

ON THE MAXIMUM EMPTY RECTANGLE PROBLEM*

A. NAAMAD, D.T. LEE

*Department of Electrical Engineering and Computer Science, Northwestern University,
Evanston, IL 60201, USA*

W.-L. HSU

*Department of Industrial Engineering and Management Sciences, Northwestern University,
Evanston, IL 60201, USA*

Received 18 February 1982

Revised 20 June 1983

Given a rectangle A and a set S of n points in A , we consider the problem, called the *maximum empty rectangle problem*, of finding a maximum area rectangle that is fully contained in A and does not contain any point of S in its interior. An $O(n^2)$ time algorithm is presented. Furthermore, it is shown that if the points of S are drawn randomly and independently from A , the problem can be solved in $O(n(\log n)^2)$ expected time.

1. Introduction

Given a rectilinearly oriented rectangle A in the cartesian plane and a set $S = \{P_1, P_2, \dots, P_n\}$ of $n > 1$ points in the interior of A , where each point P_i is specified by its X - and Y -coordinates (X_i, Y_i) , $i = 1, 2, \dots, n$ and A specified by its left boundary A_l , right boundary A_r , top boundary A_t and bottom boundary A_b , the *maximum empty rectangle* (MER) problem is to find a maximum area rectangle whose sides are parallel with those of A (and hence *similar* to A) and which is contained in A such that no point of S lies in its interior. This problem arises in situations where a rectangular shaped plant is to be located within a similar region which has a number of forbidden areas or in cutting out, for example, a 'perfect' rectangular piece from a large similarly shaped metal sheet with some defective spots. The problem could also be further modified so that the length and width of the sought-after rectangle have a certain ratio or have a certain minimum length. The special case in which a largest empty square is desired has been solved in $\theta(n \log n)$ time¹ using Voronoi diagrams in L_1 - (L_∞ -) metric [1-2] which is just a variation of the largest empty circle problem studied by Shamos [3]. We shall present in this paper

*Supported in part by the National Science Foundation under Grants MCS-7916847 and ECS-8105989.

¹ We say that $g(n) = O(f(n))$ if $|g(n)| \leq c f(n)$ for some constant $c > 0$ and all sufficiently large n , $g(n) = \Omega(f(n))$ if $g(n) \geq c f(n)$ for some constant $c > 0$ and all sufficiently large n and $g(n) = \theta(f(n))$ if $c f(n) \leq g(n) \leq c' f(n)$ for some $c, c' > 0$ and all sufficiently large n [6].

a $\theta(n^2)$ time algorithm for the MER problem and an $O(n \log^2 n)$ expected-time algorithm when the input points in S are placed inside A randomly and independently. The $O(n \log^2 n)$ expected-time algorithm can also be modified so that it runs in $O(n^2)$ time in the worst case.

2. Notation and preliminary results

In this section we introduce some notation and present a few preliminary results.

Definition. A (rectilinear) rectangle M is said to be a *restricted rectangle* (RR) if it satisfies the following conditions:

- (1) M is completely contained in A .
- (2) M contains no points of S in its interior.
- (3) Each edge of M either contains a point of S or coincides with an edge of A .

Obviously, the MER is restricted; so we can restrict our search for the MER to the RRs.

Lemma 1. *The number of RRs is bounded by $O(n^2)$, where n is the number of points in S .*

Proof. There are four possible types of RRs:

Type 1: RRs in which two opposite edges coincide with edges of A . There are $2n + 2$ rectangles of this type.

Type 2: RRs in which two adjacent edges coincide with edges of A . There are at most $2n + 2$ rectangles of this type (four of them are of type 1, too).

Type 3: RRs in which exactly one edge coincides with an edge of A . There are at most $4n$ rectangles of this type.

Type 4: RRs in which each edge contains a point of S in its interior. There are at most n^2 rectangles of this type.

The proofs for the numbers of RRs of types 1 and 2 are immediate. Let us consider type 3 RRs. Let D be an RR of type 3, with edges a , b , c and d such that a is the edge coincident with an edge of A and c the opposite edge of a . Let P_c be the point of S that c contains. It is obvious that once P_c and the edge with which a coincides are determined, D is determined. In other words, for every edge e of A , there are at most n RRs of type 3 in which one edge coincides with e . Therefore, the number of RRs of type 3 is bounded by $4n$.

As for type 4 RRs, let D be an RR of type 4 with edges a , b , c and d , such that:

- a contains the point $P_a = (X_a, Y_a)$,
- b contains the point $P_b = (X_b, Y_b)$,
- c contains the point $P_c = (X_c, Y_c)$ and
- d contains the point $P_d = (X_d, Y_d)$.

Without loss of generality, assume that:

1. $Y_a = \text{Max}\{Y_a, Y_b, Y_c, Y_d\}$
2. $Y_c = \text{Min}\{Y_a, Y_b, Y_c, Y_d\}$ and
3. $X_b < X_d$.

Under these assumptions, the following must hold:

1. $X_b = \text{Max}\{X_i \mid (X_i, Y_i) \in S, Y_c < Y_i < Y_a \text{ and } X_i > \text{Min}(X_a, X_c)\}$
for $i = 1, 2, \dots, n$.
2. $X_d = \text{Min}\{X_i \mid (X_i, Y_i) \in S, Y_c < Y_i < Y_a \text{ and } X_i > \text{Max}(X_a, X_c)\}$
for $i = 1, 2, \dots, n$.
3. There is no point $(X_i, Y_i) \in S$ such that:
 $X_b < X_i < X_d$ and $Y_c < Y_i < Y_a$.

This means that once the top and bottom coordinates (Y_a and Y_c , respectively) are determined, so are the left and right coordinates (X_b and X_d , respectively). There are at most $\binom{2}{2}$ different ways to pick the top and bottom coordinates. Therefore, the number of RRs of type 4 is bounded by $O(n^2)$. \square

Fig. 1 is an example of how the bound $O(n^2)$ is achieved. Each pair of points in $\{(X_i, Y_{i+1}), i = 1, 2, \dots, m-1\} \times \{(X_j, Y_{j-1}), j = m+2, \dots, n\}$ forms the left-top and right-bottom corner points of a different RR. When $m = \frac{1}{2}n$, there are $O(n^2)$ RRs of type 4.

Any algorithm that finds the MER by considering all possible RRs must take at

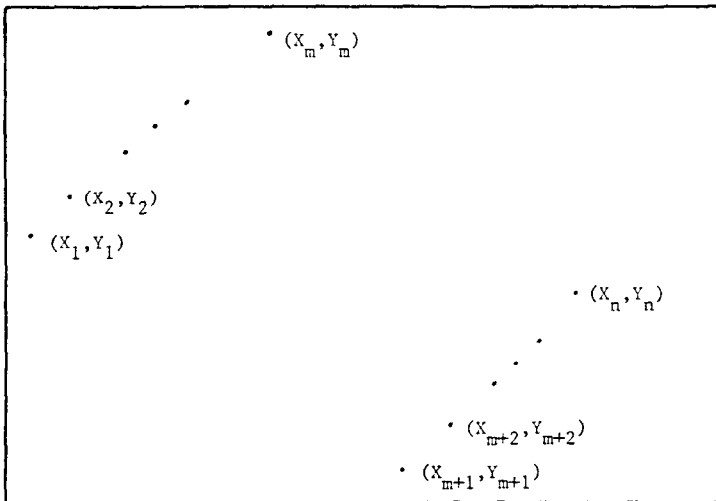


Fig. 1. A case of n points with $O(n^2)$ restricted rectangles.

least $\Omega(n^2)$ time in the worst case. Algorithm MERAlg 1 to be described finds the MER in $O(n^2)$ time by examining all possible RRs.

Although the number of RRs in the worst case is $\theta(n^2)$, we shall show in the following lemma that under certain probabilistic assumptions the average number of RRs is at most $O(n \log n)$.

Lemma 2. *Assume that the set $S = \{P_1, P_2, \dots, P_n\}$ of points are drawn randomly and independently from a rectangle A . Then the expected number of RRs is $O(n \log n)$.*

Proof. It suffices to show that the expected number of RRs of type 4 is $O(n \log n)$.

Let the corners of region A be denoted by $(0, 0)$, $(a, 0)$, $(0, b)$ and (a, b) . The assumption that P_1, \dots, P_n are drawn randomly and independently from A is equivalent to that X_1, \dots, X_n are uniformly distributed in $(0, a)$, Y_1, \dots, Y_n are uniformly distributed in $(0, b)$ and all X_i, Y_i , $i = 1, \dots, n$, are independent.

Given any point $P_i = (X_i, Y_i)$ of S , we compute the expected number of type 4 RRs whose top edges contain P_i in their interior. Any such rectangle must have

its right edge containing a point P_{i2} with $X_{i2} > X_i$, $Y_{i2} < Y_i$,

its left edge containing a point P_{i3} with $X_{i3} < X_i$, $Y_{i3} < Y_i$, and

its bottom edge containing a point P_{i4} with $X_{i3} < X_{i4} < X_{i2}$,

$Y_{i4} < \min\{Y_{i2}, Y_{i3}\}$. Furthermore, no point $P = (X, Y)$ of S can have $X_{i3} < X < X_{i2}$ and $Y_{i4} < Y < Y_i$. Based on these observations, we define the notion 'undominated' as follows. Consider all points of S that lie below P_i . A point $P = (X, Y)$ with $Y < Y_i$, is said to be *undominated* with respect to the point P_i if one of the following conditions holds:

(i) $X > X_i$ (we say the point lies to the *right* of P_i) and there is no other point $P' = (X', Y')$ such that $X_i < X' \leq X$ and $Y < Y' < Y_i$.

(ii) $X < X_i$ (we say the point lies to the *left* of P_i) and there is no other point $P' = (X', Y')$ such that $X \leq X' < X_i$ and $Y < Y' < Y_i$.

(iii) $X = X_i$, $Y < Y_i$ and there is no other point $P' = (X', Y')$ such that $X' = X = X_i$ and $Y < Y' < Y_i$. (However, the probability of having such a point is zero, hence we can concentrate on cases (i) and (ii).)

Denote by L_i the set of type 4 RRs whose top edges contain P_i in their interior. It is clear that every RR in L_i contains an undominated point on its bottom edge. Furthermore each undominated point (w.r.t. P_i) can be on the bottom edge of at most one RR in L_i . Hence $|L_i|$ is no greater than the number of undominated points. We claim that the latter has an expected value of $O(\log n)$.

Let the number of points below P_i be n' ($\leq n$). Let n_r be the number of points lying to the right of P_i . Then n_r has a binomial distribution with parameters $(n', 1 - X_i/a)$. Similarly the number of points to the left of P_i , $n_l = n' - n_r$, has a binomial distribution with parameters $(n', X_i/a)$. We now compute the expected number E_{n_r} of undominated points out of those n_r points. Sort these n_r points as M_1, \dots, M_{n_r} according to the Y -coordinates in descending order. Define random

variables C_i to be 1 if M_i is an undominated point and 0 otherwise. Then $E_{n_r} = \text{Exp}[C_1 + \dots + C_{n_r}] = \sum_{j=1}^{n_r} \text{Exp}[C_j]$. Now,

$$\begin{aligned} \text{Exp} &= \text{Pr}\{M_j \text{ is an undominated point}\} \\ &= \text{Pr}\{\text{The } X\text{-coordinate of } M_j \text{ is smaller than} \\ &\quad \text{that of } M_1, \text{ of } M_2, \dots \text{ and of } M_{j-1}\} = 1/j. \end{aligned}$$

Hence

$$E_{n_r} = \sum_{j=1}^{n_r} \text{Exp}[C_j] = \sum_{j=1}^{n_r} 1/j = O(\log n_r).$$

Similarly, the expected number E_{n_l} of undominated points to the left of P_i can be shown to be $O(\log n_l)$. Hence, given that there are n' points below P_i , the expected number of undominated points with respect to P_i is²

$$O \left[\sum_{n_r=0}^{n'} \binom{n'}{n_r} \left(1 - \frac{X_i}{a}\right)^{n_r} \left(\frac{X_i}{a}\right)^{n'-n_r} (\log n_r + \log(n' - n_r)) \right]$$

which is again bounded by

$$O \left[2 \log n' \sum_{n_r=0}^{n'} \binom{n'}{n_r} \left(1 - \frac{X_i}{a}\right)^{n_r} \left(\frac{X_i}{a}\right)^{n'-n_r} \right]$$

which is $O(\log n')$ and bounded by $O(\log n)$. Therefore the expected number of type 4 RRs is $O(n \log n)$. \square

3. The algorithms

We first present an algorithm which finds the maximum empty rectangle by considering all possible RRs. As shown in Lemma 1, the running time of any algorithm of this nature should be at least $\Omega(n^2)$. Hence the following algorithm is optimal in that sense.

Algorithm MERAlg 1 ($S, A_l, A_r, A_b, A_t, \text{MAXR}$)

Input: Four boundary values of a rectangle A : A_l, A_r, A_b and A_t (left, right, bottom and top), and a set $S = \{P_1, P_2, \dots, P_n\}$, $P_i = (X_i, Y_i)$ of points in A .

Output: MAXR, the area of the MER defined by S and A .

Method:

1. Let MGAP be the maximum gap in $\{A_l, A_r, X_1, X_2, \dots, X_n\}$.
2. $\text{MAXR} = \text{MGAP} * (A_t - A_b)$.
3. Sort S according to the Y coordinates of the points in descending order.
4. **For** $i = 1$ to n **do** steps 5-8.
5. $T_l = A_l, T_r = A_r$.

²For convenience, regard $\log n_r$ as zero when $n_r = 0$.

6. For $j = i + 1$ to n do step 7.
7. If $T_l < X_j < T_r$
 - Then do steps 7.1-7.2.
 - 7.1. $MAXR = \text{MAX}(MAXR, (T_r - T_l) * (Y_i - Y_j))$.
 - 7.2. If $X_j > X_i$
 - then $T_r = X_j$
 - else $T_l = X_j$.
8. $MAXR = \text{MAX}(MAXR, (T_r - T_l) * (Y_i - A_b))$.
9. For $i = 1$ to n do steps 10-12.
10. $R_i = \text{MIN}(A_r \cup \{X_j \mid (X_j, Y_j) \in S, Y_j > Y_i \text{ and } X_j > X_i\})$.
11. $L_i = \text{MAX}(A_l \cup \{X_j \mid (X_j, Y_j) \in S, Y_j > Y_i \text{ and } X_j < X_i\})$.
12. $MAXR = \text{MAX}(MAXR, (R_i - L_i) * (A_l - Y_i))$.

The correctness of Algorithm MERAlg 1 can be established by the following observations. Steps 1 and 2 consider RRs of type 1 whose top and bottom edges coincide with the top and bottom boundaries of A ; Step 8 considers RRs of type 3 whose bottom edges coincide with the bottom boundary of A ; steps 9 through 12 consider RRs of type 3 whose top edges coincide with the top boundary of A , and steps 4 through 7 consider the remaining RRs, i.e., RRs of type 1 whose left and right edges coincide with the left and right boundaries of A ; RRs of type 2 and type 4 and RRs of type 3 whose left or right edges coincide, respectively, with the left or right boundaries of A .

Algorithm MERAlg 1 always runs in $\theta(n^2)$ time irrespective of the actual number of RRs. As will be shown later, algorithm MERAlg 2 runs in time $O(s \log n)$, where s is the number of RRs. From Lemma 2, the expected number of RRs is $O(n \log n)$ if the points are drawn randomly and independently from the region A . It implies also that in step 7 of algorithm MERAlg 1, the condition $T_l < X_j < T_r$ holds for just $O(n \log n)$ times on the average, but MERAlg 1 checks it for $\theta(n^2)$ times. To avoid unnecessary checking for this condition, we need a mechanism to efficiently find the next point (X_j, Y_j) such that $T_l < X_j < T_r$. For this purpose a data structure T , called *semi-dynamic heap* (SDH) is adopted [4]. An SDH has the following properties: The points of S are sorted initially in the leaves of T by their X -coordinates. The leaves of T are at the same level. Note that in step 4 of MERAlg 1 the points are scanned in descending Y -coordinates. Initially all the points (leaves) are active. Each time when a point is scanned, it is 'deactivated' by a deletion operation (to be explained later). The SOL field of each internal node v of T contains the point (X, Y) whose Y -coordinate is the largest among the points that are currently in the active leaves of the subtree rooted at v . $SOL(v).X$ and $SOL(v).Y$ denote the X - and Y -coordinates of this point respectively. To guide subsequent searches, each node in T has other bookkeeping information, i.e., $LELEMENT(v)$ and $RELEMENT(v)$ which denote, respectively, the smallest and the largest X -coordinates of the active points stored in the leaves of the subtree rooted at v and $FATHER(v)$, $LSON(v)$ and $RSON(v)$ which denote v 's parent, left son and right son, respectively, in T . For

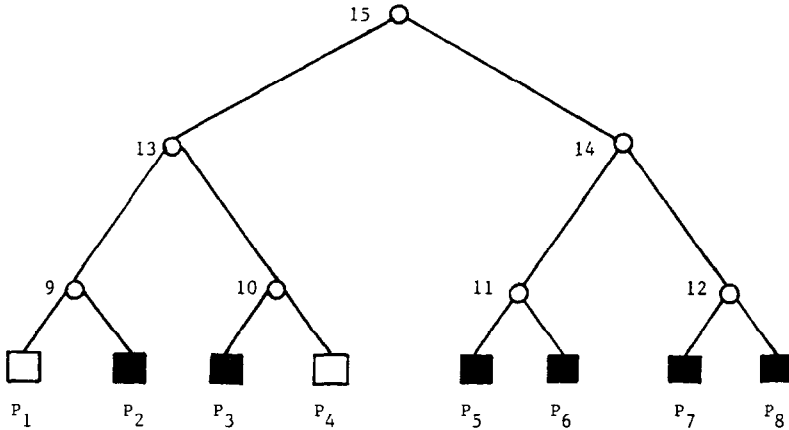


Fig. 2. Illustration of a semi-dynamic heap. $P_1=(1,5)$, $P_2=(2,7)$, $P_3=(3,2)$, $P_4=(4,8)$, $P_5=(5,4)$, $P_6=(6,1)$, $P_7=(7,9)$, $P_8=(8,3)$.

example, in Fig. 2, the darkened boxes are active leaves and SOL(13) will contain P_3 , LELEMENT(13) and RELEMENT(13) are 2 and 3 respectively.

It is clear how to construct T in $O(n)$ time after the points are sorted according to their X -coordinates. We shall perform two basic operations on T .

(i) Delete(P_x, P_y) deletes from T the point P whose coordinates (P_x, P_y) are stored in the SOL field of the root of T and returns (P_x, P_y) as output. We first locate the leaf node that contains P by its X -coordinate, 'deactivate' the leaf by setting the value of Y to A_b and update the values of SOL's along the path to the root. Obviously, this operation can be done in $O(\log n)$ time, since the height of T is bounded by $\lceil \log_2 n \rceil$.

(ii) Find(T_l, T_r, X, Y) finds the point (X, Y) in T that has the largest Y -coordinate among all the active points whose X -coordinates lie in the open interval (T_l, T_r) , by scanning all the vertices v in T that satisfy: (a) $T_l < \text{LELEMENT}(v)$ and $T_r > \text{RELEMENT}(v)$, and (b) $T_l \geq \text{LELEMENT}(\text{FATHER}(v))$ or $T_r \leq \text{RELEMENT}(\text{FATHER}(v))$. If there is no such active point in T , then Y is set to A_b and X to 0. There are at most $2\lceil \log n \rceil$ nodes in T that satisfy (a) and (b). Thus, the operation also takes $O(\log n)$ time. Let us first give detailed description of the above operations.

Procedure Find(T_l, T_r, X, Y)

1. $v = \text{root of } T; (X, Y) = (0, A_b)$.
2. **If** $T_l < \text{LELEMENT}(v)$ and $T_r > \text{RELEMENT}(v)$
then do steps 2.1-2.2.
 - 2.1. $(X, Y) = (\text{SOL}(v).X, \text{SOL}(v).Y)$.
 - 2.2. **Return.**
3. **If** v is a leaf
then return.

4. **If** $T_l \geq \text{RELEMENT}(\text{LSON}(v))$
Then do steps 4.1-4.2.
 {LSON(v) does not contain any active point whose X -coordinate lies in the interval (T_l, T_r) .}
 4.1. $v = \text{RSON}(v)$.
 4.2. **Goto** step 2.
 5. **If** $T_r \leq \text{LELEMENT}(\text{RSON}(v))$
then do steps 5.1-5.2.
 {RSON(v) does not contain any active point whose X -coordinate lies in the interval (T_l, T_r) .}
 5.1. $v = \text{LSON}(v)$.
 5.2. **Goto** step 2.
 6. {The points whose X -coordinates lie in the interval (T_l, T_r) are split between the subtrees rooted at LSON(v) and RSON(v).}
 $u = \text{LSON}(v)$; $w = \text{RSON}(v)$.
 7. {Scanning the subtree rooted at LSON(v), steps 7-9.}
If $T_l < \text{LELEMENT}(u)$
then do steps 7.1-7.2.
 *7.1. Switch(u). {The operation Switch will be explained later.}
 7.2. **Goto** step 10.
 8. **While** u is not a leaf **do** step 9.
 9. **If** $T_l < \text{LELEMENT}(\text{RSON}(u))$
then do steps 9.1-9.2.
 *9.1. Switch(RSON(u)).
 9.2. $u = \text{LSON}(u)$.
else $u = \text{RSON}(u)$.
 10. {We have finished scanning the subtree rooted at LSON(v), and start scanning the subtree rooted at RSON(v), steps 10-12.2).
If $T_r > \text{RELEMENT}(w)$
then do steps 10.1-10.2.
 *10.1 Switch(w).
 10.2. **Return**.
 11. **While** w is not a leaf **do** step 12.
 12. **If** $T_r > \text{RELEMENT}(\text{LSON}(w))$
then do steps 12.1-12.2.
 *12.1. Switch(LSON(w)).
 12.1. $w = \text{RSON}(w)$.
else $w = \text{LSON}(w)$.
- *Switch(s) stands for:
if SOL(s). $Y > Y$
then (X, Y) = (SOL(v). X , SOL(v). Y).

The correctness of procedure Find follows from the facts below.

(1) Each time we perform $\text{Switch}(s)$, $\text{LELEMENT}(s) > T_l$ and $\text{RELEMENT}(s) < T_r$.

(2) For each vertex s that satisfies $\text{LELEMENT}(s) > T_l$ and $\text{RELEMENT}(s) < T_r$, we visit it or one of its ancestors. This guarantees that we consider all the active points whose X -coordinates lie in (T_l, T_r) .

Steps 9, 10 and 11 of Algorithm MERAlg 1 take $O(n^2)$ time and can be modified to run in $O(n)$ time by using a more elaborate data structure. We maintain a doubly linked list LX of the X -coordinates of the points of S sorted in ascending order. A_l and A_r are at the beginning and end of the list. NEXT and BEFORE are pointers for the list. In the meantime, we have two arrays LY and PYX ; LY contains the Y -coordinates of the points of S sorted in ascending order and PYX contains a pointer to the list LX such that for every $i = 1, 2, \dots, n$, $(LX(PYX(i)), LY(i))$ are the X - and Y -coordinates of a point in S . For example, suppose the corner points of A are $(0, 0)$, $(0, 8)$ $(10, 0)$ and $(10, 8)$ and $S = \{(1, 5), (2, 4), (3, 7), (5, 1), (6, 2)\}$. The corresponding data structure is shown in Fig. 3.

Now we are ready to describe the algorithm MERAlg 2 which runs in $O(s \log n)$ time, where s is the number of RRs.

Algorithm MERAlg 2 ($S, A_l, A_r, A_b, A_t, \text{MAXR}$)

Input and output are the same as in Algorithm MERAlg 1.

Method:

1. Let MGAP be the maximum gap in $\{A_l, A_r, X_1, \dots, X_n\}$.
2. $\text{MAXR} = \text{MGAP} * (A_t - A_b)$.
3. Construct an SDH T as described above.
4. **Repeat** steps 5-7 n times.
5. $T_l = A_l$; $T_r = A_r$; $Y = A_t$.
6. Delete(P_x, P_y).

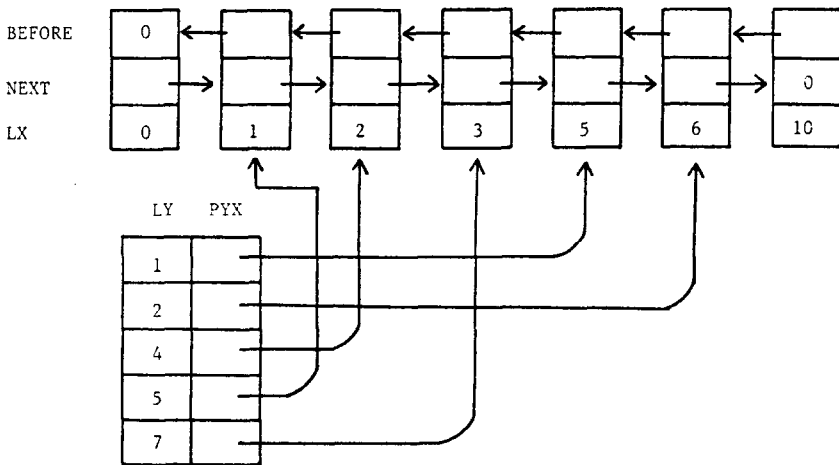


Fig. 3. Illustration of the list structures for LX and LY .

7. **While** $Y \neq A_b$ **do** steps 7.1–7.3.
 - 7.1. Find(T_l, T_r, X, Y).
 - 7.2. MAXR = MAX(MAXR, $(T_r - T_l) * (P_y - Y)$).
 - 7.3. **If** $X > P_x$
 - then** $T_r = X$
 - else** $T_l = X$.
8. {Step 8 of algorithm MERAlg 1 is embedded in step 7 when $Y = A_b$.} Construct $LX, NEXT, BEFORE, LY$ and PYX .
9. **For** $i = 1$ to n **do** steps 10–13.
10. $R_i = LX(NEXT(PYX(i)))$.
11. $L_i = LX(BEFORE(PYX(i)))$.
12. MAXR = MAX(MAXR, $(R_i - L_i) * (A_t - LY(i))$).
13. Delete $LX(PYX(i))$ from the doubly linked list.

It can easily be shown that except for step 7 all other operations take time at most $O(n \log n)$. Step 7 is executed for $\sum_{i=1}^n s_i$ times, where s_i is the number of Find operations for each point i . Since $s_i \geq 1$ and $\sum_{i=1}^n s_i = s$ is the total number of RRs, the running time of algorithm MERAlg 2 is $O(s \log n)$ and hence $O(n \log^2 n)$ on the average from Lemma 2.

In the worst case when the number s of RRs is of the order of n^2 , Algorithm MERAlg 2 runs in $O(n^2 \log n)$ time, which is worse than Algorithm MERAlg 1. The problem is that in step 7.1 for each RR, the algorithm MERAlg 2 takes $O(\log n)$ time, even if the point (X, Y) that Find operation returns is the ‘next’ point below (P_x, P_y) , the current point through which the top edge of the RR passes. To overcome this drawback we may scan the list sorted in descending Y -coordinate values as we did before in Algorithm MERAlg 1, except that when no point below (P_x, P_y) is found to lie in the interval (T_l, T_r) after $\log_2 n$ probes we invoke procedure Find. This guarantees us to obtain the next RR for consideration in time $\text{Min}(\log n, t)$, where $t \leq \log n$ is the number of probes before the next RR is found. In other words, if s_i is the number of RRs below P_i , the modified algorithm is assured to run for each P_i in $O(\min(s_i \log n, t_i))$ time, where $t_i = n - i$ is the number of points below P_i . Thus, the overall running time of the modified algorithm is $O(\min(s \log n, n^2))$.

4. Conclusion

In this paper, we present a $\theta(n^2)$ algorithm for finding the maximum empty rectangle in A that contains no point of S in its interior, where $|S| = n$. We also show that the expected number of RRs in A is $O(n \log n)$ when the points of S are drawn randomly and independently from within A . The original version of MERAlg 2 runs in time $O(s \log n)$, where s is the number of RRs. It can be modified to run in $O(\min(s \log n, n^2))$ so that the worst-case performance is still $O(n^2)$. It remains to be seen whether an algorithm of time complexity $O(s)$ can be devised. Furthermore,

whether or not the maximum empty rectangle can be found without considering all possible RRs is also an open problem.

References

- [1] D.T. Lee and C.K. Wong, Voronoi diagrams in L_1 (L_∞) metrics with 2-dimensional storage applications, *SIAM J. Comput.* 9(1) (Febr. 1980) 200-211.
- [2] F.K. Hwang, An $O(n \log n)$ algorithm for rectilinear minimal spanning trees, *J. ACM* 26(2) (April 1979) 177-182.
- [3] M.I. Shamos, *Computational Geometry*, Ph.D. dissertation, Dept. of Computer Science, Yale University, 1978.
- [4] A. Naamad, *Generalization of Heaps and Its Applications*, Ph.D. dissertation, Dept. of EE/CS, Northwestern University, Aug. 1981.