THE LOCUS APPROACH

Mark H. Overmars

RUU-CS-83-12
July 1983

Rijksuniversiteit Utrecht

Vakgroep informatica

Budapestlaan 6
Postbus 80.012
3508 TA Utrecht
Telefoon 030–53 1454
The Netherlands

THE LOCUS APPROACH

Mark H. Overmars

Technical Report RUU-CS-83-12
July 1983

Department of Computer Science
University of Utrecht
P.O. Box 80.012
3508 TA Utrecht
the Netherlands

THE LOCUS APPROACH*

Mark H. Overmars, University of Utrecht, the Netherlands

Abstract

A number of problems in Computational Geometry are solved using a technique that is known as "locus approach". This technique will be considered in detail. First the general idea behind the method and the necessary conditions a problem should satisfy to be solvable using the locus approach are described. Next the technique is demonstrated by means of some (well-known) examples. In the final part of the paper some general results are stated, showing that whole classes of searching problems can be solved using the locus approach.

1. Introduction

An important class of problems in Computational Geometry are searching problems. A searching problem is a problem in which a question (query) is asked about an object with respect to a set of objects (points). Some prime examples of geometric searching problems are listed below.

(i) Nearest neighbor searching

Given a finite set V of points in R^d and another point x in R^d, we wish to report a point y in V with smallest distance (in, say, the Euclidean metric) to x.

---

* This research was supported by the Netherlands Organization for the Advancement of Pure Research.

(ii) Range searching

Given a finite set of points in R^d and a "range" $([a_1..b_1], \ldots, [a_d..b_d])$, we wish to report, or just count, all points $y=(y_1, \ldots, y_d)$ in the set V that lie within the range, i.e., with $a_1 \leq y_1 \leq b_1$ and ... and $a_d \leq y_d \leq b_d$.

(iii) Rectangle intersection searching

Given a finite set V of rectilinearly oriented hyper-rectangles in R^d and another such rectangle x, determine all rectangles in V, or just their number, that intersect x, i.e., have at least one point in common with x. Other rectangle searching problems like the rectangle containment searching problem, that asks for all rectangles in V that are contained in x, can be defined as well.

(iv) Polygon intersection searching

Given a finite set V of polygons (or polyhedra) in R^d with some bounded number of corners and edges and another such polygon x, determine all polygons in V that intersect x. Note that both the range searching problem and the rectangle intersection searching problem are special instances of the polygon intersection searching problem.

(v) Convex hull searching

Given a finite set of points in R^d and another point x, determine whether x lies inside or outside the convex hull of the set.

Solving a searching problem consists of devising a data structure S to represent the set of objects, such that queries with different query objects can be answered efficiently. Such a data structure S can be static or dynamic. A static structure only allows for queries while a dynamic structure also allows for insertions and deletions of objects. In this paper we will only consider static data structures for searching problems. (For an overview of techniques for obtaining dynamic data structures see e.g. Overmars [10].) The efficiency of a static structure S is given by

means of three measures: the time needed to perform a query $Q_S(n)$, the time required to build S $P_S(n)$ and the amount of memory required for storing S $M_S(n)$, where n is the number of points in the set. In this paper we are in general only interested in obtaining low query times. The building time and the amount of storage required for our structures is sometimes tremendously high. The results should hence be regarded as being mainly theoretical and not meant for practical use.

In this paper we consider one basic technique for obtaining static data structures for searching problems, the "locus approach". This method is based on a transformation that maps searching problems into one basic searching problem, the region location problem.

(vi) Region location

Given a finite set V of non-intersecting polyhedra in R^d and a point x in R^d, determine the polyhedron in V p lies in (if any).

In Section 2 we will describe the kind of transformations used and show how the region location problem can be solved efficiently. In Section 3 we give a number of examples and show how the locus approach can be used for solving e.g. the nearest neighbor searching problem and the range searching problem. In Section 4 we consider the kind of properties a searching problem should satisfy to be solvable by means of the locus approach. It leads to some classes of searching problems that can be solved efficiently (with respect to the query time) using the locus approach.

2. Locus approach and region location

Assume we have a searching problem such that, given a fixed finite set V, the number of different answers to queries over V is finite. For example, for the nearest neighbor searching problem only points of the set can occur as answers and hence, the number of different answers is bounded by the size of the set, i.e., n. For the range searching problem the number of answers is bounded by n^{2d} (for more precise results see

Saxe [11]). A searching problem that clearly does not satisfy this assumption is the searching problem that, given a set of points V and another point x, asks for the distance between x and its nearest neighbor in V.

Definition. Given some searching problem PR, the *query space* of PR is the space from which the query objects can be choosen.

Assuming that query objects are described by a bounded number of reals, the query space of a searching problem is always a subset of R^{dq} where d_q is the dimension of the query space. For example, the query space of the d-dimensional nearest neighbor searching problem is R^d. For the d-dimensional range searching problem, query objects are ranges $([a_1..b_1], \ldots, [a_d..b_d])$, in which $a_i \leq b_i$. It follows that the query space is a part of R^{2d}, namely that part in which each odd numbered coordinate is smaller than the next coordinate.

Given a fixed, finite set V for a searching problem PR (that we assumed to have a finite number of different answers), it divides the query space of PR in a number of areas of constant answer, i.e., given two points in the same area, the answers to the queries with those two points over V are equal. An area is assumed to be connected. Hence, it is possible that more than one areas correspond to the same answer. This leads to the following technique for solving PR, the *locus approach*: As preprocessing determine all areas of constant answer and store with each area the corresponding answer. To answer a query, determine the area the query point lies in and report the corresponding answer. This technique is only sensible when (a) the number of areas is finite, and (b) the areas are reasonably shaped. We will assume that these areas are polyhedral and that the number of bounding edges is bounded by $O(n^k)$, where n is the number of points in the set and k is a constant depending on the type of searching problem.

Hence the locus approach reduces the searching problem PR to the region location problem. The region location problem has received considerably attention recently (see e.g. [2, 3, 4, 6, 7, 9]). We will recall a result by Dobkin and Lipton.

Theorem. ([2]) The d-dimensional region location problem can be solved in O(log N) query time, where N denotes the number of edges in the subdivision.

proof

We will only briefly describe the case in which d=2. For larger d an extension of this method can be used. For more details see [2]. To solve the 2-dimensional region location problem we draw a vertical line through each corner of the subdivision. In this way we divide the plane in O(N) slabs (see figure 1). Clearly, in O(log N) time one can determine the

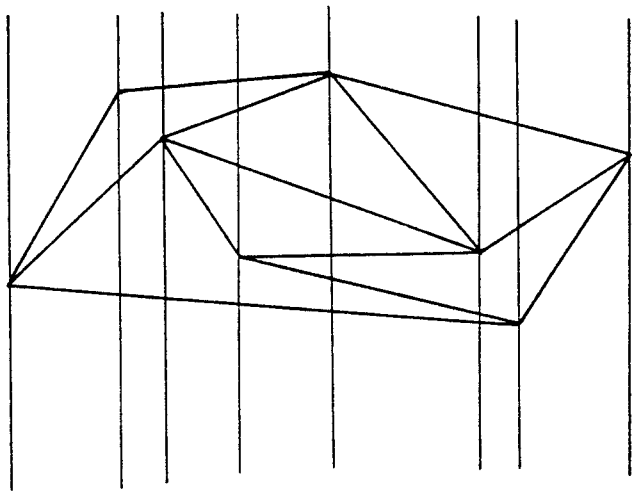

Figure 1.

slab a given query point lies in. In each slab we order the edges from top to bottom and store them in a balanced search tree. In this way we can determine in another O(log N) steps between which two edges the query point lies, and hence, in which region it lies.

□

Corollary. When a searching problem can be solved using the locus approach it can be solved in O(log n) time, where n is the number of objects in the set.

proof

The number of edges in the subdivision is bounded by $O(n^k)$. Hence, the query time is bounded by $O(\log n^k) = O(\log n)$ for k is constant.

□

Although the query time of the method described is very low, the pre-processing time and the amount of memory required can become very high. Only in the 2-dimensional case space-efficient solutions to the region location problem are known ([3, 6]).

3. Some examples

In this section we will show how the locus approach, described in the preceeding section, can be used for solving a number of searching problems.

(i) Nearest neighbor searching

Let us first consider the planar case. For each point p in the set V, the area of the query space (which is the plane itself) for which p is the corresponding answer is a polygon located around p. The subdivision of the plane in areas of constant answer is the so-called Voronoi diagram (see e.g [12]). See figure 2 for an example. The number of edges in a

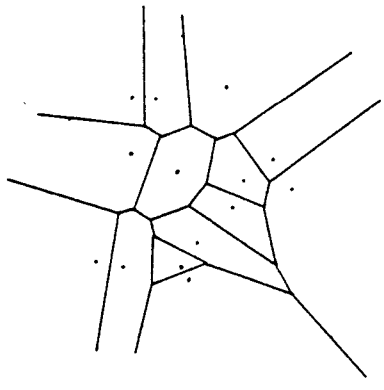

Figure 2.

Voronoi diagram is bounded by $O(n)$. Hence, the 2-dimensional nearest neighbor searching problem can be solved in $O(\log n)$ time. This result

can easily be extended to hold for the d-dimensional nearest neighbor searching problem as well. It also holds when other measures of distance are used.

(ii) Range searching

Let us first consider the 1-dimensional range searching problem, i.e., we are given a set of points on a line and we ask for those points that lie within a given segment [a..b]. The query space is a part of R^2. For each point p in the set, p is reported when $p \geq a$ and $p \leq b$, hence, when the range lies in the area of the query space shown in figure 3.a. For each point in the set there is such an area. These n areas divide the query space in $O(n^2)$ areas of constant answer (see figure 3.b). Each of these

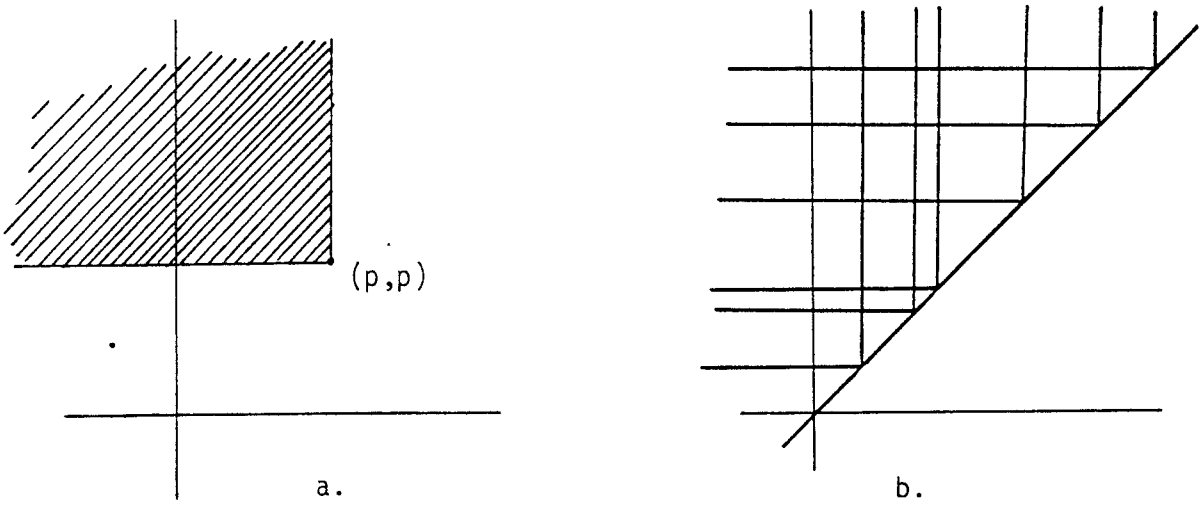

Figure 3.

areas is either a rectangle (possibly unbounded) or a triangle. Applying the locus approach we obtain a query time of O(log n) (not counting the time required for reporting the answer).

In the d-dimensional case we get a division of the 2d-dimensional query space into $O(n^{2d})$ areas of constant answer. Hence, we can again use the locus approach to solve the problem in O(log n) query time.

(iii) Rectangle intersection searching

It has been shown that the d-dimensional rectangle intersection searching problem can be solved using the 2d-dimensional range searching

problem (Lee and Wong [8]). Hence, the rectangle intersection searching problem can be solved within O(log n) query time using the locus approach. Similar results exist for other rectangle searching problems like the rectangle containment searching problem (Edelsbrunner and Overmars [5]).

(iv) Polygon intersection searching

Although for the general polygon intersection searching problem the query space can be divided into a finite number of areas of constant answer, it is not always possible to use the locus approach, for these areas need not be polyhedral. For some special cases the problem can be solved using the locus approach, for example when the query polyhedron consists of a point.

(v) Convex hull searching

Clearly, the convex hull searching problem can be solved using the locus approach. The query space is divided into two areas, the inside and the outside of the convex hull of the point set.

4. Classes of problems.

In this section we will describe some classes of problems that can be solved using the locus approach. As a result, one only has to prove that a searching problem belongs to one of these classes to show that it is solvable in O(log n) query time using the locus approach.

A searching problem can be described as a function that maps query objects of type T_1 and sets of objects of type T_2 into answers of type T_3. We can denote a searching problem PR as

$$PR: T_1 \times 2^{T_2} \rightarrow T_3.$$

In the subdivision of the query space, each bounding hyper-plane gives rise to an inequality that is linear in the query object. The answer to all these inequalities uniquely determines the region a query point lies in. This observation leads to the following result:

Theorem. A searching problem PR is solvable using the locus approach iff

$$\exists_k \forall_{V \in 2^{T_2}} \exists_{i \leq |V|^k} \exists_{f_1,\ldots,f_i} \exists_{\square} \forall_{x \in T_1}$$
$$PR(x,V) = \square(V, f_1(x), \ldots, f_i(x))$$

, where the f_i's are linear inequalities in x.

proof

$\Rightarrow$: The functions f_j are the bounding hyper-planes in the subdivision. Their number must be bounded by $|V|^k$ for some constant k depending on the type of searching problem. The function $\square$ associates the answer to a particular region in the subdivision,

$\Leftarrow$: Trivial.

□

It is very hard to verify whether a searching problem satifies this condition. Therefor we will restrict our class of searching problems in such a way that checking whether a problem belongs to the class is easier.

Theorem. A searching problem PR is solvable using the locus approach when

$$\exists_{f,\square} \forall_{V \in 2^{T_2}} \forall_{x \in T_1}$$
$$PR(x,V) = \square(V, f(x,p_1), \ldots, f(x,p_n))$$

, where $V = p_1,\ldots,p_n$ and $f(x,p_i)$ can be computed by linear inequalities in x.

As an example let us consider the range counting problem that asks for the number of points in a set V that lie within a range x. Then we take $f(x,p_i)=1$ if x contains p_i and $f(x,p_i)=0$ otherwise. $\square$ just adds up the values of $f(x,p_1)$, ... , $f(x,p_n)$.

An important class of searching problems are the "decomposable searching problems" (defined by Bentley [1]). An extension was given in [10]:

Definition. A searching problem PR is called a C(n)-*decomposable searching problem* when for any partition AUB of the set V:

$$PR(x,V) = \square(PR(x,A),PR(x,B))$$

, where $\square$ takes $C(n)$ time to compute when V contains n points.

Numerous searching problems are $C(n)$-decomposable for some function $C(n)$. From the definition it follows that for each $C(n)$-decomposable searching problem there exists a function $\square'$ such that

$$PR(x,V) = \square'(PR(x,\{p_1\}), \ldots ,PR(x,\{p_n\}))$$

, where $V = \{p_1, \ldots ,p_n\}$. Hence, we obtain the following result:

Theorem. Each $C(n)$-decomposable searching problem PR for which $PR(x,\{p\})$ can be solved by means of linear inequalities in x, can be solved in $O(\log n)$ query time, using the locus approach.

5. Conclusions and extensions

We have shown that a large number of searching problems can be solved with very low (in fact often optimal) query time using one general method, the locus approach. This result is in general only interesting from theoretical point of view. The amount of preprocessing and storage required is often tremendously high. Moreover the structures we obtain are static. Only in the two-dimensional case the locus approach is sometimes a practical solution to a searching problem.

An important open problem that remains is how the technique can be adapted to the case in which the subdivision of the query space is not polyhedral. It asks for a solution of the region location problem in which the regions are bounded by algebraïc curves rather than by hyperplanes. Some solution to this problem are known in the case d=2 ([3, 4]) but in higher-dimensional space no results do exist yet.

6. References

1 Bentley, J.L., Decomposable searching problems, Inform. Proc. Lett. 8 (1979) 244-251.

2 Dobkin, D.P. and R.J. Lipton, Multidimensional searching problems, SIAM J. Computing 5 (1976) 181-186.

3 Edelsbrunner, H., An optimal solution for searching in general planar subdivisions, inst. for Information Processing, T.U. Graz, 1983, to appear.

4 Edelsbrunner, H. and H.A. Maurer, A space-optimal solution of general region location, Theor. Comp. Science 16 (1981) 329-336.

5 Edelsbrunner, H. and M.H. Overmars, On the equivalence of some rectangle problems, Inform. Proc. Lett. 14 (1982) 124-127.

6 Kirkpatrick, D.G., Optimal search in planar subdivision, SIAM J. Computing 12 (1983) 28-35.

7 Lee, D.T. and F.P. Preparata, Location of a point in a planar subdivision and its applications, SIAM J. Computing 6 (1977) 594-606.

8 Lee, D.T. and C.K. Wong, Finding intersections of rectangles by range search, J. Algorithms 2 (1981) 337-347.

9 Lipton, R.J. and R.E. Tarjan, Applications of a planar separator theorem, Proc. 18th Annual IEEE Symp. on Foundations of Computer Science, 1977, 162-170.

10 Overmars, M.H. The design of dynamic data structures, Springer Lect. Notes in Comp. Science Vol. 156, Springer-Verlag, Heidelberg, 1983.

11 Saxe, J.B., On the number of range queries in k-space, Discrete Applied Math. 1 (1979) 217-225.

12 Shamos, M.I. and D. Hoey, Closest-point problems, Proc. 16th Annual IEEE Symp. on Foundations of Computer Science, 1975, 151-162.