Theoretical Computer Science 16 (1981) 329–336
North-Holland Publishing Company

NOTE

A SPACE-OPTIMAL SOLUTION OF GENERAL REGION LOCATION

H. EDELSBRUNNER and H.A. MAURER
Institut für Informationsverarbeitung, Technical University of Graz, A-8010 Graz, Austria

Communicated by M. Nivat
Received June 1980
Revised February 1981

Abstract. In 1979 Kirkpatrick obtained a practically feasible algorithm for planar region location working in linear space and logarithmic time, provided the regions are bounded by straight line segments. No algorithm requiring only linear space and log-polynomial time was known, so far, for general planar region location, i.e. for the case where regions are bounded by curves more complicated than straight line segments. As main result of this paper such an algorithm is presented.

1. Introduction and preliminaries

The central notion in this paper is that of a region: A *region* R is a simply connected, closed and bounded subset of the Euclidean plane such that

(i) any line parallel to the y-axis intersects R in at most two points or is a tangent line, and

(ii) there are algorithms l, r and a such that

(a) l yields in O(1) steps a left endpoint of R which we denote by $l(R)$;

(b) r yields in O(1) steps a right endpoint of R which we denote by $r(R)$;

(c) if p is any point whose x-coordinate is between the x-coordinates of $l(R)$ and $r(R)$, then $a(R, p)$ yields in O(1) time 'below', 'in', or 'above', depending on whether p is below, in, or above R.

Typical regions are triangles, rectangles, convex polygons with a bounded number of edges, circles, segments of circles, etc.

In our definition of region and throughout this paper we choose the y-axis as prefered axis. Everything to be said is of course also true if x- and y-axis are consistently interchanged. Our aim in this paper is to develop a data structure which allows us to preprocess a collection of n non-overlapping regions such that the *region location problem* (or *region location query*) for an arbitrary point p can be answered

0304-3975/81/0000–0000/$02.75

efficiently, i.e. the region to which p belongs (if it exists) can be determined efficiently.

The region location problem has attracted considerable attention in the past since it occurs as subproblem of many important questions, such as e.g.

(a) the post-office problem (also called nearest neighbor problem), see e.g. Knuth [7], Shamos [13], and Maurer and Ottmann [11],

(b) the near neighbor problem see e.g. Bentley and Maurer [1], and Preparata [12] and

(c) the contour problem, see Lipski, Preparata [9].

A first solution to the region location problem was obtained by Dobkin and Lipton [2] using a 'slab technique', requiring $O(n^2)$ space and $O(n^2)$ preprocessing, but requiring only $O(\log n)$ time for queries.

For regions bounded by straight-line segments a space-optimal (i.e. $O(n)$ space) solution with $O(\log^2 n)$ query time was given by Lee and Preparata in [8]. This was improved by ingenious methods to $O(\log n)$ query time by Lipton and Tarjan [10]. A practically feasible method with same space and time bounds was finally developed by Kirkpatrick [6]. This result essentially settles the region location problem for polygonal regions. However, none of the techniques mentioned above (except for the initial 'slab technique') carries over to arbitrary regions.

The best solution so far for the general region location problem is due to Preparata [12], who obtains, via a clever refinement of the 'slab technique', an algorithm running in $O(\log n)$ time and requiring $O(n \log n)$ space. Preparata's algorithm has two significant drawbacks:

(i) it is quite complicated and not easy to implement, and

(ii) it uses more than $O(n)$ space.

This paper introduces a space-optimal algorithm for the region location problem, the only linear space algorithm with 'reasonable' (i.e. polynomial in $\log n$) query time known to date.

Our algorithm is applicable to a number of special region location problems. It can be carried over to dynamic situations by using any of a variety of known dynamization methods that dynamize by splitting the static data structure (these methods are also applicable to Preparata's and Kirkpatrick's methods), or by balancing the structure. The latter method yields the best known solution for dynamic rectangle location (i.e. the regions are all non-overlapping rectilinearly-oriented rectangles). We discuss these dynamization aspects at the end of this paper.

Before starting our technical sections, a few notational remarks are convenient.

For a point p, p_x and p_y denote its x- and y-coordinate, respectively. Thus, for a region R, the left endpoint q of R is the point with coordinates $l(R)_x$ and $l(R)_y$, i.e. $q = (l(R)_x, l(R)_y)$.

A region R_1 is *below* a region R_2 (and R_2 is *above* R_1) if a line l parallel to the y-axis exists which intersects both R_1 and R_2 and, along l, R_1 is below R_2 (R_2 is above R_1). Observe that if R_1 and R_2 are non-overlapping, and R_1 is below R_2 along a line l parallel to the y-axis, then R_1 is below R_2 along each such line.

2. A space-optimal solution

In this section we reduce the region location problem in two steps to a simpler problem which we term 'low-point' problem. Using an efficient algorithm for the low-point problem we obtain a space-optimal solution for the region location problem for n non-overlapping regions: Using $O(n)$ space, $O(n \log n)$ preprocessing time and $O(\log^3 n)$ time for answering a query is shown to suffice.

We first require the notion of *n skewered regions*: We call n non-overlapping regions *skewered* if there is a line (the skewer) parallel to the y-axis meeting each of the regions. (Note the similarity to the structure in [3].) The *skewered-regions problem* is the region location problem for n skewered regions. Its significance for the (general) region location problem is due to the following result.

Lemma 2.1. *If A' is an algorithm which solves the region location problem involving n skewered regions with preprocessing $P'(n) = f_1(n)\, n \log n$, storage $S'(n) = f_2(n)n$ and query time $Q'(n)$ (where f_1 and f_2 are strictly positive, non-decreasing functions), then an algorithm A for the region problem involving n regions can be obtained with preprocessing $P(n) = O(P'(n))$, storage $S(n) = O(S'(n))$ and query time $Q(n) = O(Q'(n) \log n)$.*

Proof. Let C be a collection of n non-overlapping regions and let $Z = \{l(R)_x \mid R \text{ in } C\} \cup \{r(R)_x \mid R \text{ in } C\}$ be the set of all endpoints of regions in C. We will assume that Z consists of $2n$ distinct values which we list, in ascending order, as $Z = a_1, a_2, \ldots, a_{2n}$. Let $m = \frac{1}{2}(a_n + a_{n+1})$ and let g be the line $x = m$. Let

$$C_1 = \{R \text{ in } C \mid R \text{ is to the left of } g\},$$

$$C_2 = \{R \text{ in } C \mid g \text{ intersects } R\},$$

$$C_3 = \{R \text{ in } C \mid R \text{ is to the right of } g\}.$$

The skewer tree for C is a tree with root W whose value is m, whose left subtree is the skewer tree for C_1 and whose right subtree is the skewer tree for C_3. Further C_2, a set of skewered regions (suitably preprocessed for skewered-region queries using algorithm A') is also associated with W. Observe that $|C_1| \leq \frac{1}{2}n$, $|C_3| \leq \frac{1}{2}n$, and $|C_2| \leq n$.

To answer a region location query for a point q using such a skewer tree we carry out a skewered-region query for q in C_2 requiring time $O(Q'(n))$; in addition, we carry out a region location query for q in the region tree corresponding to either C_1 or C_3 depending on whether $q_x < m$ or $q_x > m$. Evidently, $Q(n) = Q(\frac{1}{2}n) + O(Q'(n))$, i.e. $Q(n) = O(Q'(n) \log n)$. We clearly have $P(n) = O(n \log n) + \bar{P}(n)$, where $\bar{P}(n)$ is the time required for the preprocessing of the skewered regions. Since no region is skewered twice we have

$$\bar{P}(n) \leq \sum_{n_1+n_2+\cdots+n_t=n} P'(n_i) \leq f_1(n) \log n \sum_{n_1+n_2+\cdots+n_t=n} n_i$$
$$\leq f_1(n) n \log n,$$

consequently $P(n) = O(P'(n))$ which completes the proof.

We next introduce an auxiliary problem, the low-point problem. We show in Lemma 2.2 how solutions to that problem lead to solutions for the skewered-regions problem and hence, by Lemma 2.1, to the region problem. By exhibiting in Lemma 2.3 a space efficient algorithm for the low-point problem we obtain as Theorem 2.4 the space efficient algorithm for the region problem mentioned earlier.

Consider a set P of n points in the plane. For convenience, we assume that all x-coordinates and all y-coordinates are distinct. A low-point query consists of a point q and yields as answer that point of P with minimal y-coordinate which is to the left and above q (if such a point exists). Fig. 1 illustrates that for $P = \{p_1, p_2, \ldots, p_8\}$ and q as shown, p_3 is the desired point.

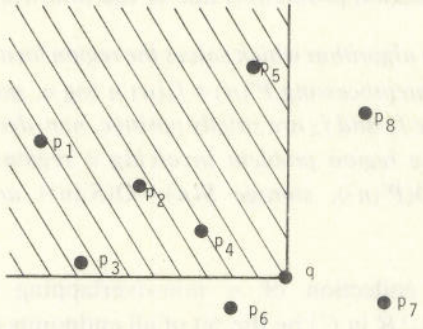

Fig. 1.

The next lemma establishes the connection between the low-point and the skewered-regions problem.

Lemma 2.2. *Any algorithm A'' for the low-point problem involving n points with query time $Q''(n)$, preprocessing time $P''(n) = f_1(n)n \log n$ and storage requirement $S''(n) = f_2(n)n$ gives rise to an algorithm A' for the skewered-regions problem involving n regions with query time $Q'(n) = O(Q''(n) \log n)$, preprocessing time $P'(n) = O(P''(n))$ and storage requirement $S'(n) = O(S''(n))$. (As in Lemma 2.1 f_1 and f_2 are strictly positive, non-decreasing functions.)*

Proof. Consider a set C of n regions skewered by a line g and sorted as R_1, $R_2, \ldots, R_n$ such that R_i is below R_{i+1} for $i = 1, 2, \ldots, n-1$. Let $p'_1, p'_2, \ldots, p'_n$ be the left endpoints of the regions $R_1, R_2, \ldots, R_n$, respectively. Let l be a line parallel to the y-axis passing through the leftmost of the points $p'_1, p'_2, \ldots, p'_n$. (See Fig. 2(a) for an example.)

To understand our algorithm, consider a line h parallel to the x-axis above R_n (indicated as dashed line in Fig. 2(a)) and imagine each region R_i extended to the left up to the line h as follows: through the left endpoint p_i of the region R_i draw a line upward and parallel to the y-axis until the line either meets the bottom curve of some region or until it meets the line h. (The lines at issue are indicated as dashed lines in

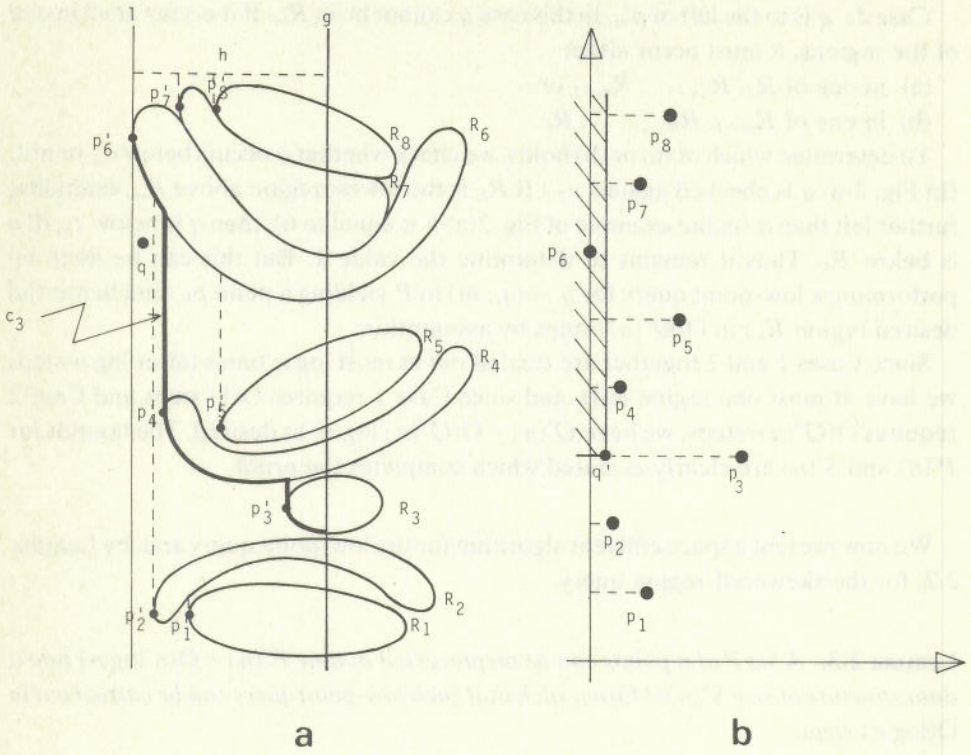

Fig. 2.

Fig. 2(a).) In this way we have associated with each region R_i a curve c_i bounding that region from below (indicated solidly in Fig. 2(a) for R_3) such that a point q above c_i cannot occur in any region R_j with j smaller than i.

Consider now a set of n points defined by $P = \{p_i = (x_i, i) \mid x_i$ is x-coordinate of p'_i, for i ranging from 1 to $n\}$. The set of points corresponding to $p'_1, p'_2, \ldots, p'_8$ of Fig. 2(a) is shown in Fig. 2(b). Observe that the y-coordinates of p'_5 is smaller than the one of p'_4 but the situation is the other way round for p_4 and p_5.

Consider the set P preprocessed for an algorithm A for low-point queries in P. We are now ready to describe how we locate a point $q = (q_x, q_y)$ between the lines l, h, and g among the regions $R_1, R_2, \ldots, R_n$. (The cases that q is left of l, right of g, or above h are either trivial or treated analogously.) At each stage we will try to locate q among the regions $R_i, R_{i+1}, \ldots, R_j$ (initially i is equal to 1 and j is equal to n). We let $m = \frac{1}{2}(i+j)$ and consider the relative position of q with respect to R_m. Two cases are distinguished:

Case 1: q is to the right of p'_m (the left endpoint of R_m). We test whether q is in R_m (in which case we are finished), whether q is below R_m (in which case we continue our location algorithm for the regions $R_1, R_2, \ldots, R_{m-1}$), or whether q is above R_m (in which case the location algorithm is continued for the regions $R_{m+1}, R_{m+2}, \ldots, R_j$).

Case 2: q is to the left of p'_m. In this case q cannot be in R_m. If it occurs at all in one of the regions, it must occur either

(a) in one of $R_1, R_2, \ldots, R_{m-1}$ or

(b) in one of $R_{m+1}, R_{m+2}, \ldots, R_j$.

To determine which of (a) or (b) holds, we check whether q occurs below c_m or not. (In Fig. 2(a) q is checked against c_3.) If R_k is the lowest region above R_m extending further left than q (in the example of Fig. 2(a) k is equal to 6), then q is below c_m iff q is below R_k. Thus it remains to determine the value k. But this can be done by performing a low-point query for $q' = (q_x, m)$ in P yielding a point p_k (and hence the desired region R_k) in $O(Q''(n))$ steps by assumption.

Since Cases 1 and 2 together are carried out at most $\log n$ times (after $\log n$ steps we have at most one region left), and since Case 1 requires $O(1)$ steps and Case 2 requires $O(Q''(n))$ steps, we have $Q'(n) = O(Q''(n) \log n)$ as desired. The bounds for $P'(n)$ and $S'(n)$ are clearly as stated which completes the proof.

We now present a space efficient algorithm for the low-point query and, by Lemma 2.2, for the skewered-region query.

Lemma 2.3. *A set P of n points can be preprocessed in time $P''(n) = O(n \log n)$ into a data structure of size $S''(n) = O(n)$, such that each low-point query can be carried out in $O(\log n)$ steps.*

Proof. As before, we assume that all x-coordinates and all y-coordinates are distinct and strictly positive. Our algorithm is obtained by combining the 'locus approach' (i.e. the idea of regions of constant answer) with the polygon location algorithm of Kirkpatrick [6].

Consider the set of n points P sorted is ascending order by y-coordinate, $P = p_1, p_2, \ldots, p_n$. Let R be an axis-parallel rectangle containing all of P. Associate, for i ranging from 1 to n, to each point p_i a region M_i as follows:

M_i is that part of R below and to the right of p_i which is not yet in $M_1 \cup M_2 \cup \cdots \cup M_{i-1}$. Observe that each M_i is defined by two half-rays starting at p_i, each producing exactly one new intersection point. Thus, after carrying out the described process for $p_1, p_2, \ldots, p_n$, we have a straight-line subdivision of R with exactly $4+3n$ vertices.

The situation is depicted for 6 points in Fig. 3.

The crucial observation is that a point q is in M_i if and only if the low-point query for q yields as answer p_i. Thus, to solve a low-point query for q we just have to carry out a polygon location algorithm in the straight-line subdivision of R given by the $4+3n$ points. By the algorithm of Kirkpatrick [6], this can be done in $O(\log n)$ steps using $O(n)$ space and $O(n \log n)$ preprocessing which completes the proof.

We now have obtained the main theorem of this paper.

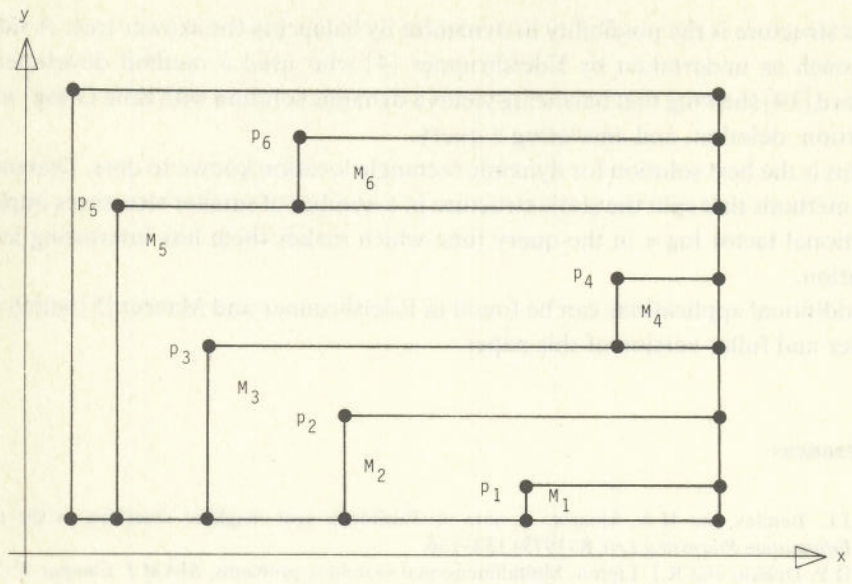

Fig. 3.

Theorem 2.4. *Region location in a collection of n non-overlapping regions can be carried out in time* $Q(n) = O(\log^3 n)$, *using a data structure requiring* $O(n \log n)$ *preprocessing and* $O(n)$ *space.*

Proof. Combine Lemma 2.1, Lemma 2.2, and Lemma 2.3.

3. Remarks and applications

In the preceding section, a new data structure supporting general region location in the plane has been introduced. The significance of this problem (and therefore of the structure) stems from the reduction of a number of geometric problems to certain types of region location.

Our solution is well suited for practical situations, since it requires only linear space and is quite simply implemented. We would like to mention briefly one type of region location that occurs as subproblem for computing the contour of a set of vertical and horizontal line segments which was investigated by Lipski and Preparata [9].

Given a set of non-overlapping rectilinearly-oriented rectangles as set of regions, determine the rectangle a query point is in. Since the regions are known to be rectangles, the reduction to the low-point problem (see Lemma 2.2) is not necessary which saves a factor $\log n$ in time for answering a query.

Hence, we have a solution for the static rectangle location problem with linear space and $O(\log^2 n)$ query time. Another advantage of working without the low-

point structure is the possibility to dynamize by balancing the skewer tree. A similar approach as undertaken by Edelsbrunner [4] who used a method developed by Willard [14] showing that balancing yields a dynamic solution with time $O(\log^2 n)$ for insertion, deletion, and answering a query.

This is the best solution for dynamic rectangle location known to date. Dynamization methods that split the static structure in a number of smaller structures imply an additional factor $\log n$ in the query time which makes them less interesting in this situation.

Additional applications can be found in Edelsbrunner and Maurer [5] which is an earlier and fuller version of this paper.

References

[1] J.L. Bentley and H.A. Maurer, A note on Euclidean near neighbor searching in the plane, *Information Processing Lett.* **8** (1979) 133–136.
[2] D.P. Dobkin and R.J. Lipton, Multidimensional searching problems, *SIAM J. Comput.* **5** (1976) 181–186.
[3] H. Edelsbrunner, A new approach to rectangle intersections, submitted for publication.
[4] H. Edelsbrunner, Dynamic data structures for orthogonal intersection queries, Technical University of Graz, Institut für Informationsverarbeitung, Report 59 (1980).
[5] H. Edelsbrunner and H.A. Maurer, On region location in the plane, Technical University of Graz, Institut für Informationsverarbeitung, Report 52 (1980).
[6] D.G. Kirkpatrick, Optimal search in planar subdivisions, University of British Columbia, Department of Computer Science (1979).
[7] D.E. Knuth, *The Art of Computer Programming. Vol. III, Sorting and Searching* (Addison-Wesley, Reading, MA, 1973).
[8] D.T. Lee and F.P. Preparata, Location of a point in a planar subdivision and its applications, *SIAM J. Comput.* **6** (1977) 594–606.
[9] W. Lipski, Jr. and F.P. Preparata, Segments, rectangles, contours, *J. Algorithms* **2** (1981) 63–76.
[10] R.J. Lipton and R.E. Tarjan, Applications of a planar separator theorem, *Proc. 18th Annual FOCS Symposium* (1977) 162–170.
[11] H.A. Maurer and T.A. Ottmann, Manipulating a set of points–a survey, in: M. Nagl and H.-J. Schneider, Eds., *Applied Computer Science 13* (Carl Hanser, Munich, 1979) 9–29.
[12] F.P. Preparata, A new approach to planar point location, University of Illinois at Urbana–Champaign, Coordinated Science Laboratory, Report R-829 (1978).
[13] M.I. Shamos, Geometric complexity, *Proc. 7th Annual ACM Symposium on Theory of Computing* (1975) 224–233.
[14] D.E. Willard, An introduction to super-B-trees, University of Iowa, Department of Computer Science (1979).