Information Processing Letters 16 (1983) 193–198
North-Holland Publishing Company 13 May 1983

AN IMPROVED ALGORITHM FOR THE FIXED-RADIUS NEIGHBOR PROBLEM *

Bernard CHAZELLE

Department of Computer Science, Brown University, Providence, RI 02912, U.S.A.

Communicated by Michael A. Harrison
Received 10 February 1982
Revised 2 November 1982

Keywords: Computational geometry, multidimensional searching, planar point location, planar graphs

1. Introduction

The planar fixed-radius near neighbor problem can be stated as follows: Preprocess a set P of N points in the plane so that all points of P lying within some fixed radius r of a new point can be listed effectively.

This problem has many practical applications in domains as varied as molecular graphics, statistics, air traffic control or data transmission [3]. Although a great deal of work has been done on the subject, the intricacy of the L_2 (Euclidean) metric has often led to consider the L_1 (Manhattan) or L_∞ metrics instead (where the locus of points within a fixed distance of a given point becomes a square), or to introduce simplifying assumptions on the density of points (e.g., sparsity conditions) [3]. In [1], however, Bentley and Maurer do examine the worst-case performance of our searching problem in the Euclidean metric, and they develop a *locus* method which is optimal for that criterion. More precisely, for any query point M, the A points of the set P less than r away from M can be reported in A + log N time. Unfortunately, the preprocessing involved becomes rapidly prohibitive as N grows, since the time P(N) required to organize the data into their structure and the size S(N) of this structure are both $O(N^3)$. It was, however, conjectured in [1] that clever use of an efficient planar point location algorithm could lead to a significant cut-down in the preprocessing work. Recall that the planar point location problem is that of, given a connected planar straight-line graph and a query point M, determining which region (face) of the graph contains M.

Several fast algorithms for planar point location are available in the literature [4,6,7,8,11], but because of its applicability to non-straight-line graphs, we will choose Preparata's algorithm [11] as the basis for our investigation. His method rests on a clever partitioning of the edges of the subdivision into O(log N) segments, and exploits the partial ordering among edges to construct a balanced search tree yielding logarithmic query time. The reason why, in both [1] and [11], it is suggested that the efficiency of a point location procedure might have great incidence on its fixed-radius neighbor counterpart can be easily seen. The N circles of radius r centered at the points of P subdivide the plane into regions which have the following property: To the question, "*which are the points of* P *less than* r *away from* M?", two points give the same answer if and only if they lie in the same region. In consequence, it appears that in order to report the neighbors of M, it is sufficient to locate M in the subdivision, provided that each region gives access to a list of the circles

* This research was sponsored by the defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory under Contract F33615-82-K-1539.

which contain it. Indeed, it is easy to see that if M lies in region f, point T of P is a neighbor of M if and only if f lies in the circle of radius r centered at T. This paper develops this argument, and shows how Preparata's method along with efficient range-search procedures [2,5,9] can be combined to produce an algorithm for the fixed-radius problem with the following features:

(1) *Query time*: $Q(N) = O(A + \log N)$.
(2) *Preprocessing time*: $P(N) = O(N^2 \log N)$.
(3) *Storage requirement*: $S(N) = O(N^2 \log N)$.

Before proceeding, we should observe that the major difficulty is that there may be as many as $\Omega(N^2)$ regions in the subdivision, each of them corresponding to a list of $\Omega(N)$ circles (i.e., $\Omega(N)$ neighbors). Therefore an explicit representation of all possible answers may, alone, require $\Omega(N^3)$ space, and must be avoided.

2. Setting up the data structure

Given N points $M_1, \ldots, M_N$, we apply a standard technique [12] to construct the planar graph G formed by the circles of radius r centered at these points. To obtain an adjacency list representation of G, we may first lay down the edges of the graph by taking each circle C in turn, and computing its intersections with the $N - 1$ other circles (details on this operation are given in Appendix A). This produces at most $2N - 2$ points, which after being sorted around C give exactly the edges contributed by the circle. Note that, in order to avoid having edges without endpoints (e.g., isolated circles), we may put a dummy vertex at the top and the bottom of each circle. This will also add the following property to the graph:

(i) All edges are curves which are single-valued in the Y-coordinate.

Another minor difficulty to solve is the ambiguity in the edge representation, if no provision is made for telling on which side of the circle the actual segment between two vertices A, B lies. One possible remedy is to represent the edge by the ordered pair (A, B), such that if C is the center of the circle that contributes the edge, the points A, B, C occur in clockwise order (see Fig. 1).

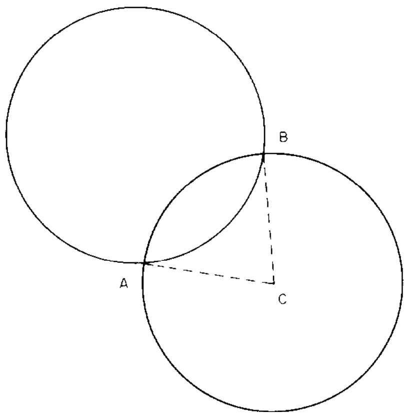

Fig. 1. Representing edges.

Next we can easily compute the adjacencies of the graph by first duplicating every edge, i.e., for every pair (A, B), adding the pair (B, A), then sorting the resulting list with respect to the first element of each pair. The order between points can be that of their abscissa, and in case of ties, that of their ordinate. It is easy to see that in its final state, the list contains, grouped together, all the edges adjacent to each vertex. We may then label the vertices, and we will have the complete adjacency lists of G. Since the graph has $O(N^2)$ edges, this operation requires $O(N^2 \log N)$ time and $O(N^2)$ space.

Actually, the planar location algorithm which we will use later on, relies on a *doubly-connected-edge-list* (DCEL) representation of G, whereby each edge (A, B) is represented by a 6-field node

$$[A, B, f_A, f_B, v_A, v_B].$$

f_B (resp. f_A) is the name of the region adjacent to AB which lies (resp. does not lie) in the circle supporting the edge (A, B), and v_A (resp. v_B) is a pointer to the edge which follows AB counterclockwise around A (resp. B) (see Fig. 2).

Details about this representation as well as a linear algorithm for converting an adjacency list representation into a DCEL can be found in [10]. The final requirement to be fully able to apply Preparata's point location algorithm is:

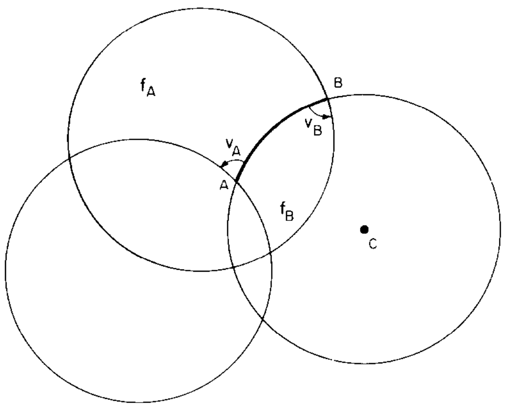

Fig. 2. The DCEL representation of a graph.

(ii) The discrimination of a point with any of the curves in the graph can be done in constant time.

This is trivially satisfied in the case of circles. Note that an efficient way of determining the position of a point M with respect to a circle passing through A, B involves keeping the center C of the circle along with the edge (A, B), now having 7-field nodes,

$$[A, B, C, f_A, f_B, v_A, v_B],$$

in the DCEL. With this added information, a simple evaluation of the distance CM is sufficient to determine the position of M with respect to the circle.

We can now set up the search tree for G described in [11], which requires $O(N^2 \log N)$ time and space and we will be in a position to locate an arbitrary point M in O(log N) time. Next we need some mechanism to allow us to report the circles overlapping the region where M lies. We have already ruled out the solution which consists of having a pointer from each region to the list of its overlapping circles, because of its excessive cost. Instead, we introduce a new structure H.

3. Towards a better search structure

H is an acyclic directed graph whose vertices are the faces of G and whose edges are labeled and defined algorithmically as follows: for each node $[A, B, C, f_A, f_B, v_A, v_B]$ in turn, if f_A has not been assigned an outgoing edge already, set an edge from f_A to f_B, and label the edge with the name of the circle supporting A, B (see Fig. 3).

Note that the graph H is well-defined, yet not uniquely defined. It is clear that the adjacency lists for H can be set up in time linear in the number of 7-tuples, i.e., $O(N^2)$. A closer examination of the regions f_i lead to single out a particular type of shape called *holly*, which lies at the very basis of the algorithm developed later on.

Definition. A face f_i is a *holly* if all its bordering edges are concave (see Fig. 4).

Note that the outside, unbounded face, f_0, is always a holly. Expressed in terms of the graph H, a holly is simple a vertex with incoming edges only. Our first observation is that all the vertices of H have exactly one outgoing edge, except for the hollies, which have none. As a result, there is a unique path emanating from every vertex. If we follow this path, we may notice that we never enter any circle, and we leave only those circles indicated by the labels of the edges on the path. Therefore, each face visited can never be re-entered later on, since this would entail re-entering a circle

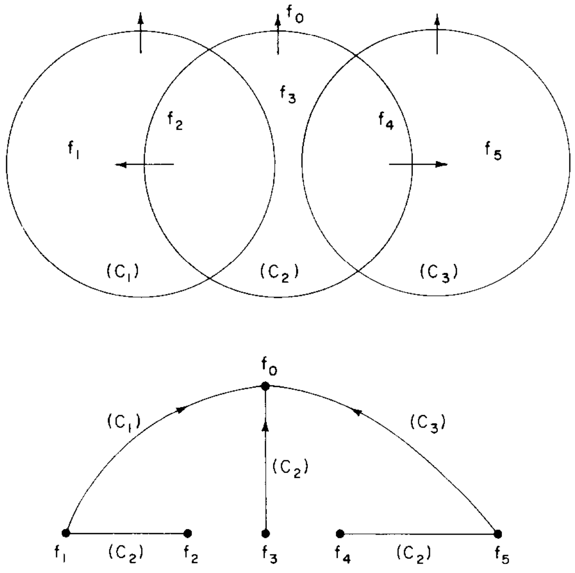

Fig. 3. The graph H.

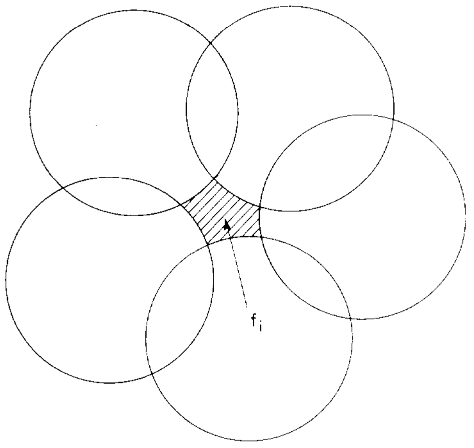

Fig. 4. An example of a *holly*.

we previously left. As a result, H is acyclic, and any path leads to a holly.

It follows that in order to determine all the circles containing a point M, we may first locate the face of G that contains M, then follow the path in H from this face, listing the circles indicated as labels of the edges traversed along the way. This enumeration corresponds to listing the circles which we are leaving. At the end of the traversal (i.e., in a holly), the last circles which remain to be listed are exactly all those containing the holly. Note that if the holly is the unbounded face f_0, we are finished.

However alluring, we must resist the temptation of having a list of all the circles containing each holly, for we may have a prohibitive $\Omega(N^2)$ number of hollies (see Appendix A). Instead, we observe that since all circles have same radius, any circle which contains a holly must intersect every circle adjacent to the holly. To make this necessary condition useful, however, we must make it sufficient.

Let Γ be any circle adjacent to our bounded holly h, with v_1 one endpoint of the common arc. If Δ is a circle intersecting Γ, we can define its Γ-*interval* as the portion of the boundary of Γ which lies inside Δ. Thus it appears that a circle contains h if and only if its Γ-interval is non-empty and contains v_1 (see Fig. 5).

With this observation in hand, it becomes possible to preprocess Γ so that the B circles which

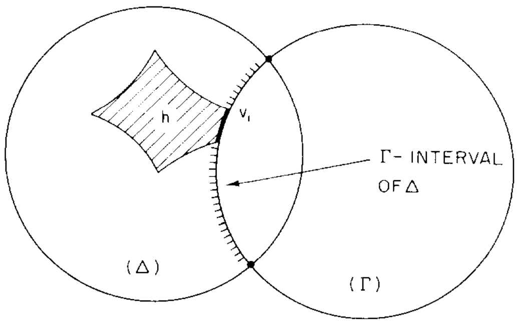

Fig. 5. Characterizing the inclusion of a holly in a circle.

contain h can be reported in $O(B + \log N)$ time. Indeed, we are reduced to a one-dimensional range-query problem, and several efficient algorithms are at our disposal. The problem is, in its form of interest to us, that of reporting the B segments containing a query point M, given a set of N collinear segments. Partly because of its relevance for VLSI design rule checking, this problem has been given considerable attention recently, and several algorithms have been produced, giving an optimal $O(B + \log N)$ query time, with some preprocessing work entailing between $O(N)$ and $O(N \log N)$ time and space costs [2,5,9].

To make the analogy complete, we must choose an origin O_Γ on each circle Γ, corresponding to a 0 polar angle. In this manner, each point on the circle can be defined uniquely by its polar angle, measured between 0 and 2π. A Γ-interval which does not contain the origin is then mapped to a single angular interval. Otherwise, we simply break up the Γ-interval into two parts, with the origin O_Γ as the common endpoint.

We are now in a position to apply any of the known range-query algorithms mentioned above, and report all the containing circles that had not been yet listed. The overhead for setting up whichever range-query data structure is chosen amounts at worst to $O(N \log N)$ for each circle; therefore, the entire preprocessing remains within $O(N^2 \log N)$, which proves our earlier claim.

4. Conclusions

The algorithm which we have presented is a good illustration of how the combined use of powerful tools for seemingly unrelated problems can often be fruitful. Three methods for central problems of computational geometry – locus, range-query, planar point location – are instrumental in the working of the algorithm, and to a large extent, its efficiency owes to theirs. Extensions of the algorithm include the possibility of passing the radius as a parameter, choosing other, non-Euclidean metrics, supporting insertion and deletion of points, and of course, generalizing to higher dimensions.

Appendix A

A.1

One point which we have so far left aside concerns the complexity of the arithmetic computations involved in the algorithm. Since we do not have to evaluate distances per se, but simply compare them, we do not need square roots and the four operations $+$, $-$, $\times$, $\div$ seem sufficient. They are so, indeed, if we do not account for the task of intersecting circles, required at the beginning of the algorithm. For this, we can show that two invocations of the square root operation per intersection calculation are actually sufficient.

In the following, the pair (r_1, r_2) will respectively denote the X- and Y-coordinates of a point R. Let A, B be the centers of two intersecting circles of radius r, with C, D denoting the two intersection points (see Fig. 6). Let Ω be the segment AB's mid-point and δ be the length of AB.

Defining a new orthonormal system of coordinates,

$$(\Omega, \Omega X', \Omega Y'),$$

with $\Omega X'$ in the direction of AB, we can easily express the old coordinates (m_1, m_2) of a point M in terms of its new coordinates (m'_1, m'_2):

$$\begin{pmatrix} m_1 \\ m_2 \end{pmatrix} = \begin{pmatrix} \Omega_1 \\ \Omega_2 \end{pmatrix} + \frac{1}{\delta}\begin{pmatrix} b_1 - a_1 & a_2 - b_2 \\ b_2 - a_2 & b_1 - a_1 \end{pmatrix}\begin{pmatrix} m'_1 \\ m'_2 \end{pmatrix}.$$

In consequence, we may express the points C, D in terms of their new coordinates, which involves the use of the second square root operation

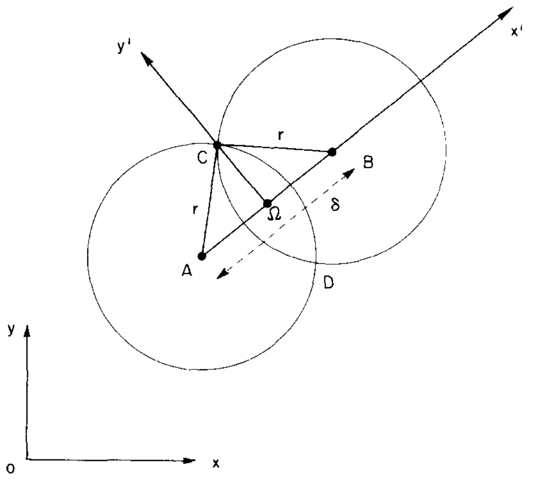

Fig. 6. Intersecting two circles.

announced above:

$$\begin{cases} c'_1 = d'_1 = 0, \\ c'_2 = d'_2 = \pm\sqrt{r^2 - \frac{1}{4}\delta^2}. \end{cases}$$

A.2

We justify here the use of sophisticated range-query structures by displaying an arrangement of points which gives $\Omega(N^2)$ hollies (see Fig. 7). It is therefore prohibitive simply to set up a list of containing circles for each holly.

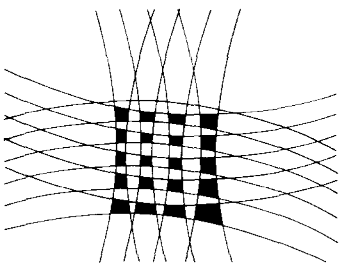
Fig. 7. A quadratic number of hollies.

References

[1] J.L. Bentley and H.A. Maurer, A note on Euclidean near neighbor searching in the plane, Inform. Process. Lett. 8 (1979) 133–136.

[2] J.L. Bentley and T. Ottmann, Algorithms for reporting and counting geometric intersections, IEEE Trans. Comput. C-28 (9) (1979).

[3] J.L. Bentley, D.F. Stanat and E.H. Williams Jr., The complexity of finding fixed-radius near neighbours, Inform. Process. Lett. 6 (1977) 209–212.

[4] D.P. Dobkin and R.J. Lipton, Multidimensional searching problems, SIAM J. Comput. 5 (2) (1976) 181–186.

[5] H. Edelsbrunner, A time- and space-optimal solution for the planar all-intersecting-rectangles problem, Tech. Rept., Technische Universitat Graz, 1980.

[6] D.G. Kirkpatrick, Optimal search in planar subdivisions, Detailed abstract, University of British Columbia, Canada, 1980.

[7] D.T. Lee and F.P. Preparata, Location of a point in a planar subdivision and its applications, Proc. 8th SIGACT Symp., Hershey (1976) pp. 231–235.

[8] R.J. Lipton and R.E. Tarjan, Applications of a planar separator theorem, Proc. 18th IEEE FOCS Symp., Providence (1977) pp. 162–170.

[9] E.M. McCreight, Efficient algorithms for enumerating intersecting intervals and rectangles, Tech. Rept. Xerox PARC, CSL-80-9, 1980.

[10] D.E. Muller and F.P. Preparata, Finding the intersection of two convex polyhedra, Theoret. Comput. Sci. 7 (1978) 217–236.

[11] F.P. Preparata, A new approach to planar location, SIAM J. Comput. 19 (3) (1981).

[12] A.A.G. Requicha, Representations for rigid solids: Theory, methods and systems, ACM Comput. Surveys 12 (4) (1980) 437–464.