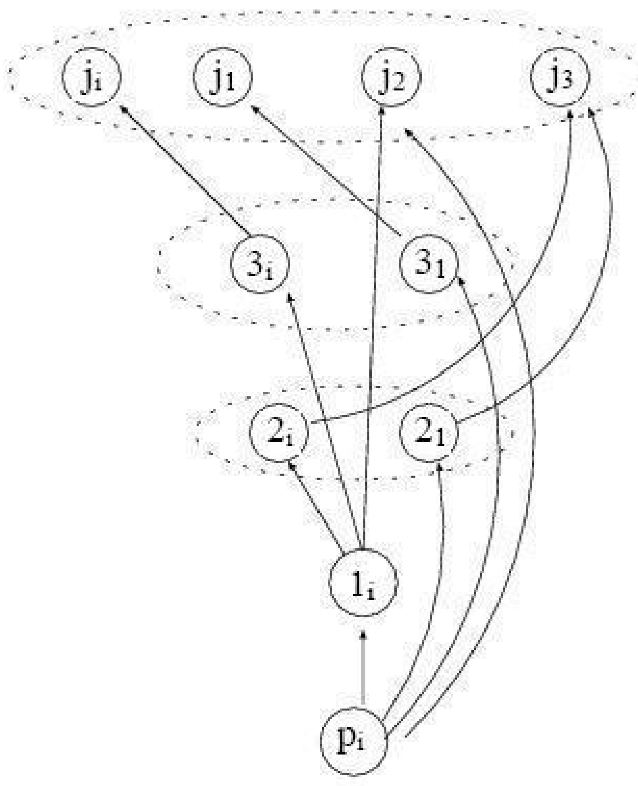

Figure 4.7: SG Graph

5. Traverse the graph G, to find the nodes of the graph SG.
6. Take the nodes of SG, and construct the graph SG and represent it using adjacency list representation.
7. Traverse SG, to find the max-weight path from p_i to a footprint of p_j.

For the present problem only computing the maximum area staircase polygon among the points is P''_{ij} will not be sufficient. Let $MEOP(p_i, p_j)$ contains a staircase polygon $MESP(p_i, p_j)$, that has an edge (p_i, q_α) along V_i then it includes an edge-visible polygon with base (p_i, q_α) to the left of V_i. Similarly if the $MESP$ has an edge (p_j, q'_α) along V_j then the $MEOP$ contains an edge-visible polygon with base (p_j, q'_α) to the right of V_j. Also if $MESP$ has an edge (p_i, r_β) along H_i then the $MEOP$ contains an *L_polygon* with base $[p_i, r_\beta]$ and if $MESP$ has an edge (p_j, r'_β) along H_j then the $MEOP$ contains an $L\_polygon$ with base $[p_j, r'_\beta]$. Thus in order to compute the $MEOP$ of maximum area, we need to modify the weight of some edges of the graph SG as follows and then compute the maximum weighted path in the graph SG.

For each foorprint q_α on V_i (q_α is the footprint of p_α contributed by p_i, i.e., $\alpha_i = q_\alpha$) of some point $p_\alpha \in P'_{ij}$, change the weight of its each outgoing edge $e \in E'$ to $w(e) + area(M_1(p_i, q_\alpha))$.

For each edge $e' = (p_i, \gamma) \in E'$, where γ is a footprint of some $p_k \in P'_{ij}$, then change the weight of e' to $w(e') + area(LB(p_i, r_k))$, where r_k is the projection of p_k on H_i.

For each $p_{\alpha'} \in P'_{ij}$, if there exists an edge e'' from a footprint of $p_{\alpha'}$ to a footprint of p_j, then change its weight to $w(e'') + area(M_3(p_j, q'_{\alpha'}))$, where $q'_{\alpha'}$ is the projection of $p_{\alpha'}$ on V_j.

For each incoming edge e' on $r_{\beta'}$, change the weight of e' to $w(e')+area(LA(p_j, r_{\beta'}))$, where $r_{\beta'}$ is the projection of some point $p_\beta \in P'_{ij}$ on H_j ($r_{\beta'}$ is also the footprint of p_j contributed by p_β, i.e., $j_\beta = r_{\beta'}$).

To find the $MEOP(p_i, p_j)$, we have to find the max-weight path in the modified digraph SG. The following theorem gives the required time complexity and space complexity to find $MEOP(p_i, p_j)$.

Theorem 4.1 *The $MEOP(p_i, p_j)$ can be find in $O(n^3)$ time using $O(|E'|)$ space.*

Proof 4.1 *The construction time of graph SG is $O(n|E|)$ and the max-weight path finding in the acyclic graph SG takes $O(|E'|) = O(n|E|)$ time. The time complexity to compute $MEOP(p_i, p_j)$ will be the maximum among the time complexities to calculate M_1, LB and the max-path in $MESP$. So the time complexity will be $O(n|E|)$, which may be $O(n^3)$ in the worst case. Again the space complexity will be $O(|E'|)$, because this is the maximum among the space complexities among the space complexities needed to calculate M_1, LB and the max-path in $MESP$.*

Theorem 4.2 *The largest $MEOP$ among a set of n points can be computed in $O(n^5)$ time using $O(n^3)$ space.*

Proof 4.2 *It follows from Theorem 4.1, and that we are considering every pair of points* p_i *and* p_j *and there are* $O(n^2)$ *such pairs.*

Chapter 5

Conclusion

We have given an algorithm for computing the largest empty orthoconvex polygon among a set of n points in $2D$. The algorithm is implemented in C language. Though its worst case time complexity is $O(n^5)$, it runs very fast.

Bibliography

[1] A. Datta and G. D. S. Ramkumar, *On some largest empty orthoconvex polygons in a point set*, Proc. FSTTCS, LNCS 472, pp. 270-285, 1990.

[2] S. C. Nandy and B. B. Bhattacharya, *On finding an empty staircse polygon of largest area (width) in a planar point-set* Computational Geometry: Theory and Applications, vol. 26, pp. 143-171, 2003.