A unified algorithm for finding maximum and minimum object enclosing rectangles and cuboids

Article *in* Computers & Mathematics with Applications · April 1995

DOI: 10.1016/0898-1221(95)00029-X

Subhas C. Nandy
Indian Statistical Institute
113 PUBLICATIONS **534** CITATIONS
SEE PROFILE

Bhargab B. Bhattacharya
Indian Statistical Institute
314 PUBLICATIONS **1,535** CITATIONS
SEE PROFILE

Pergamon

Computers Math. Applic. Vol. 29, No. 8, pp. 45–61, 1995
Copyright©1995 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0898-1221/95 $9.50 + 0.00

0898-1221(95)00029-1

A Unified Algorithm for Finding Maximum and Minimum Object Enclosing Rectangles and Cuboids

S. C. NANDY AND B. B. BHATTACHARYA
Indian Statistical Institute, 203 B. T. Road
Calcutta - 700 035, India
bhargab@isical.ernet.in

(Received July 1994; accepted August 1994)

Abstract—Given a set of n points in $\mathbb{R}^2$ bounded within a rectangular floor F, and a rectangular plate P of specified size, we consider the following two problems: find an isothetic position of P such that it encloses (i) maximum and (ii) minimum number of points, keeping P totally contained within F. For both of these problems, a new algorithm based on interval tree data structure is presented, which runs in $O(n \log n)$ time and consumes $O(n)$ space. If polygonal objects of arbitrary size and shape are distributed in $\mathbb{R}^2$, the proposed algorithm can be tailored for locating the position of the plate to enclose maximum or minimum number of objects with the same time and space complexity. Finally, the algorithm is extended for identifying a cuboid, i.e., a rectangular parallelepiped that encloses maximum number of polyhedral objects in $\mathbb{R}^3$. Thus, the proposed technique serves as a unified paradigm for solving a general class of enclosure problems encountered in computational geometry and pattern recognition.

Keywords—Computational geometry, Range searching, Interval tree, Algorithm, Complexity.

1. INTRODUCTION

Range searching problems of computational geometry have manifold applications to operations research, database management, pattern recognition, robotics, VLSI layout design, to name a few. Given a set of points distributed randomly on a two-dimensional Euclidean plane ($\mathbb{R}^2$), various range queries are often asked. Efficient algorithms based on *range trees* or *Voronoi diagram*, can report the subset of points contained in a given rectangular region [1–3], or in a circle [4–6]. Chazelle *et al.* [4] and Lee [7] studied the problem of finding k-nearest neighbors in a set of n points in $\mathbb{R}^2$. Aggarwal *et al.* [8] proposed an algorithm for finding an isothetic rectangle that encloses maximum number of red points but no black point on a floor. A new approach to range searching called *filtering search*, was introduced by Chazelle [9] improving the time and space complexity of all the aforesaid problems.

A related range searching problem of finding the position of circular disc of *given radius* enclosing maximum number of points in $\mathbb{R}^2$ was considered by Chazelle and Lee [10], and an $O(n^2)$ algorithm was suggested. A similar problem was to find the *position of a given isothetic rectangle containing maximum number of points* for which an $O(n \log^2 n)$ time and $O(n \log n)$ space algorithm was reported [11]. Later, an $O(n \log n)$ time algorithm was proposed [12] for finding the maximum clique in the rectangle intersection graph, which can also be used for the above maximum enclosure problem in a plane. In a d-dimensional space ($d \geq 3$), finding the maximum clique in the hyper-rectangle intersection graph or equivalently, the max-enclosure problem, can be solved in $O(n^{d-1})$ time [13].

A slightly different form of range searching includes the classical maximal-empty-rectangle (MER) problem amidst a set of points [8,14–16]. Chazelle *et al.* [15] and Aggarwal *et al.* [8] reported the location of the largest isothetic MER in time $O(n \log^3 n)$ and $O(n \log^2 n)$, respectively. In an interactive VLSI environment, one often requires MER's among a set of solid isothetic rectangles or polygons instead of points [17]. More often than not, an empty rectangle of desired size and shape (aspect ratio) is not available. This leads to the problem of *locating a rectangle of given area and aspect ratio that encloses minimum number of points or other rectilinear obstructions.*

This paper outlines a unified algorithm based on the well-known line-sweep paradigm utilizing an interval tree as the underlying data structure. This is applicable to both the problems of locating (i) the maximum- and (ii) the minimum-point enclosing rectangle of given dimension, amidst a sea of n points in $\mathbb{R}^2$. The time and space complexities are $O(n \log n)$ and $O(n)$, respectively. The same algorithm works for finding the rectangle enclosing the maximum or minimum number of arbitrary polygons. Finally, we consider the problem in a 3-D scenario: to position a rectangular parallelepiped, i.e., a cuboid of given dimension, maximizing its containment in $\mathbb{R}^3$. This problem is of interest to computer graphics, and can be solved using our proposed plane-sweep technique.

The paper is organized as follows. In Section 2, the maximum- and minimum-point containment problems in $\mathbb{R}^2$ are formulated and an optimal algorithm is designed. Section 3 outlines the possible extensions of the above problems, where the points are replaced by rectangles or in general, by simple polygons. Location of a cuboid enclosing maximum number of points or objects in $\mathbb{R}^3$, is presented in Section 4. Concluding remarks appear in Section 5.

2. POINT-CONTAINMENT PROBLEM

Let n points be distributed on a rectangular floor F, whose top-left corner is at $(0,0)$ and the bottom-right corner is at (u,v). A rectangular plate P of size $(\alpha \times \beta)$ is given, where $\alpha (\leq u)$ is the length and $\beta (\leq v)$ is the width of P. We now consider the following two problems:

P1. (Max-enclosure): find an isothetic position of P enclosing maximum number of points in F.

P2. (Min-enclosure): find an isothetic position of P enclosing minimum number of points such that P is completely contained within F.

2.1. Formulation

The enclosure-problem stated above can be mapped to geometric intersection problems. For each point $p_i \in F$, an isothetic rectangle of size $(\alpha \times \beta)$ is drawn, such that its top-left corner coincides with p_i. Lemmas stated below now follow easily.

Lemma 1. *A set of isothetic rectangles satisfies Helly property* [12], *i.e., all the rectangles in the set have a common nonempty intersection if and only if every pair of rectangles are intersecting.* ∎

Lemma 2. *Consider k isothetic rectangles each of size $(\alpha \times \beta)$, and assume that they have a common non-empty intersection region Q, which itself will be of rectangular shape. Let the isothetic rectangle P of size $(\alpha \times \beta)$ be so placed that its bottom-right corner c_{br} lies in the region Q. Then P encloses the top-left corner of each of the k rectangles.* ∎

Remark. Lemmata 1 and 2 suggest that if n isothetic rectangles each of size $(\alpha \times \beta)$ are drawn as above corresponding to n given points, then problem **P1** reduces to locating $Q_{\max}$, the common intersection region of maximum number of rectangles. One solution of the maximum-enclosure problem can now be found by coinciding the corner c_{br} of P with the bottom-right corner of $Q_{\max}$. If P is positioned like this, no part of P will go outside of F. Moreover, for every position of c_{br}

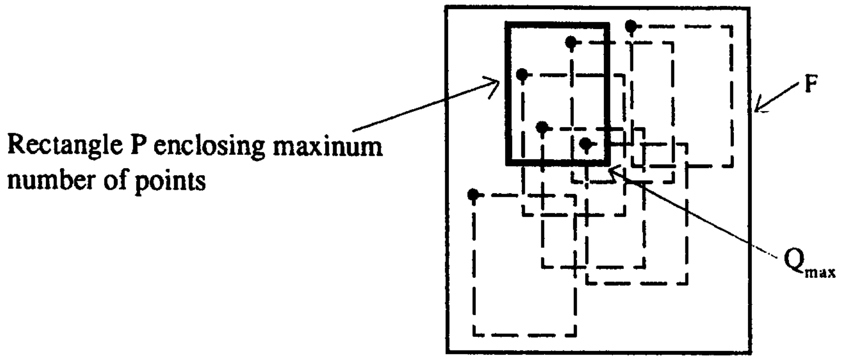


Figure 1. Proof of Lemma 1.

anywhere within $Q_{\max}$ for which the plate P in contained in F, maximum enclosure is guaranteed. This is illustrated in Figure 1.

These geometric intersection problems have the following graph-theoretic interpretation. The intersection graph of a family of isothetic rectangles is known as a *rectangle graph* $RG(V, E)$ [12]. Each node in V corresponds to a rectangle; two vertices are adjacent in RG if the rectangles representing them are intersecting.

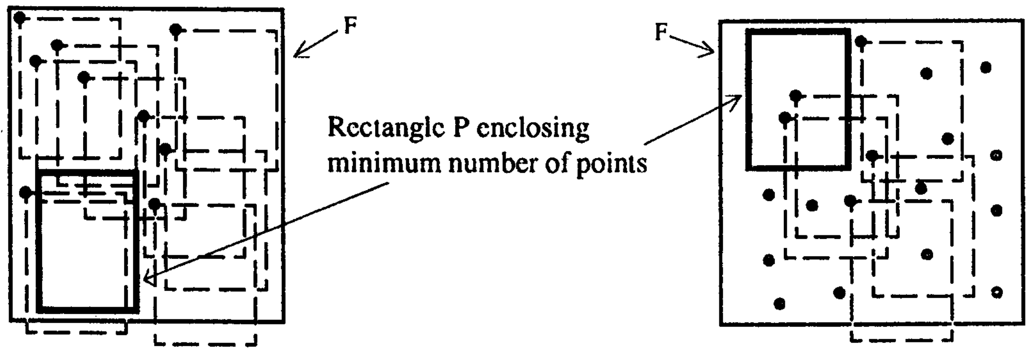
Let $V' \in V$ be a subset of nodes which constitute a maximum clique in RG. Let R' be the set of rectangles denoting the nodes in V'. If all rectangles are of equal size, then from Lemmas 1 and 2, it follows that

$$Q_{\max} = \bigcap\nolimits_{R_i \in R'} R_i.$$

Thus, problem **P1** reduces to identifying the maximum clique of RG, or equivalently, the common area $Q_{\max}$ formed by the elements of the maximum clique.

For the problem **P2**, $Q_{\min}$, which represents the common intersection region of minimum number of rectangles, sometimes plays a similar role, but in addition, one needs to consider other subtle factors. In this case, a position for the plate P within F may exist that does not enclose any point at all. Furthermore, one should also ensure that a rectangle of size $(\alpha \times \beta)$ whose bottom-right corner lies within $Q_{\min}$, is *contained completely* within the floor F.

The fact that differentiates problem **P2** from **P1** is that $Q_{\min}$ may not correspond to a minimum clique of RG. Figure 2 demonstrates that the desired position of P is not obtainable from the intersection region corresponding to the minimum clique.



(a) Minimum clique is of size 3 whereas the solution rectangle is empty.

(b) Minimum clique is of size 3 but the solution rectangle encloses two points.

Figure 2.

An $O(n \log n)$ algorithm for finding the maximum clique in a rectangle graph is reported earlier [12], which can be used to solve the max-enclosure problem (**P1**). However, this algorithm cannot readily be applied for solving the minimum-point enclosing problem (**P2**). Here, we propose an alternative and easy-to-implement algorithm based on *interval tree* data structure [18]. This solves both problems in $O(n \log n)$ time and $O(n)$ space. Next, the proposed algorithm is extended to tackle the general enclosure problems, where polygonal objects are given instead of points, or in a three-dimensional environment.

2.2. Bounds on the Number of Cliques

The result stated below gives an upper bound on the number of cliques in a rectangle graph.

THEOREM 1. *The total number of cliques in $RG(V, E)$ can be at most $O(n^2)$, where n is the number of vertices in the graph.*

PROOF. Project the horizontal span of each rectangle on the X-axis. This creates a set of n intervals. Consider the interval graph [19] corresponding to the above family of intervals. The total number of (maximal) cliques in an interval graph is at most $O(n)$. Similarly, the total number of cliques in the interval graph formed by projecting vertical spans of the rectangles on the Y-axis is $O(n)$. Two rectangles intersect if and only if both their X- and Y-projections intersect. Thus, there could be at most $O(n^2)$ cliques in RG. ∎

We now give an instance where the actual bound is attained. Let $n = 4k$, for some k. Four groups of rectangles are drawn as in Figure 3, each consisting of $n/4$ rectangles. It is now easy to identify $n^2/16$ cliques (shown as shaded regions).

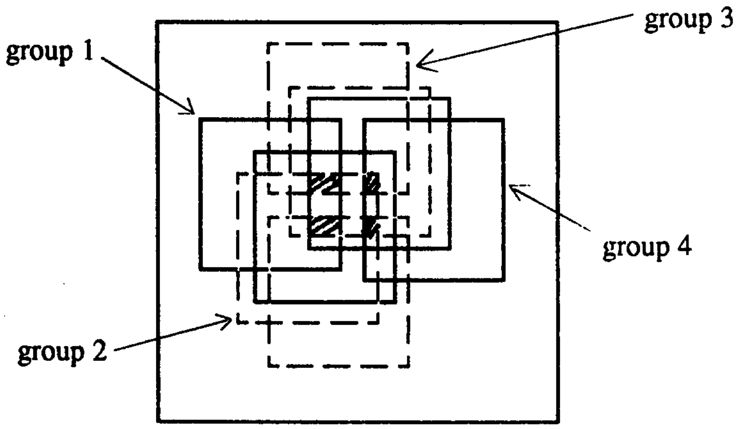


Figure 3. Example demonstration $O(N^2)$ cliques.

2.3. The Proposed Method

The key idea behind the proposed method is to scan a horizontal sweep-line across the rectangular floor F from its top to the bottom. Let $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ be the n points given on F. For each of these points (x_i, y_i), an isothetic rectangle R_i of size $(\alpha \times \beta)$ is drawn such that its top-left corner coincides with the point. Successive positions of the sweep-line are determined by y-coordinates of the points arranged in ascending order. The top and bottom sides of the rectangles are processed in increasing order of their y-values as the sweep line progresses; ties, if any, are resolved from left to right. This systematic processing of the rectangles is termed as *proper processing*. A rectangle is said to be *active* if its top boundary has been processed by the advancing sweep line, but whose bottom is not yet processed.

DEFINITION. Two intervals $[a, b]$ and $[c, d]$, $a < b$, $c < d$, $a < c$, are said to be *disjoint* if $c \geq b$; they *overlap* if $c < b < d$. The interval $[a, b]$ *contains* the interval $[c, d]$ if $a \leq c < d \leq b$. Two intervals are *intersecting* if either they overlap or one contains the other.

DEFINITION. A *window* is an interval $[a, b]$, $0 \leq a < b \leq u$, such that

(i) $a \in S$, $b \in S$ where S denotes the set $\{0, u, x_1, x_2, \dots, x_n, x_1 + \alpha, x_2 + \alpha, \dots, x_n + \alpha\}$;

(ii) the horizontal span of every active rectangle is either disjoint to or contains the interval $[a, b]$.

EXAMPLE. Consider a sweep line at height h as in Figure 4. At this instant of time, four rectangles are active. The interval $[x_1, x_2]$ is a window, but $[x_1, x_3]$ is not, as the latter intersects the horizontal span $[x_2, x_2 + \alpha]$ of a currently active rectangle.

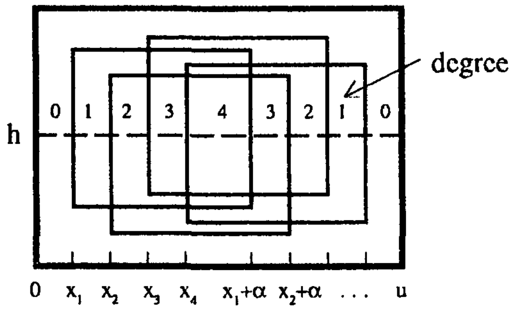


Figure 4. A set of active rectangles at height h and the fundamental windows of different degrees.

REMARK. If k rectangles are active, the horizontal span of the floor F is partitioned into $(2k + 1)$ windows whose intervals are *mutually-disjoint* and *collectively-exhaustive* at that particular height. Furthermore, they are linearly ordered from $x = 0$ to $x = u$. Thus, the number of windows *at any instant of time*, can be at most $2n + 1$.

DEFINITION. A rectangle is said to contain a window $[\ell, r]$, if the interval $[\ell, r]$ is contained within the horizontal span of the rectangle. The current *degree* (δ) of a window $[\ell, r]$ is the number of active rectangles containing the window.

The degrees of all windows for the above example are shown in Figure 4. From geometry of isothetic rectangles, it is obvious that the degrees of two adjacent windows differ exactly by one.

2.3.1. Generation and disappearance of windows

The processing starts when the horizontal sweep line coincides with the top boundary of the floor F. Subsequent positions of the sweep line are determined by y-coordinates of the top and bottom sides of the rectangles R_i, $(i = 1, 2, \dots, n)$ in ascending order, until it hits the bottom of F. For simplicity, we assume that no two points have the same x-coordinate. Also, assume that none of the points appears on the boundary of F.

The set of windows is updated dynamically. Initially, we have only one window $[0, u]$ corresponding to the top of F. Let us denote the horizontal span of R_i by the interval $I_i[\ell, r]$. When the sweep line hits the top(bottom) of a rectangle R_i, the number of active rectangles increases by one; the number of windows increases (decreases) exactly by two or one depending on whether or not I_i is contained in $[0, u]$.

Let R_i be a new rectangle whose *top* is currently under processing. The windows that are disjoint to I_i will remain as they are, their degrees being unchanged.

Since all windows at any instant of time are mutually disjoint, the following two *mutually-exclusive* and *collectively-exhaustive* cases might occur:

(i) if one or more windows are contained in I_i, then exactly two other windows must *overlap* with I_i. This happens because the set of windows is linearly ordered. The windows that are contained, remain same but their degrees are increased by one. Each of the two overlapping windows will be split into two smaller windows and their degrees are updated accordingly.

(ii) there exists a window $W_j[a,b]$ which contains I_i. In this case, W_j disappears and three new windows $[a,\ell]$, $[\ell,r]$ and $[r,b]$ are generated with degrees δ_j, δ_j+1, δ_j, respectively, where δ_j is the degree of W_j.

Figure 5 illustrates the processing of the top of a rectangle.

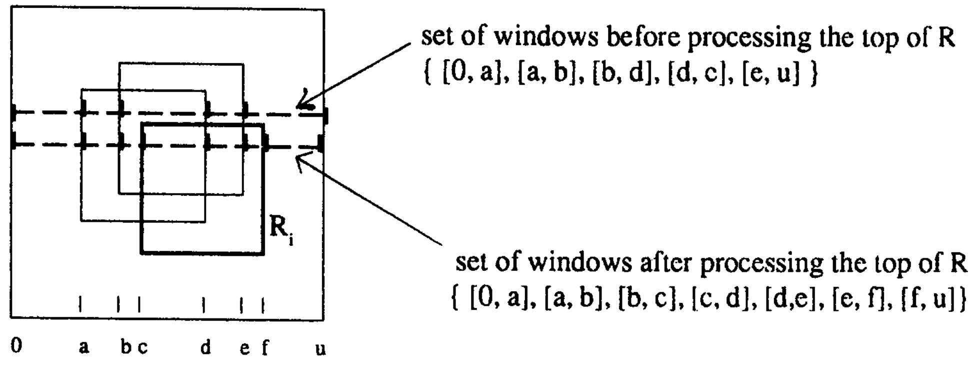

set of windows before processing the top of R
{ [0, a], [a, b], [b, d], [d, c], [e, u] }
set of windows after processing the top of R
{ [0, a], [a, b], [b, c], [c, d], [d,e], [e, f], [f, u]}
R_i
0
a
b
c
d
e
f
u

Figure 5. Processing the top of rectangle R_i.

When the bottom of R_i is processed, the count of active rectangles decreases by one. Two adjacent windows that touch the vertical line $x=\ell$, one from the left and the other from the right, are merged together to form a single window. Similarly, if $r<u$, two adjacent windows that are separated by the line $x=r$, are merged together. Their degrees are adjusted accordingly.

2.3.2. Enclosure problems and their relation to windows

Recall that problem **P1**, i.e., recognizing the rectangle enclosing maximum number of points, boils down to locating the rectangular region $Q_{\max}$. Let $[p,q]$ denote the interval representing the horizontal span of $Q_{\max}$. Now, the following lemma is immediate.

LEMMA 3. *At some point of time during processing, the interval $[p,q]$ will appear as a window. Furthermore, it will have the maximum degree among all the windows generated while processing the floor from top to bottom.* ∎

Similarly, problem **P2**, i.e., identifying the rectangle that encloses minimum number of points, can be solved by finding the window $[\ell,r]$ which satisfies two criteria:

(i) the rectangle (α,β) whose bottom-right corner coincides with (r,y_i), is contained within F, and

(ii) the degree of the window $[\ell,r]$ is minimum, where y_i denotes the y-coordinate of the next position of the sweep line.

To summarize, problem **P1** requires identification of the window having the maximum degree. In problem **P2**, one needs to find a window with minimum degree that satisfies certain containment property. All this information about the windows can be handled very conveniently and managed dynamically, using an interval tree as the underlying data structure.

2.4. Data Structure

Before describing our technique, we summarize some attributes of an interval tree which is used here as a vehicle to guide our search.

An *interval tree* (T) is a leaf-oriented balanced binary search tree where the leaf nodes from left to right hold distinct x-coordinates $\{x'_1,x'_2,\ldots,x'_m\}$, $m\le 2n+2$, sorted in ascending order, and $\forall i$, $1\le i\le m$, $x'_i\in S$ and lies within the interval $[0,u]$, where $S=\{0,u,x_1,x_2,\ldots,x_n, x_1+\alpha,x_2+\alpha,\ldots,x_n+\alpha\}$. Each internal node w will have the following information:

(i) Let T' be a subtree of T with a set of leaf nodes $\{x'_1, x'_2, \ldots, x'_m\}$. The root w of the subtree T' has the discriminant

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

The discriminant value of a leaf node is assumed to be the x-coordinate attached to it.

(ii) The left subtree of w is the interval tree with leaf nodes $\{x'_1, x'_2, \ldots, x'_{\lfloor m/2 \rfloor}\}$, and the right subtree is the interval tree with leaf nodes $\{x'_{\lfloor m/2 \rfloor + 1}, \ldots, x'_m\}$.

(iii) A secondary list $(w.L)$ of nodes with three fields $L.\ell$, $L.r$ 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 an additional forward link from w to the last node in the list.

Once the skeletal interval tree T is created, it is used to manage the database of windows. As the sweep line is moved, windows appear and disappear dynamically. When a new window $[\ell, r]$ appears, it is inserted in T. To do this, a top-down scan is made starting from the root w of T until it finds a node z satisfying the condition $\ell < d(z) < r$, for the first time. The window $[\ell, r]$ is attached to a node z which must be an *internal node* of T. This follows from the fact that the left (ℓ) and right (r) extremities of each window and the discriminants of all leaf nodes belong to the set S; the discriminant values of all internal nodes are distinct from the members of the set S. Thus, a *unique* node z with such a strict inequality can always be found. Similarly, when a window disappears, it is deleted from the interval tree.

LEMMA 4. *At any instant of time, the windows are associated to different internal nodes of the interval tree, each node carrying at most one window.*

PROOF. From the earlier discussion, it is clear that a window will be attached to a unique internal node in T. A floor scenario that contributes m leaf nodes in the interval tree can cause at most $m-1$ windows at any instant; the number of internal nodes of T will also be $m-1$. The windows are *mutually disjoint* and *linearly ordered.* As the discriminant values of all internal nodes are distinct from the members of the set S, the rest is obvious. ∎

To implement our algorithm, we need to associate the following secondary information with each internal node z of the interval tree T:

(i) the **window** associated to the node and its **degree**;

(ii) a field **maxdegree/mindegree** containing the maximum/minimum degree among all windows in the subtree T' rooted at z;

(iii) a pointer **target** pointing to the node whose associated window is of maximum/minimum degree in the subtree T'; in the case of a tie,

  (a) choose one arbitrarily for maximum enclosure problem and

  (b) choose the rightmost minimum for minimum enclosure problem;

(iv) a field **excess** that is used to update degrees of the windows in T';

(v) a pointer **father** pointing to its predecessor node in T.

2.5. Theme of the Algorithm

The processing starts inserting the window $[0, u]$ corresponding to the top of the floor in T. The **maxdegree(mindegree)** and **excess** fields of all internal nodes of T are set to zero. The **target** pointers are set to null. The top and bottom sides of all rectangles are then processed in proper sequence as stated earlier.

When the top (bottom) of a rectangle R_i whose horizontal span is the interval $I_i[\ell, r]$, is processed, windows appear (disappear). To determine this, one needs to find the set of windows that intersect the interval $[\ell, r]$. The interval tree T is traversed from the root to find the first node v^*, whose discriminant falls in the interval $I_i[\ell, r]$. Let F_{IN} be the path from the root to the node preceding v^*. Let F_L (F_R) be the path from v^* to the leaf node ℓ (r) as shown in Figure 6. Let T^* denote the subtree rooted at v^* and bounded by F_L and F_R, both inclusive. Let Forest(T^*) denote the forest whose components are subtrees of T^* and whose roots are immediate successors of nodes on F_L or F_R; in other words, Forest$(T^*) = T^* \setminus (F_L \bigcup F_R)$. The results stated in the following theorem are used later in our algorithm.

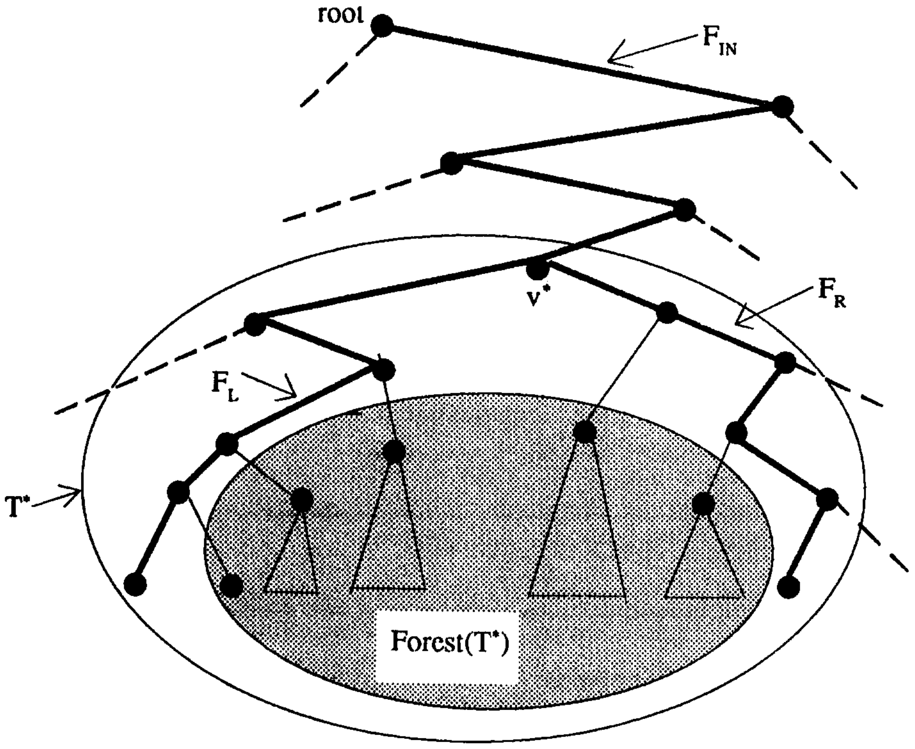


Figure 6. Search paths in the interval tree.

THEOREM 2.

(i) *The windows that intersect the interval I_i must be attached to the nodes in F_{IN}, and the nodes in T^*;*

(ii) *the windows (at most two) that overlap the interval I_i must be attached to the nodes lying on the paths F_{IN}, F_L, or F_R;*

(iii) *let z be any node in Forest(T^*); then the window attached to z is contained in the interval I_i, and every window that is contained in I_i, must be attached to some nodes of T^*;*

(iv) *if the interval I_i is contained in a window, it cannot intersect any other window at that instant of time. The window that contains I_i must be attached to a node in $F_{IN} \bigcup v^*$. This case can only arise while processing the top of a rectangle.*

PROOF. Follows from the properties of an interval tree and the nature of windows. ∎

For processing the top or bottom of each rectangle, one needs two passes. In the *forward pass*, search begins from the root of T, and explores F_{IN}, F_L and F_R. For each node in the search path, the **excess** field (as observed in the earlier pass), is propagated down to its two children, and then reset to zero. Concurrently, the insertion and deletion of windows, updating of their degrees and **excess** fields, are accomplished in this pass. The *backward pass* starts from the two leaf nodes of F_L and F_R, and continues until the root of T is reached. The final updating of **maxdegree/mindegree** and **target** pointers is done in this pass, with subsequent announcement of results.

Forward pass

Three cases may arise during the forward pass.

CASE 1. Let the window $W[x,y]$ overlap with $I_i[\ell,r]$ such that $x < \ell < y$. Clearly, W will be attached to a node on F_{IN} or F_L. Let the degree of W be δ. When the top of the rectangle R_i is processed, the window W is split into two windows $[x,\ell]$ and $[\ell,y]$ with degrees δ and $\delta+1$, respectively. Similarly, when W overlaps with $I_i[\ell,r]$ such that $\ell < y < r$, then it must be attached to some node on F_{IN} or F_R, and a similar action is taken. When the bottom of a rectangle is processed, at most two pairs of adjacent windows are obtained; one from the nodes in $F_{IN} \bigcup F_L$ and the other from a node in $F_{IN} \bigcup F_R$, whose common endpoints are ℓ and r, respectively. Each pair is merged to form a single window and inserted in the interval tree. The degrees of the new windows are updated accordingly.

CASE 2. The window $W[x,y]$ is contained in the interval $I_i[\ell,r]$. We go down along the search path F_L as well as F_R to check this. In this case, the degree of W is increased (decreased) by one as the top (bottom) of the rectangle is processed. Next, all windows present in Forest(T^*) must be contained in $I_i[\ell,r]$ as observed earlier. The degrees of all such windows will be increased (decreased) by one. This updating can be done *collectively* by incrementing (decrementing) **excess** and **maxdegree/mindegree** fields of all roots of Forest(T^*).

CASE 3. The interval $I_i[\ell,r]$ is contained in the window $W[x,y]$ attached to node z of $F_{IN} \bigcup v^*$. This situation may arise while processing the top of a rectangle. In this case, W is split into three parts $[x,\ell]$, $[\ell,r]$ and $[r,y]$ with degrees δ, $\delta+1$ and δ, respectively. It is easy to show that one of them will be attached to z; the other two will be attached to two distinct nodes on $F_L \bigcup F_R$.

In the forward pass, the **excess** fields evaluated in the earlier pass are also propagated along the search path $F_{IN} \bigcup F_L \bigcup F_R$ down the tree. Let v be a node on $F_{IN} \bigcup F_L \bigcup F_R$, whose successors are v_1 and v_2. When v is processed, **excess**(v) is added to **excess** and **maxdegree/mindegree** fields of v_1 and v_2. The degrees of windows attached to v_1 and v_2, if any, are also incremented by **excess**(v), and then **excess**(v) is reset to zero.

Backward pass

Once a rectangle is processed in the forward pass, a backward pass is performed starting from the two leaf nodes ℓ and r on F_L and F_R, respectively, retracing the path up to the root of T through the **father** pointers. For each node v in $F_{IN} \bigcup F_L \bigcup F_R$, the **maxdegree/mindegree** field and the **target** pointer are updated by comparing those of its two children v_1 and v_2.

Reporting the maximum-enclosing rectangle

We use a global variable **current-maximum** that holds the window having the maximum degree explored so far. To report the rectangle enclosing maximum number of points during the forward pass while processing the bottom side of each rectangle, we do the following: locate a node on the search path $F_{IN} \bigcup F_L \bigcup F_R$ whose **maxdegree** is maximum and find using the **target** pointer, the window W having the maximum **degree** just before updating. If the degree of W exceeds the current maximum, then **current-maximum** is set to W; the y-coordinate of the bottom side of the concerned rectangle is also stored. After complete processing, the window having the maximum degree will be identified; let it be $[\ell,r]$ and the y-coordinate of the bottom of the concerned rectangle be y_i. From Lemma 3, this window is contained in $Q_{\max}$. The bottom-right corner c_{br} of the plate P is now positioned at (r, y_i). In this position, P will enclose maximum number of points and lie totally within F (see Figure 1).

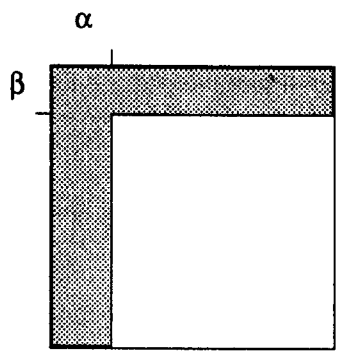


Figure 7. Forbidden zone for the min-enclosure problem.

Reporting the minimum-enclosing rectangle

If the minimum-enclosing rectangle is to be reported, the **mindegree** field is observed at the time of processing top of each rectangle; if the corner c_{br} of the plate $P(\alpha \times \beta)$, is placed on the rightmost point of such a window, it will enclose minimum number of points. As mentioned earlier, if c_{br} is placed in the minimum intersection region, some part of P may lie outside F. The shaded region in Figure 7 is the forbidden zone for positioning c_{br}. Thus, the value of **mindegree** is observed below the horizontal line $y = \beta$. The windows to the left of the line $x = \alpha$ are ignored, and the left extremities of the windows that intersect the line $x = \alpha$ are set to α. The rest of the processing is similar to that of the maximum-enclosing rectangle problem; the only difference is that the **mindegree** field of a node is set by observing those of its two successors in the backward pass. Ties, if any, are resolved by setting the target pointer to the rightmost window present in the subtree T'. One should stop processing when a window is obtained with **degree** zero, otherwise continue until the floor of F is reached.

The algorithm is given in the Appendix.

2.6. Complexity Analysis

Creation of the skeleton interval tree T requires $O(n \log n)$ time. Windows are attached to the nodes of T dynamically, but at any instant of time, the number of windows present in T is $O(n)$. Hence, the space complexity of the problem is $O(n)$. The processing of the top or bottom of a rectangle requires a forward and a backward traversal of nodes along the paths F_{IN}, F_L and F_R whose lengths are bounded by the depth of the interval tree. The updating of degrees of all windows in the Forest(T^*) is done collectively, and hence, can be accomplished in $O(\log n)$ time. Thus, the time complexity of processing a rectangle is $O(\log n)$. The total time complexity of processing all rectangles is $O(n \log n)$ for both the max- and min-enclosure problems.

3. OTHER APPLICATIONS

3.1. Rectangle-Containment Problem

The algorithm described earlier for solving point-containment problems can now be easily extended to the more general problem of maximizing/minimizing the rectangle-containment. These problems often arise in VLSI module placement, where one has to locate a rectangle of given size on the chip floor that is maximally or minimally congested. In other words, given a set of nonoverlapping, isothetic rectangular blocks $B_1, B_2, \ldots, B_n$ of arbitrary size on a rectangular floor F, one has to position an isothetic rectangular plate P of size $(\alpha \times \beta)$ on the floor such that

(i) P completely encloses maximum number of blocks (max-enclosure);

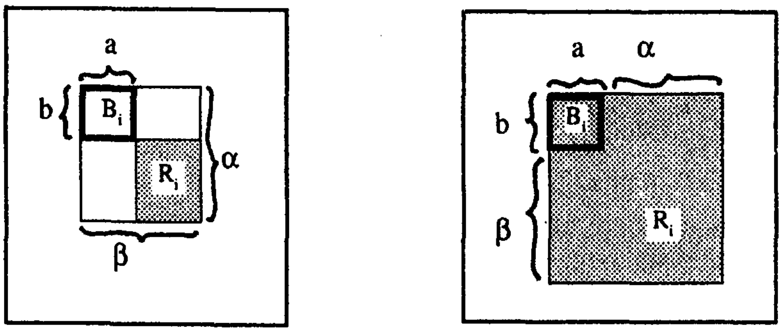
(ii) P encloses (completely or partly) minimum number of blocks (min-enclosure).

The plate P should be totally contained within F as before.

3.1.1. Theme of the algorithm

If the dimension of any rectangular block B_i exceeds that of P, then P cannot accommodate B_i. So, we can simply disregard it for the max-enclosure problem.

Consider a block $B_i(a \times b)$ whose top-left corner lie at the point p on the floor F (Figure 8a). Let us now draw a rectangle of size $(\alpha \times \beta)$ isothetically, with its top-left corner at p. Let R_i denote the shaded rectangular area, such that the two end points of its diagonal lie at the bottom-right corners of B_i and the bigger rectangle. Clearly, P will enclose B_i completely, if and only if the bottom-right corner of P lies in R_i. The shaded area R_i is defined as the *prime rectangle* for the block B_i.



(a) (b)

Figure 8. Prime rectangles.

For the min-enclosure problem, we define prime rectangles in a slightly different way. For each block $B_i(a \times b)$, the prime rectangle would be the isothetic rectangle R_i of size $((\alpha + a) \times (\beta + b))$, whose top-left corner coincides with that of B_i. Clearly, P will intersect B_i if and only if the bottom-right corner of P lies in R_i as shown in Figure 8b.

Once the prime rectangles for all blocks are drawn, the max- / min-enclosure problems become trivial. The max-enclosure problem can be solved by finding the maximum clique of the rectangle graph of the prime rectangles. To solve the min-enclosure problem, one has to locate a rectangular region where minimum number of prime rectangles intersect and a rectangle P of size $(\alpha \times \beta)$ can be drawn appropriately such that no part of P goes outside the floor F. Since construction of all prime rectangles takes $O(n)$ time, the overall complexity for both the problems will still be $O(n \log n)$.

3.2. Polygon-Containment Problem

The polygon-containment problem is the generalization of those discussed in the Section 3.1. Given n arbitrary polygons on the floor F, place a rectangular plate P isothetically such that

(i) P encloses completely maximum number of polygons, or

(ii) P intersects minimum number of polygons.

The max-enclosure problem for polygons immediately reduces to the rectangle-containment problem. The plate P encloses a polygon if and only if P encloses the minimum isothetic rectangle that circumscribes the polygon. Hence, the time complexity of the max-enclosure problem will be $O(N + n \log n)$, where N is the total number of vertices of all the polygons.

The min-enclosure problem for polygons as stated above, cannot be solved using the same technique. However, for isothetic polygons it can be solved by partitioning each polygon into disjoint rectangles, and then applying the following modified algorithm for solving the min-enclosure problem with rectangles in $O(N \log N)$ time. For general polygons, the location of P that *encloses completely* minimum number of polygons, can be solved easily in $O(N + n \log n)$ time.

Modified min-enclosure algorithm for isothetic polygons

Split each isothetic polygon into a minimum number of disjoint rectangles. Rectangles corresponding to a particular polygon have the same identity. The prime rectangles are drawn for each component. In the min-enclosure algorithm, when the top of a rectangle R is processed the degree of a window is increased if the corresponding identity of the polygon has newly appeared on that window. If more than one rectangle with the same identity as that of R overlap on a particular window, the count is increased by one but its degree is not incremented. Similarly, when the bottom of a rectangle R is processed, the degree of a window is decreased by one only if no rectangle with the same identity as of R is present on that window; otherwise the count corresponding to that identity is decremented by one. The algorithm terminates with a window of overall minimum degree. The minimum number of disjoint rectangles covering an isothetic polygon is linear in the number of edges. Thus, the overall complexity of the algorithm is $O(M \log M)$, where M is the total number of edges of all the polygons on the floor.

4. MAX-ENCLOSURE PROBLEM IN 3-D SPACE

Let a set of n points be distributed in a three-dimensional space and a rectangular box of specified size (length, breadth and height) be given. Our objective is to find the position of the box that encloses maximum number of points. The problem is a straightforward generalization of the two-dimensional max-enclosure problem with applications to clustering problem of pattern recognition, operations research, 3-D graphics etc. The algorithm for finding maximum clique in a hyper-rectangle intersection graph [13] can be used to solve this problem in $O(n^2)$ time. In this section, we extend our plane-sweep technique based on interval trees to tackle this problem. This algorithm gives a better average-case performance.

DEFINITION. A cuboid (rectangular parallelepiped) is a three-dimensional box bounded by six rectangular faces parallel to the appropriate coordinate axes. The corners of a cuboid C are represented by $C^{(i,j,k)}$, where i = top(t) or bottom(b), j = north(n) or south(s) and k = east(e) or west(w).

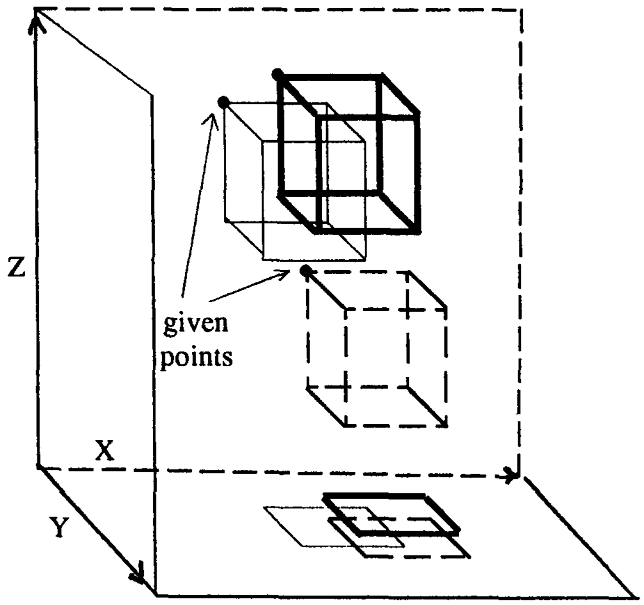
Assume that a set of n points be distributed in the 3-D space and let Π denote the given cuboid of size $(\alpha \times \beta \times \gamma)$. For each point i, $1 \leq i \leq n$, we draw a cuboid C_i of size $(\alpha \times \beta \times \gamma)$ such that the corner $C_i^{(t,n,w)}$ coincides with i. As in the 2-D case, the desired position of Π enclosing maximum number of points can be found by identifying the maximum clique of the intersection graph of the cuboids $C_i (1 \leq i \leq n)$. Furthermore, the intersection region of a clique has the shape of a cuboid.

4.1. Method

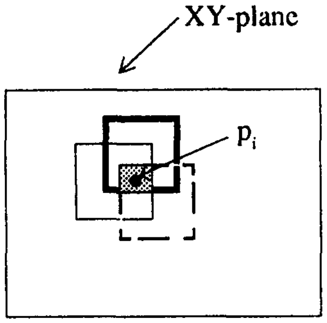
Our algorithm works in two phases. First we take the projections of all cuboids C_i, $1 \leq i \leq n$ on the XY-plane, and obtain a set of identical rectangles each of size $(\alpha \times \beta)$, as shown in Figure 9a. Then we find the set R^* of all maximal cliques of the rectangle intersection graph RG, by suitably modifying our earlier max-enclosure algorithm. Each clique in R^* denotes an intersection region, from which we choose a representative point p_i (see Figure 9b). Thus, we get a set of points $S = (P_1, P_2, \dots, P_\lambda)$, on the XY-plane, where λ is the cardinality of the set R^*.

In second phase, we determine the position of the given cuboid Π so that it encloses maximum number of points. This is equivalent to choosing a point (x_i, y_i, z_i) such that $(x_i, y_i) \in S$, and is enclosed by the largest number of C_i's. We find such a point by constructing a 2-D tree [20] on the set of points P and then using plane-sweep technique.

A 2-D tree with nodes corresponding to points in S can be constructed as follows. The root of the tree corresponds to the median of S with respect to x-coordinates of $p_1, p_2, \dots, p_\lambda$ and will have two children. The line (parallel to the y-axis) that passes through the median and partitions the set S into two equal halves, is called the *discriminant line*. Each of the two sets



(a) Projections of cuboids on the XY-plane.



(b) Any point in the maximal-intersection region of projected rectangles.

Figure 9.

is then subdivided into two equal-sized sets by a discriminant line parallel to the x-axis. This process is continued till all the sets become singletons or empty (see Figure 10). In addition, each node of the 2-D tree contains the following fields.

(i) **x**- and **y**-coordinates of the point p_m that lies on the discriminant line;

(ii) **degree** of the point p_m (indicating the number of cuboids enclosing p_m at a particular instant of time during the sweep of the XY-plane); initially, all degrees are set to zero;

(iii) **maxdegree** containing the maximum degree among all nodes in the subtree rooted at p_m;

(iv) a pointer **target** pointing to the node having maximum degree in the subtree rooted at p_m;

(v) an **excess** field, which is used to update degrees of all points in the subtree rooted at p_m (analogous to the **excess** field of the nodes in interval tree discussed earlier);

(vi) a pointer **father** pointing to its predecessor node.

We now implement the downward sweep of the XY-plane (see Figure 11). Successive positions of the plane are determined by z-coordinates of top and bottom sides of the cuboids in nonincreasing order. Each time the top (bottom) of a cuboid is processed, the search initiates from the root of the 2-D tree with the corresponding query rectangle R_i. By using the 2-D tree, one can now determine the points S' in S that are enclosed within R_i. When a top (bottom) side is processed, degrees of all points in S' are increased (decreased) by one. This updating can be made *collectively* (without explicitly reporting each member of S', and updating their degrees individually), by using the **excess** field as described in the 2-D case. The point whose degree is currently maximum, can be obtained using **maxdegree** and **target** fields. The processing stops when the sweeping plane reaches the floor. We then observe the point $(x_i, y_i) \in S$, whose degree is maximum, and by noting the z-coordinate (z_i) where it is attained. The cuboid Π enclosing maximum number of points can now be determined by coinciding the corner $\Pi^{(b,s,e)}$ with the point (x_i, y_i, z_i).

4.2. Complexity

All maximal cliques of the rectangle graph RG formed by projections of the cuboids on the XY-plane, can be obtained in $O(\lambda + n \log n)$ time by modifying our algorithm slightly. The 2-D tree consisting of λ representative points can be constructed in $O(\lambda \log \lambda)$ time and in $O(\lambda)$ space.

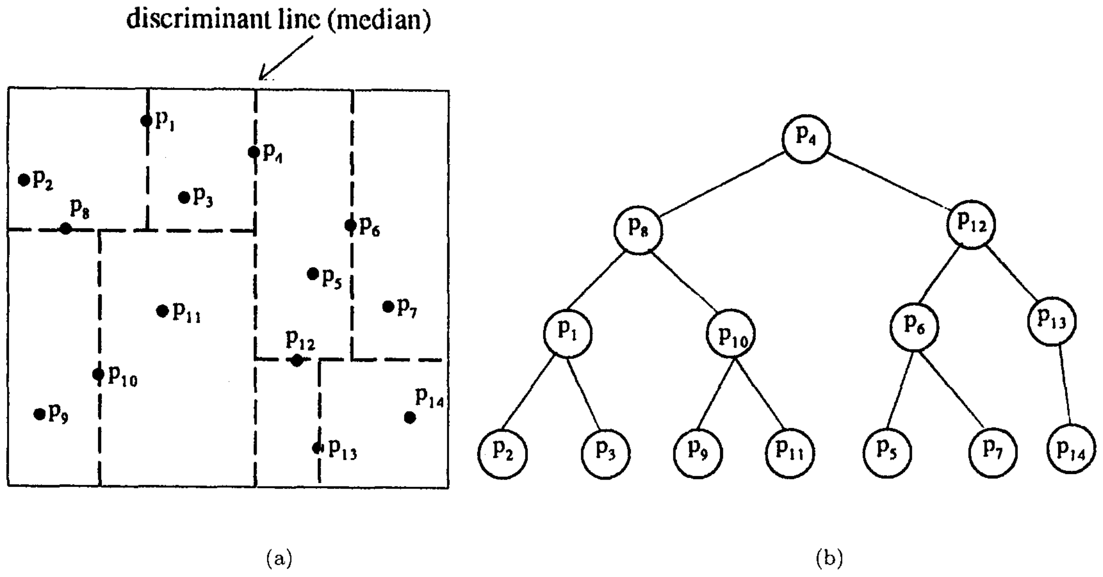


Figure 10. Downward sweep of the XY-plane and updating of degrees of the representative points.

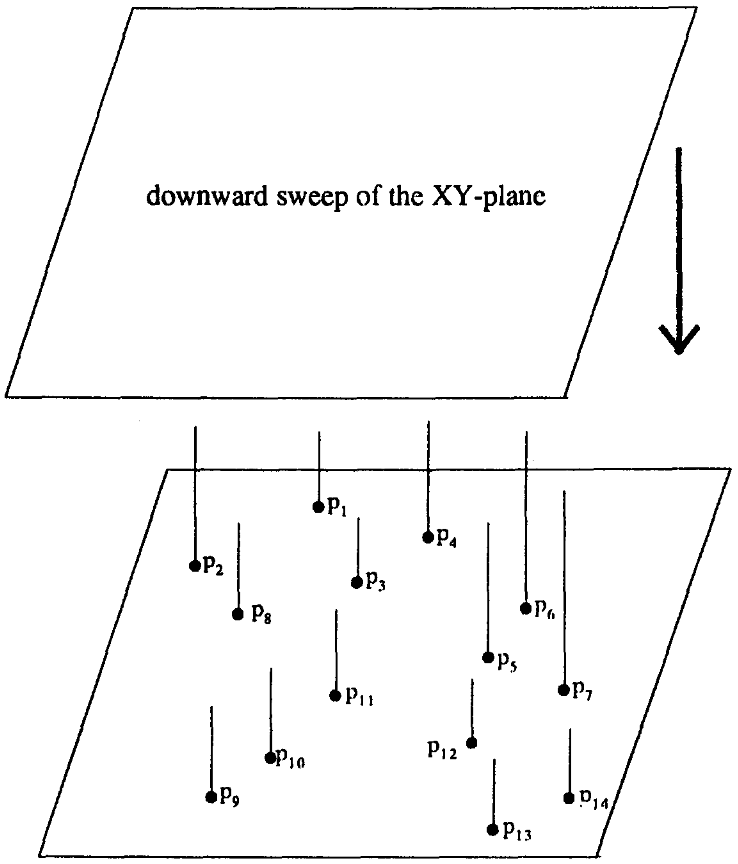


Figure 11. Downward sweep of the XY-plane and updating of degrees of the representative points.

For processing the top and bottom face of a cuboid, the search initiates from the root of the 2-D tree. At any node p_j on the search path, the discriminant line may or may not intersect the query rectangle. In the former case, the query rectangle is split into two components, which in turn, will serve as query rectangles of the two children of p_j. In the latter case, the search is directed to the appropriate child of p_j. The search with the current face will terminate when each of the components of the face is either bounded by exactly four discriminant lines, or reaches a leaf node. Given a query rectangle, the complexity of such a traversal in a 2-D tree having λ leaves is $O(\sqrt{\lambda})$. Hence, the total time complexity of the algorithm is $O(\lambda \log \lambda + n\sqrt{\lambda})$ where λ may be $O(n^2)$ in the worst case (Theorem 1). Empirical evidence however, shows that λ is subquadratic in most cases.

5. CONCLUSION

To summarize, this chapter demonstrates a new, unified and easy-to-implement method of finding maximum- and minimum-point enclosing rectangle in a 2-D plane, in $O(n \log n)$ time and $O(n)$ space. The technique is based on the classical line-sweep paradigm and implemented using an interval tree as the underlying data structure. The method can easily be extended for solving the max- or min-enclosure problems amidst a set of rectangular or in general, polygonal obstacles in a plane. The max-enclosure problem in the 3-D case can also be formulated in this framework, and an algorithm based on plane-sweep technique is presented. The method works amidst a set of points, cuboids or polyhedral objects of arbitrary shapes. For the min-enclosure problem in a 3-D environment, an $O(n^2)$ algorithm can be developed similarly. These enclosure problems have various applications to pattern recognition, VLSI design automation and computer graphics.

APPENDIX 7.1

ALGORITHM : OPTIMAL-ENCLOSURE

```
Input  : A set of points distributed in $\mathbb{R}^2$ and a given rectangular plate $P$ of size $(\alpha, \beta)$.
Output: The position of the bottom-right corner of $P$ satisfying optimality of the
        objective function.
begin
    Let $R = \{R_1, R_2, \ldots R_n\}$ be the set of isothetic rectangles of size $(\alpha, \beta)$ where $R_i$ is
    drawn positioning its top-left corner at the i-th point;
    Construct a skeleton interval tree (T) with the $x$-coordinates of the left and
    right boundaries of all members in $R$;
    Insert the interval corresponding to the roof of the floor in T with degree zero;
    /* Process top and bottom boundaries of the rectangles in $R$ in proper sequence. */
    For each boundary do
    Let $I_i$ be the current boundary with horizontal interval $[\ell, r]$ at height $h$
    begin
        $F_{IN}$ := set of nodes from root to a node $v^*$ (fork node) such that $1 \leq d(v^*) \leq r$;
        $F_L$ := set of nodes from on the path from $v^*$ to $\ell$;
        $F_R$ := set of nodes from on the path from $v^*$ to $r$;
        if $I$ is a top boundary then
        begin
            if problem = min-enclosure then SET-MIN-DEGRE;
            for each node $v \in F_{IN}$ do PROCESS-$F_{IN}(v)$;
            for each node $v \in F_L$ do PROCESS-$F_L(v)$;
            for each node $v \in F_R$ do PROCESS-$F_R(v)$;
        end;
        if $I$ is a bottom boundary then
```

```
            begin
                for each node $v \in F_{IN}$ do PROCESS-$F_{IN}(v)$;
                for each node $v \in F_L$ do PROCESS-$F_L(v)$;
                for each node $v \in F_R$ do PROCESS-$F_R(v)$;
                if problem = max-enclosure then SET-MAX-DEGRE;
            end;
            traverse from leaf nodes $a$ and $b$ to the root along the search path to find
            the window with maximum degree and to set the max or mindegree fields
            and target pointers of all the nodes along the search path;
        end;
    end;

Procedure PROCESS-$F_{IN}(v)$
begin
    propagate excess($v$) to the left and right children of $v$ and set excess($v$) to 0;
    let $W[x, y]$ be the window associated to node $v$ with degree $\delta$;
    if $x < \ell < y$ then split $[x, y]$ into $[x, \ell]$ and $[\ell, y]$ with degrees $\delta$ and $\delta$+1 respectively;
    if $x < r < y$ then split $[x, y]$ into $[x, r]$ and $[r, y]$ with degrees $\delta$+1 and $\delta$ respectively;
    if $[\ell, r] \in [x, y]$ then split $[x, y]$ into $[x, \ell]$, $[\ell, r]$ and $[r, y]$ with degrees $\delta$, $\delta$+1 and $\delta$
                        respectively;
end;

Procedure PROCESS-$F_L(v)$
begin
    propagate excess($v$) to the left and right children of $v$ and set excess($v$) to 0;
    let $W[x, y]$ be the window associated to node $v$ with degree $\delta$;
    if $[x, y] \in [\ell, r]$ then
    begin
        if $[\ell, r]$ corresponds to the top boundary then increase degree($v$) by 1;
        if $[\ell, r]$ corresponds to the bottom boundary then decrease degree($v$) by 1;
    end;
    if $[x, y]$ overlaps $[\ell, r]$ then split $[x, y]$ into $[x, \ell]$ and $[\ell, y]$ with degrees $\delta$ and $\delta$+1
                            respectively;
    if search path proceeds to the left of $v$ then add (subtract) 1 to (from) the
                            maxdegree and excess field of its right child depending
                            on whether $[\ell, r]$ is a top (bottom) boundary;
end;

Procedure PROCESS-$F_R(v)$
begin
    Similar to processing nodes in $F_L$;
end;
Procedure SET-MIN-DEGREE
begin
    /* let current-minimum contains the minimum degree up to the current
        instant of time.
        $(p_x, p_y)$ is a point having minimum degree */
    for all nodes on the search path (in $F_{IN}$, $F_L$ and $F_R$) check the mindegree fields;
    if (mindegree < min) then
    begin
        obtain the window $[a, b]$ having minimum degree using target pointer;
```

```
            if ($b < \alpha$ or $h < \beta$) /* check boundary conditions*/
                then exit else min := mindegree; $(p_x, p_y) := (b, h)$;
        end;
end;
Procedure SET-MAX-DEGREE
begin
        the procedure is identical to SET-MIN-DEGREE; before setting $(p_x, p_y)$,
        the boundary conditions need not be checked;
end;
```

REFERENCES

1. J.L. Bentley and J.H. Friedman, Data structure for range searching, *Computing Surveys* **II**, 397–409 (1979).
2. J. Bentley and H.A. Maurer, Efficient worst case data structure for range searching, *Acta Informatica* **13**, 155–168 (1980).
3. G.S. Leuker, A data structure for orthogonal range queries, In *Proc. of the 19th Annual IEEE Symp. on Foundation of Computer Science*, pp. 28–34, (1978).
4. B. Chazelle, R. Cole, F.P. Preparata and C.K. Yap, New upper bounds for neighbor searching, Tech. Rep. CS-84-11, Brown University, Providence, RI, (1984).
5. B. Chazelle, An improved algorithm for fixed-radius neighbor problem, *Information Processing Letters* **16**, 193–198 (1983).
6. B. Chazelle and H. Edelsbrunner, Optimal solutions for a class of point retrieval problems, *Journal of Symbolic Computations* **1**, 47–56 (1985).
7. D.T. Lee, On k-nearest neighbor Voronoi diagrams in the plane, *IEEE Transactions on Computers* **C-31**, 478–487 (1982).
8. A. Aggarwal and S. Suri, Fast algorithm for computing the largest empty rectangle, In *Proc. 3rd Annual ACM Symposium on Computational Geometry*, pp. 278–290, (1987).
9. B. Chazelle, Filtering search: A new approach to query-answering, *SIAM Journal on Computing* **15**, 703–724 (1986).
10. B.M. Chazelle and D.T. Lee, On circle placement problem, *Computing* **36**, 1–16 (1986).
11. D.T. Lee and F.P. Preparata, An improved algorithm for the rectangle enclosure problem, *Journal of Algorithms* **3**, 218–224 (1982).
12. H. Imai and T. Asano, Finding the connected components and maximum clique of an intersection graph of rectangles in the plane, *Journal of Algorithms* **4**, 310–323 (1983).
13. D.T. Lee, Maximum clique problem of rectangle graphs, In *Advances in Computing Research*, (Edited by F.P. Preparata), pp. 91–107, JAI Press, (1983).
14. A. Naamad, D.T. Lee and W.L. Hsu, On the maximum empty rectangle problem, *Discrete Applied Mathematics* **8**, 267–277 (1984).
15. B. Chazelle, R.L. Drysdale and D.T. Lee, Computing the largest empty rectangle, *SIAM Journal of Computing* **15**, 300–315 (1986).
16. M. Orlowski, A new algorithm for the largest empty rectangle problem, *Algorithmica* **5**, 65–73 (1990).
17. S.C. Nandy, B.B. Bhattacharya and S. Ray, Efficient algorithms for identifying all maximal isothetic empty rectangles in VLSI layout design, In *Proc. FST & TCS - 10, Lecture Notes in Computer Science*, Vol. 437, pp. 255–269, Springer-Verlag, (1990).
18. F.P. Preparata and M.L. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, (1985).
19. M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, (1980).
20. J.L. Bentley, Multidimensional binary search trees used for associative searching, *Communications of the ACM* **18**, 509–517 (1975).
21. H. Edelsbrunner, Dynamic data structure for orthogonal intersection queries, Rep. F59, Tech. Univ. Graz, Institute for Informationsverarbeitung, (1980).
22. A. Asano, M. Sato and T. Ohtsuki, Computational geometric algorithms, In *Layout Design and Verification, Advances in CAD for VLSI*, (Edited by T. Ohtsuki), Vol. 4, pp. 295–347, North Holland, (1986).
23. S.C. Nandy, Studies on some geometric algorithms with applications to VLSI, Ph.D. Thesis, Calcutta University, (1994).
24. D.T. Lee and C.K. Wong, Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees, *Acta Informatica* **9**, 23–29 (1977).