SOME TECHNIQUES FOR GEOMETRIC SEARCHING WITH IMPLICIT SET REPRESENTATIONS

Bernard Chazelle

CS-TR-095-87

June 1987

SOME TECHNIQUES FOR GEOMETRIC SEARCHING WITH IMPLICIT SET REPRESENTATIONS

by

Bernard Chazelle

Department of Computer Science
Princeton University
Princeton, New Jersey 08544

Abstract:

There are many efficient ways of searching a set when all its elements can be represented in memory. Often, however, the domain of the search is too large to have each element stored separately and some implicit representation must be used. Whether it is still possible to search efficiently in these conditions is the underlying theme of this paper. We look at several occurrences of this problem in computational geometry and we propose various lines of attack. In the course of doing so, we improve the solutions of several specific problems; for example, computing order statistics, performing polygonal range searching, testing algebraic predicates, etc.

This research was supported in part by the National Science Foundation under Grant MCS83-03925.

Categories and Subject Descriptors: E.1 [**Data**]; Data Structures, F.2.2 [**Analysis of Algorithms**]; Nonnumerical Algorithms and Problems

General Terms: Algorithms, Computational Geometry

Additional Key Words and Phrases: Searching, Order Statistics, Implicit Data Structures

Author's address: Department of Computer Science, Princeton University, Princeton, NJ 08544.

1. Introduction

We present a number of new results in computational geometry, all of which in their own way address the same basic question: How can a set be searched efficiently if it cannot be explicitly represented in memory? This problem arises in situations where searching is defined over a domain that is represented by an implicit description and not by each of its elements.

The problem of computing the k^{th} largest element in the set $X + Y = \{x + y \mid x \in X, y \in Y\}$, where X and Y are two sets of n integers each, illustrates this point. Phrased as a selection problem, it can be solved in $O(n^2)$ time by first expanding the set $X + Y$ in its entirety and then applying a linear-time selection algorithm [Bl]. It is well-known, however, that the solution can be found in $O(n \log n)$ time [Sh]. We will see in this paper that computing order statistics in Euclidean space raises problems of a similar, though more complex, nature.

A different, but related, phenomenon occurs when a collection of geometric objects must be preprocessed to facilitate the answering of specific queries. The so-called *locus approach* is to form the set of all possible answers in preprocessing and then search among them to find the answer. As should be expected, such a solution is often too costly to be contemplated. Sometimes, the problem has a simple enough structure that an implicit encoding of the search domain is sufficient to allow fast answers. For example, orthogonal range searching among n points in E^d yields a search domain of size $O(n^{2d})$ via the locus approach, but it is well-known [PS] that the problem can be solved very efficiently.

Often, however, the search domain grows polynomially in the size of the output, and yet no truly efficient methods have been found. Given n lines in the plane, determining whether a query point passes through at least one of them is such a problem. The lines create a subdivision of the plane into $O(n^2)$ regions so, using point location, the problem is solvable in quadratic space and logarithmic time. On the other hand, $O(n^{.667})$ time is the best worst-case query time achievable to date [HW], if only $O(n)$ space may be used. Whether this problem is in the class $PLOG$, that is, can be solved in $O(n \log^a n)$ space and $O(\log^b n)$ query time, for two constants $a, b \geq 0$, is still unknown.

This paper proposes various lines of attack to deal with the questions raised above. In Section 2, we take a sample of problems representative of a large class of geometric selection problems. We prove that the k^{th} longest bridge between two convex polygons can be found in linear time (a bridge is a segment that intersects each polygon in one vertex). We show that selecting the k^{th} largest interdistance between n points can be done nearly as fast as finding the largest one. We also give an algorithm for selecting the k^{th} smallest-area triangle among n points in the plane within almost the same time bound as for getting the smallest one.

In Section 3, we turn to the problem of organizing n points in the plane so that reporting which points fall within a query triangle can be done efficiently. Although expanding the search domain of this problem requires $O(n^6)$ space (consider all triples of positions of lines), we can show that

$O(n \log^2 n)$ space is actually sufficient for fast retrieval provided that the angles of the triangle can be bounded below by a constant.

Section 4 is concerned with the problem of determining if among n geometric objects a subset of them of fixed size satisfies a given property. We show that the naive algorithm can always be improved — if only by a small margin. The speed-up achieved is only of theoretical interest, but it is very general.

This paper should be regarded as a collection of techniques for dealing with implicit set searching. These can be encapsulated in a few simple rules:

1. Investigating the combinatorial structure of a geometric problem and stripping it from its *geometry* by appropriate transformations (Section 2).
2. Identifying the *hard case* of a problem and making minimal assumptions to rule it out and then solving the remaining problem efficiently (Section 3).
3. Checking whether the inner loop of an algorithm can be sped up with a table look-up mechanism (Section 4).

2. Computing Order Statistics in Euclidean Space

Using classical terminology, we call *selection* the operation of computing the k^{th} largest element in a totally ordered set. The complexity of selection has been shown to be linear in the size of the input set [Bl]. As is well-known, however, this result is not necessarily optimal when the set is defined implicitly. For example, selection in $X + Y$ can be done in $O(n \log n)$ time, where n is the input size [FJ1,FJ3,JM,Sh], and in $O(n)$ if X and Y are already sorted [FJ3,MA].

Other cases of implicit sets for which nontrivial bounds have been achieved include matrices with sorted rows and/or columnns [FJ1,FJ3,GM]. The motivation for studying such problems comes from operations research [GM], location theory (e.g. finding p-centers [FJ2,Me]) and statistics (e.g. computing the Hodges-Lehmann estimator [Sh]). Other examples of searching a large domain without explicit representations can be found in [Co2,M].

Selection in geometric sets so far has been mostly confined to extremal problems, i.e., for the cases $k = 1$ or $k =$ "cardinality of the set". In this category, we find the following, widely studied problems: given a finite point set $S \subseteq E^2$, compute the smallest or largest inter-point distance [BS,Y], or the smallest triangle with vertices in S [CGL,EOS]; given two convex polygons P and Q, compute the minimum distance between them [CW,MT,T], etc. In this work, we generalize these questions into selection problems and prove nontrivial upper bounds on their complexity. We show that, surprisingly, selection often is barely more difficult than the corresponding extremal problem.

Our emphasis will be in the techniques used rather than in the results themselves. The rationale is that a considerable number of selection problems can be formulated, but not many differ fundamentally in the way they are solved. Indeed, our techniques can be used to solve many others. In all

cases, we exploit the geometric components of the problems at the outset and then focus exclusively on their combinatorial aspects.

2.1. Preliminaries

Let S be a totally ordered set partitioned into m subsets $S_1, \ldots, S_m$. We assume that each set S_i is stored in a linear array, with the elements appearing in increasing order. It is easy to adapt a technique of Jefferson, Shamos and Tarjan [Sh] to compute x, the k^{th} largest element of S in $O(m \log \frac{|S|}{m})$ time. Assume that $k \geq \frac{1}{2}|S|$; briefly, the method involves shrinking each S_i by discarding elements from the left and/or right end. The general step consists of computing the lower quartile y_i in each S_i, which is easily done in $O(m)$ time. Then, also in $O(m)$ steps, we find the weighted lower quartile z of $\{y_1, \ldots, y_m\}$, with each y_i assigned the current size of S_i as weight. We easily find that $z \leq x$, therefore at least a quarter of the elements in each S_i for which $y_i \leq z$ can be discarded. This allows us to remove at least $\frac{1}{16}$th of the elements in S from further consideration, which leads to the claimed $O(m \log \frac{|S|}{m})$ upper bound. This method was improved in [FJ1], from which we quote the following result.

Lemma 1. (*Frederickson and Johnson*) – Let S be a totally ordered set partitioned into m sorted subsets $S_1, \ldots, S_m$. It is possible to compute the k^{th} largest element in S in $O(m + p \log \frac{k}{p})$, where $p = \min(k, m)$. This complexity is optimal.

Suppose now that S consists of the elements of an $n \times m$ matrix $(1 < m \leq n)$, each of whose rows and columns appears in nondecreasing order. Frederickson and Johnson [FJ3] have described an $O(m \log \frac{2n}{m})$ time algorithm for computing the k^{th} largest element in S. Unfortunately this result is a little too restrictive for our purposes. In the following lemma, we extend these methods to handle a more general class of matrices.

Let A be an $n \times n$ matrix $(a_{i,j})$. Each $a_{i,j}$ is either a real number or an empty entry (called a *blank*). We say that A is *monotone* if the real entries in each row of A form an interval (possibly empty) of the form $a_{i,j}, a_{i,j+1}, \ldots, a_{i,k}$, with $a_{i,j} < a_{i,j+1} < \ldots < a_{i,k}$. The equivalent property must also hold with respect to each column (Fig.1). The matrix A is said to be represented in *standard form* if

1. it is stored in $O(n)$ space,
2. each entry $a_{i,j}$ can be computed in constant time, and
3. there is an $n \times 2$ array, called the *map* of A, which indicates for each row the indices of their first and last real entries.

Our monotone matrices resemble the *sorted* matrices of [FJ3], except for the presence of blanks. The difficulty in trying to convert them into sorted matrices is that assigning numerical values to blanks may not allow both rows and columns to be sorted simultaneously. We go around this problem by using ideas from [FJ3,MA] to develop a new algorithm.

Lemma 2. Let c be a positive integer and $A_1, \ldots, A_c$ be a collection of $n \times n$ monotone matrices given in standard form, and let S be the set of real entries. The k^{th} largest element in S can be computed in $O(n)$ time.

Proof: As a starter, consider the case $c = 1$, and let A be the unique matrix under consideration. For each row $i = 1, \ldots, n$, let l_i (resp. r_i) indicate the index of the first (resp. last) real entry. We set $l_i = r_i = 0$ if row i has no real entry. We augment A with two columns of blanks, one with index 0 and the other with index $n+1$. We treat each blank of row i as $-\infty$ for the *positions* left of l_i and $+\infty$ for the positions right of r_i. If $l_i = r_i = 0$ then all the entries of row i are $+\infty$, except for the entry with index 0, which is treated as $-\infty$. We define the *trace* of a real z, denoted $T(z)$, as the set of its *positions* among the rows of A. We have $T(z) = \{t_1, \ldots, t_n\}$, where t_i is the unique index such that $a_{i,t_i} \leq z < a_{i,t_i+1}$; note that $t_i = 0$ if $l_i = 0$. Next we show that $T(z)$ can be computed in $O(n)$ time. We successively describe the algorithm, prove its correctness, and then establish its complexity.

Let t be initialized to the value n. For $i = 1, 2, \ldots, n$, perform the following case-analysis:

1. $z < a_{i,l_i}$: set t_i to $l_i - 1$.
2. $z > a_{i,r_i}$: set t_i to r_i.
3. $a_{i,l_i} \leq z \leq a_{i,r_i}$: if either $t < l_i$ or $t > r_i$ then set t to r_i, else leave t unchanged for the time being. In all cases, compare z and $a_{i,t}$, next, and decrement t by one as long as $z < a_{i,t}$. Finally, set t_i to t.

Correctness is easily proven by induction. Whenever t is updated, it ends up pointing either to an entry equal to z or to one having to its right a real entry greater than z. Therefore if $a_{i,l_i} \leq z \leq a_{i,r_i}$ and $l_i \leq t \leq r_i$, by monotonicity, $a_{i,t+1}$ is greater than z, so the search may proceed towards the left. We will show now that the running time is $O(n)$. Clearly it is sufficient to show that if for i, j, h $(i < h)$ the four entries $a_{i,j}, a_{i,j+1}, a_{h,j}, a_{h,j+1}$ are real, then not all four are examined during the computation (to see this, mark off the leftmost and rightmost entries visited in each row). Assume that they are; since at row i, the variable t takes on at least one value strictly less than $j+1$ and at row h it takes on the value $j+1$, it must be reset to a value greater than j at some row β; $i < \beta \leq h$ (this can only occur at step 3). Let α be the largest index u less than β such that $a_{u,l_u} \leq z \leq a_{u,r_u}$. Since both $a_{i,j}$ and $a_{i,j+1}$ are examined, we have $i \leq \alpha$. The following

derivations use the monotonicity of A repeatedly. Since the entries a_{α,t_α} and a_{β,t_α} are respectively real and blank, the entry a_{h,t_α} is blank. Since $t_\alpha < j$ and a_{h,t_h} is real (at least two values in row h are examined), we have $a_{h,t_h} \leq z$ and $t_\alpha < t_h$. Since $a_{\alpha,t_\alpha+1}$ is greater than z, so are a_{α,t_h} and hence a_{h,t_h}, a contradiction (Fig.2).

With $T(z)$ in hand, we can perform two types of operations of great use later on.

1. **Ranking** z: to compute the rank of z in A, i.e., the number of real entries $a_{ij} \leq z$, we set $\delta_i = 0$ if $l_i = 0$ and $\delta_i = 1$ otherwise. The rank of z is given by the expression $\sum_{1 \leq i \leq n} \delta_i(t_i - l_i + 1)$, which can be evaluated in $O(n)$ time.
2. **Computing Intervals**: Let $z_1 < z_2$ be two arbitrary reals and let N be the set of entries in A comprised between z_1 and z_2. Once the traces of z_1 and z_2 are available, it is trivial to compute the set N (in its full expansion) in $O(n + |N|)$ time.

We now discard the dummy columns of A (indices 0 and $n+1$), which are no longer useful. Let P be the $(\lceil \frac{n}{3} \rceil \times \lceil \frac{n}{3} \rceil)$ matrix obtained by picking every third row and every third column in A. We have $p_{i,j} = a_{3i-2,3j-2}$, with $1 \leq i,j \leq \lceil \frac{n}{3} \rceil$. Note that it is possible to compute P (in standard form) in $O(n)$ time. Let x be an arbitrary real number. We define $r_a(x)$ and $r_p(x)$ as the ranks of x among the real entries of A and P, respectively. Let $\lambda = r_p(x)$ and L be the set of λ real entries in P less than or equal to x. An element $p_{i,j}$ of L is said to be *covered* if $p_{i,j-1}$, $p_{i-1,j}$ and $p_{i-1,j-1}$ are real entries (hence, lie in L). We next show that most elements of L are covered. For convenience, we call undefined entries such as $p_{0,0}$ blank. Each row of P has at most one real entry $p_{i,j}$ with $p_{i,j-1}$ blank. Also, it is impossible to have $p_{i,j}$ and $p_{k,j}$ both real and uncovered ($i \neq k$), with both $p_{i,j-1}$ and $p_{k,j-1}$ real. Therefore there are at most $2\lceil n/3 \rceil - 1$ uncovered real entries in P, hence at least $\lambda - 2\lceil n/3 \rceil + 1$ covered entries in L. We now exploit the one-to-one correspondence between each covered element $a_{i,j}$ in L and the set $\Lambda_{i,j} = \{a_{\alpha,\beta} \mid i-2 \leq \alpha \leq i \text{ and } j-2 \leq \beta \leq j\}$. Since $a_{i,j}$ is covered and A is monotone, $\Lambda_{i,j}$ consists exclusively of real entries. This implies that A contains at least $9\big(\lambda - 2\lceil \frac{n}{3} \rceil + 1\big)$ real entries less than or equal to x, i.e.,

$$r_a(x) \geq 9\big(r_p(x) - 2\lceil \frac{n}{3} \rceil + 1\big). \tag{1}$$

We say that a real entry $a_{i,j}$ is *protected* if each element of $\Lambda_{i,j}$ is real. Each row of A has at most two real entries $a_{i,j}$ such that $a_{i,j-1}$ or $a_{i,j-2}$ is blank. Also, it is impossible to have $a_{i,j}, a_{k,j}, a_{l,j}$ real and unprotected ($i < k < l$), with $a_{i,j-1}$, $a_{i,j-2}$, $a_{k,j-1}$, $a_{k,j-2}$, $a_{l,j-1}$, and $a_{l,j-2}$ all real. Therefore there are at most $2n + 2(n-2)$ unprotected real entries in A. Let M be the set of real entries in A less than or equal to x. Obviously, for each element $a_{i,j}$ of M which is protected, there exists one element of L in $\Lambda_{i,j}$. We can then partition M into its subset of non-protected elements and a

subset mapping to L. Since at most nine elements of M can map to the same element in L, we have $|M| \leq 9|L| + 4(n-1)$. Combining with (1) and simplifying a little, we obtain

$$9r_p(x) - 6n - 9 \leq r_a(x) \leq 9r_p(x) + 4n - 4. \tag{2}$$

Let p be the number of real entries in P, and let $k_1 = \lfloor \frac{k-4n+4}{9} \rfloor$ and $k_2 = \lceil k/9 + 2n/3 \rceil + 1$. To compute z, the k^{th} largest element in A, we will compute recursively z_1 and z_2, the k_1^{th} and k_2^{th} largest elements of P, respectively. This is assuming that $1 \leq k_1, k_2 \leq p$; otherwise if $k_1 < 1$, set z_1 to be the smallest real entry in A. Similarly if $k_2 > p$, set z_2 to be the largest real entry in A. From (2), we derive that z lies in the set $N = \{a_{i,j} \mid z_1 \leq a_{i,j} \leq z_2\}$ and that in all cases $|N| = O(n)$. The idea is then to determine the set N, compute the rank of z_1 in A, and then use a linear-time selection algorithm to derive the $(k+1-r_a(z_1))^{\text{th}}$ largest element in N, which is also the k^{th} largest element in A. As shown earlier, each of these operations can be executed in $O(n+|N|) = O(n)$ time. As a result, the overall complexity of the algorithm follows a recurrence of the form $T(1) = O(1)$ and $T(n) = 2T(\lceil n/3 \rceil) + O(n)$, from which we derive $T(n) = O(n)$.

For the case $c > 1$, we simply line up the diagonals of each matrix with the diagonal of a $cn \times cn$ matrix (Fig.3). ∎

2.2. Selecting a Bridge-Length

Let $P = \{v_0, \ldots, v_{m-1}\}$ and $Q = \{w_m, \ldots, w_{n-1}\}$ be two disjoint convex polygons. We define a *bridge* between P and Q to be any segment $v_i w_j$ which does not intersect the interior of P or Q. McKenna and Toussaint [MT] have described a linear time algorithm for computing the shortest distance between any v_i and w_j (see also [CW,T]). Although it is easily seen that the nearest pair of vertices does not necessarily form a bridge, it is simple to determine the shortest bridge in $O(n)$ time using the ideas behind McKenna and Toussaint's algorithm. How hard is it to compute the k^{th} longest bridge between P and Q? Note that there are in general on the order of n^2 bridges, so applying a linear-time selection algorithm to the explicit set of all bridges requires at least quadratic time. We will show that computing the k^{th} largest bridge is no more difficult than computing the smallest one.

The basic idea is to encode all bridge lengths into a constant number of monotone matrices, and then apply Lemma 2. Let A be the $m \times (n-m)$ matrix, where $a_{i,j}$ is the length of $v_i w_j$ if this segment is a bridge, or a blank otherwise. Unfortunately, A is in general not monotone, so some refinement is in order. We say that A is *decomposed* into matrices $A_1, \ldots, A_c$ if there is a one-to-one correspondence between the real entries of A and those of $A_1, \ldots, A_c$.

Lemma 3. The matrix A can be decomposed into four or fewer monotone matrices, each of which can be computed in standard form in $O(n)$ time.

Proof: Using for example the merge part of Preparata and Hong's convex hull algorithm [PH], compute the two outer tangents common to P and Q in $O(n)$ time. We refer here to each line tangent to P and Q with both polygons on the same side. For convenience, we will assume that no three points are collinear. The points of contact partition P (resp. Q) into two chains; one of these chains has each of its points visible from at least one point of its counterpart in Q. We label these two visible chains $P^* = \{p_1, \ldots, p_{n_1}\}$ and $Q^* = \{q_1, \ldots, q_{n_2}\}$. The former turns counterclockwise around P^* and the latter clockwise around Q^* (Fig.4). The real entries of A are identical to the real entries of the $(n_1 \times n_2)$ matrix $B = (b_{ij})$: $b_{ij} = |p_i q_j|$ if $p_i q_j$ is a bridge else b_{ij} is blank. Consider the effect of a tangent line rolling clockwise around Q^* starting at $q_1 p_1$. This line will pass successively through $p_1, p_2, \ldots$, until it becomes tangent to P^*; let AB be the segment thus obtained with A (resp. B) a vertex of Q^* (resp. P^*). At this point switch the polygon on which the rolling takes place from Q^* to P^* and pursue a counterclockwise rotation around P^* until the line passes through $p_{n_1} q_{n_2}$. It is easy to simulate these two rotations in $O(n)$ time. This puts each p_i in correspondence with one or several consecutive vertices of Q^* (one at first and then possibly several past AB). Let $a(i)$ be the smallest index such that $q_{a(i)}$ is in correspondence with p_i. The function $a(i)$ is well defined for each value of i. It indicates one of the limit bridges in the interval of bridges emanating from p_i. A symmetric rolling process starting at $p_{n_1} q_{n_2}$ —counterclockwise around Q^* and then clockwise around P^* past CD— leads to the function $b(i)$. We omit the proof that, for each i, we have $a(i) \leq b(i)$, and that both functions are nondecreasing (proof partially based on the uniqueness of AB and CD). Another way of characterizing these two functions is to show that the set of bridges adjacent to p_i is precisely $\{p_i q_{a(i)}, \ldots, p_i q_{b(i)}\}$. From this result, we derive the useful information that within each row of B the set of real entries forms a consecutive interval. Of course a similar reasoning leads to the same fact concerning each column of B.

From the fact that Q is convex and p_i is visible from $q_{a(i)}, \ldots, q_{b(i)}$ outside Q, it easily follows that the sequence of distances $\{|p_i q_{a(i)}|, \ldots, |p_i q_{b(i)}|\}$ is unimodal. It can be rewritten as a nonempty decreasing sequence $\{|p_i q_{a(i)}|, \ldots, |p_i q_{c(i)}|\}$, followed by an increasing sequence (possibly empty) $\{|p_i q_{c(i)+1}|, \ldots, |p_i q_{b(i)}|\}$. The first sequence is of *horizontal type 0* and the latter of *horizontal type 1*. Next we show that, like $a(i)$ and $b(i)$, the function $c(i)$ is nondecreasing. Assume that it is not; then for some i, we have $c(i) > c(i+1)$. The sequence of inequalities

$$a(i) \leq a(i+1) \leq c(i+1) < c(i) \leq b(i) \leq b(i+1)$$

implies that p_i and p_{i+1} are both visible from $q_{c(i)}$ and $q_{c(i+1)}$. This implies the convexity of the quadrilateral $p_i p_{i+1} q_{c(i)} q_{c(i+1)}$, which in turns contradicts the fact that $|p_i q_{c(i)}| \leq |p_i q_{c(i+1)}|$ and

$|p_{i+1}q_{c(i+1)}| \leq |p_{i+1}q_{c(i)}|$. The fact that $c(i)$ is nondecreasing and that each row is unimodal suggests a straightforward $O(n)$ method for computing $c(1), \ldots, c(n_1)$.

The same reasoning applied to the columns of B leads to the definition of a function $c'(i)$, which puts in correspondence each q_i with its shortest bridge. For the same reasons, $c'(i)$ is nondecreasing. An entry of horizontal type i and vertical type j is said to be of type ij. The matrix B is thus partitioned into regions of type 00,01,10, and 11 (Fig.5). To compute the boundaries between these regions is easily done in $O(n)$ time. Also, each collection of regions of a given type forms a matrix whose real entries appear consecutively in both rows and columns. Why is that so? Obviously, the horizontal types of two consecutive entries on the same row cannot be 1 and 0 in this order. We claim that the two vertical types cannot give the sequence 0,1. Indeed, doing so would contradict the fact that $c'(i)$ is nondecreasing. This shows that although the regions of a given type may be made of disconnected pieces, the whole collection consists of consecutive (possibly empty) intervals of real entries on each row. A similar reasoning leads to the same conclusion with respect to the columns. As a result, we can decompose B into four matrices (or fewer) of respective type 00, 01, 10, 11. Each of these matrices is monotone (after appropriate rotations for all but those of type 11, because our definition of matrix monotonicity requires increasing rows and columns). ∎

Combining Lemmas 2 and 3, we conclude:

Theorem 1. Given two disjoint convex polygons with n vertices, it is possible to compute the k^{th} longest bridge between them in $O(n)$ time and space.

2.3. Selecting a distance between n points

Let S be a set of n points in d-dimensional Euclidean space, E^d. How difficult is it to determine the pair of points whose distance to each other is the k^{th} largest? Yao has shown that the two points furthest apart can be found in less than quadratic time [Y]. We will show that selecting the k^{th} largest interdistance can be done with little added cost.

Theorem 2. Let S be a set of n points in E^d. The k^{th} largest interdistance between points of S can be found in $O(n^{2-1/(2^{d+1}-1)} \log n)$ time.

Proof: Let $p_1, \ldots, p_n$ be the points of S and let $S_1, \ldots, S_\alpha$ be a partition of S into α groups of at most $p = \lceil n/\alpha \rceil$ points each. Let $h_{i,j}$ be the hyperplane bisecting p_i and p_j. We begin by applying Dobkin and Lipton's method for point location [DL] separately to each set of hyperplanes $\{h_{k,l} \mid p_k, p_l \in S_i\}$. For each i between 1 and α this produces a search tree T_i, whose leaves we label $1, 2, \ldots$ in arbitrary order. Note that the regions corresponding to the leaves of T_i are a refinement of the Voronoi diagram of S_i. Next we use T_i to locate each point of S among the bisectors of

S_i. Let $L_{i,j}$ be the set of points of S whose path in T_i ends at the leaf labelled j. Because of the structure of T_i each point of $L_{i,j}$ lies in on the same side of the bisectors of S_i. This implies that, for each j, there exists an ordering of S_i such that the k^{th} point in the ordering is the k^{th} furthest point of S_i from *any* point of $L_{i,j}$. This ordering, denoted $S_i^{(j)}$, can be computed by sorting the distances between the points of S_i and an arbitrary point of $L_{i,j}$. This scheme allows us to partition the set of interdistances between the points of S into sorted subsets. Each subset belongs to a class characterized by an ordering of the form $S_i^{(j)}$. Random access into the subset is done via $S_i^{(j)}$, which is stored in full in an array. We now satisfy the conditions of Lemma 1 for computing the k^{th} largest interdistance in S. (We leave it as an exercise to the reader to show how to deal with the fact that distances appear twice.) What is the complexity of this algorithm?

Applying Dobkin and Lipton's method to m hyperplanes in d-dimensional space produces a search tree of $O(m^{2^d-1})$ leaves at the cost of $O(m^{2^d-1}\log m)$ preprocessing. It follows that all the trees T_i can be constructed in a total of $O(\alpha p^{2^{d+1}-2}\log p)$ time. Since the trees have height $O(\log p)$, computing the sets $L_{i,j}$ can be done in a straightforward fashion in $O(\alpha n\log n + \alpha p^{2^{d+1}-2})$ time. Computing the orderings $S_i^{(j)}$ requires $O(\alpha p^{2^{d+1}-1}\ \log p)$ operations and application of Lemma 1 $O(\alpha n\log n)$. This gives a total running time of

$$O(\alpha p^{2^{d+1}-1}\log n + \alpha n\log n).$$

Setting $p = n^{1/(2^{d+1}-1)}$ completes the proof. ∎

It is easy to improve the upper bound for $d = 2$ and $d = 3$, and in general, in any situation where point location can be done more efficiently than in [DL].

2.4. Selecting a triangle among n points

Given n points in the Euclidean plane, it was shown independently in [CGL] and [EOS] that the smallest-area triangle formed by any three points —or any of them if there are several— can be found in $O(n^2)$ time. We will show that selecting the k^{th} largest triangle is barely more difficult.

Theorem 3. Let S be a set of n points in the Euclidean plane. The k^{th} largest-area triangle formed by the points of S can be computed in time $O(n^2\log n\log(2\lceil k/n^2\rceil))$.

Proof: Let A be the arrangement formed by n lines in the plane. Let (Ox, Oy) be an orthogonal system of coordinates such that no line is parallel to the y-direction. We turn our attention to the sequence of lists obtained by sweeping the plane with a vertical line. For each vertex v of the arrangement, consider the points of the lines of A that have the same x-coordinates as v; let $L(v)$ denote the list of these points sorted in nondecreasing order of y-coordinates. (It suffices that $L(v)$ contains the indices of the corresponding lines of A.) Note that v appears twice in $L(v)$. Let

$v_1, \ldots, v_p$ be the vertices of A in nondecreasing order of x-coordinates ($p = \binom{n}{2}$); ties are broken arbitrarily. Observe that, for every $i > 1$, the list $L(v_i)$ differs from $L(v_{i-1})$ in only two places. A recent result of Cole [Co1] shows that in these conditions it is possible to preprocess the set of lists in $O(n^2 \log n)$ time so that for any pair i, j the j^{th} item in $L(v_i)$ can be found in $O(\log n)$ time. To see the connection between this discussion and our selection problem, transform every point $p : (a, b)$ of S into the line $T_p : y = ax + b$. Let v be the vertex formed by the intersection of two lines T_p and T_q, where p and q are two points of S. The list $L(v)$ can be partitioned into two subsets, $L^+(v)$ and $L^-(v)$, which contain all the points of $L(v)$ respectively above and below v. Note that every element of $L(v)$ can be associated with a triangle formed by three points of S: p, q, and the point r such that T_r contributes the element of $L(v)$ in question. It is straightforward to prove that $L^-(v)$ and $L^+(v)$ correspond to triangles of either decreasing or increasing area (we assume for convenience that the points of S are in general position). This allows us to apply Lemma 1 to find the k^{th} largest triangle. (Note that the multiple occurrence of any triangle is easy to deal with.) Random access into the sorted subsets of Lemma 1 can be done in $O(\log n)$ time, provided that the cardinality of, say, $L^-(v_i)$ has been computed for each i. This can be done in $O(n^2 \log n)$ time by applying a standard sweep-line algorithm. It can also be done in $O(n^2)$ time but we do not really need a faster method at this point. Lemma 1 shows that the complexity of the algorithm is

$$O(n^2 \log n + \min(k, 2\binom{n}{2}) \log n \log(k/\min(k, 2\binom{n}{2}))) = O(n^2 \log n \log(2\lceil k/n^2 \rceil)).$$

∎

2.5. Discussion

We will limit ourselves to the three problems discussed above because they illustrate some of the different techniques one can use to solve selection problems in implicit geometric sets. In the first case, we used a reduction to selection in a monotone matrix: this was the best we could hope for since it can be solved optimally. In the second and third cases we reduced the problems to selection in a collection of sorted sets represented implicitly. For the selection of interdistances, we tried to break down the problem into highly structured subproblems. In the case of the triangle areas, we used a simulation of random-access to draw the benefits of Lemma 1.

As one can easily imagine, many other selection problems are solvable by these techniques – see [Ch3,Sa] for additional examples. We close this section here as our objective was less solving selection problems for their own sake than presenting general lines of attack for problems of that nature. We summarize the two-pronged approach used:

1. Decompose the underlying (implicit) set into strongly structured components, and map the original problem into a collection of selection problems on sorted sets or monotone matrices.

2. Implement the random-access primitive required in the solution of the selection problems.

However large the class of problems amenable to this treatment may be, some selection problems seem inherently more difficult. For example, we submit the following open problem: given n points in the Euclidean plane, how hard is it to compute the k^{th} largest-area convex polygon formed by any subset of the points? Note that the *smallest* polygon can be computed in $O(n^2)$ time (disregarding 2-gons), while the *largest* (i.e., the convex hull) can be found in $O(n \log n)$.

3. Triangular Range Search

Let $S = \{p_1, \ldots, p_n\}$ be a set of n points in E^2. The *triangular range search* problem, or triangle problem for short, is to preprocess S so that for any given query triangle T the points of S that lie in T can be computed efficiently. To date, the most efficient solutions to this problem require $O(n)$ space and $O(k+n^{.667})$ query time [HW] (using probabilistic preprocessing), or $O(n)$ space and $O(k+n^{.695})$ query time [EW], or $O(n^{2+\epsilon}/\log n)$ space and $O(k + \log n \log 1/\epsilon)$ query time [CY], where k is the number of points reported, which is also the size of the output. Obviously the problem is at least as difficult as the point-on-a-line problem: given n points, determine if a query line passes through any of the points. This problem can be restated in dual space by transforming points into lines and lines into points [Ch1]. Then it becomes the problem of deciding whether a query point lies on any of n given lines. The lines form an arrangement which subdivides the plane into $O(n^2)$ convex regions, and can thus be searched efficiently with appropriate preprocessing [PS]. Unfortunately the search domain is quadratic in size, which is excessive in many applications. Whether it can be represented implicitly — after all it is defined only by n lines — and still accommodate fast searching is open. What we can show, however, is that this degenerate version of the triangle problem seems to be the most difficult subproblem. Indeed, if the angles of the query triangle are bounded below by a constant, then the problem falls in the class *PLOG*, that is, can be solved in $O(n \times \text{polylog}(n))$ space and $O(k+\text{polylog}(n))$ query time. Our next result is one step towards a practical solution to the triangle problem. It involves a geometric construction of independent interest. In the following, the term *bounded-angle* triangle problem refers to the restricted version of the triangular range search problem where no angle of a query triangle is allowed below a constant α.

Theorem 4. The bounded-angle triangle problem on n points can be solved in $O(n \log^2 n)$ space and $O(k + \log^2 n)$ query time, where k is the size of the output.

Proof: The main idea is to decompose the query triangle into $O(\log n)$ grounded triangles. A triangle is said to be *grounded* with respect to a line L if it has a right angle and one of the edges adjacent to it is collinear with L. Let S be a set of n points in the plane and let L be an arbitrary line. From [CG] we know that there exists a data structure of linear size, $T_L(S)$, such that for any grounded

triangle q all k points of $S \bigcap q$ can be reported in $O(k+\log n)$ time. Suppose now that q is still a right triangle but that one of its non-hypothenuse edges is parallel to L (and not necessarily collinear with it). We will show that the problem can be solved just the same, although at an extra multiplicative cost of $\log n$ in time and space.

Let $p_1, \ldots, p_n$ be the points of S sorted along a line normal to L. We define a binary tree $\mathcal{T}$ with various auxiliary structures attached to its nodes. The construction is recursive.

1. If $n = 1, 2$ then $\mathcal{T}$ is a simple node with a pointer to a list of the points in S.
2. If $n > 2$ then let $m = \lfloor n/2 \rfloor$ and let D be a line parallel to L which intersects the segment $p_m p_{m+1}$. The root v of $\mathcal{T}$ has a pointer to $T_D(S)$; its left and right subtrees are defined recursively with respect to $\{p_1, \ldots, p_m\}$ and $\{p_{m+1}, \ldots, p_n\}$.

Given any query triangle q (satisfying the conditions indicated above), the points of $S \bigcap q$ can be found by the following procedure. If S consists of one or two points, check each of them for inclusion in q. Otherwise, check if q intersects the line D associated with the root of $\mathcal{T}$. If not, then recur in the appropriate subtree. Else, the intersection of q with one of the halfplanes delimited by D is a triangle grounded with respect to D. The points of S falling in it can be retrieved by using $T_D(S)$. Once this has been done, the algorithm can recur with respect to one of the root's children. It is elementary to verify that if k points must be reported then the algorithm will take $O(k+\log^2 n)$ time. The storage required is $O(n \log n)$.

Next we can turn to the general bounded-angle triangle problem. We assume that each angle of the query triangle q is at least as large as α. Let D_0 be an arbitrary line in the plane; we construct a data structure of the previous type with respect to each direction in the collection

$$\Delta = \{D_0, \ldots, D_p\},$$

where D_i is a line forming an angle $i\alpha$ with D_0 and $p = \lceil \pi/\alpha \rceil - 1$. Angles between lines are measured in $[0, \pi)$. The theorem will follow once we show how to partition q into a constant number of right triangles with one non-hypothenuse edge parallel to some line in Δ. Let ABC be an arbitrary triangle, each of whose angles is greater than or equal to α. We can always assume that each angle is at most $\pi/2$. If this is not the case, there is a unique vertex, say A, displaying an angle larger than $\pi/2$ (Fig.6-A). Let D be the point of BC which bisects the angle $\angle(AB, AC)$ and assume without loss of generality that $\angle(DC, DA)$ is larger than or equal to $\pi/2$. If it is equal we are done, since $\angle(AC, AD)$ and $\angle(AD, AB)$ are between $\pi/4$ and $\pi/2$. Note that in that case we must have the preassigned condition $\alpha < \pi/4$. If $\angle(DC, DA)$ is not equal to $\pi/2$, then we draw the altitude DE from D to AC. It is now easy to prove that each angle among the triangles ADB, AED, and ECD is between α and $\pi/2$. Obviously $\angle(DA, DB)$ is larger than $\angle(AD, AC)$ (think of the line parallel to BC passing through A), therefore $\angle(DA, DB) > \frac{1}{2}\angle(AB, AC) > \pi/4 > \alpha$. We

also have $\angle(AD, AE) > \pi/4 > \alpha$. Now, since neither $\angle(BC, BA)$ nor $\angle(CA, CB)$ are less than α, $\angle(AB, AC)$ is at most equal to $\pi - 2\alpha$, therefore $\angle(AD, AE)$ does not exceed $\pi/2 - \alpha$. This implies that $\angle(DE, DA)$ is at least as large as α. Finally, since $\angle(DC, DA)$ is at least $\pi/2$, $\angle(CE, CD)$ cannot exceed $\angle(BD, BA)$, hence it is no larger than $\pi/4$. This shows that $\angle(DC, DE)$ is at least $\pi/4 \geq \alpha$. Checking the other angles is trivial, so let's now assume that ABC has all of its angles between α and $\pi/2$.

There must exist a segment AA' parallel to some D_a in Δ with $A' \in BC$ (Fig.6–B). Let I_B (resp. I_C) be the intersection of the line passing through AA' with the normal to AA' passing through B (resp. C). Since $\angle(AB, AC) \leq \frac{\pi}{2}$, the triangle ABC lies entirely on one side of the normal to AA' passing through A, therefore we have $I_B \in AI_C$ or $I_C \in AI_B$. Without loss of generality, assume that $I_B \in AI_C$. Let B' be the intersection of AC with the line passing through BI_B. There exists a segment CC' parallel to some $D_c \in \Delta$ with $C' \in BB'$. Since AI_BB' forms a right triangle, $\angle(B'B, B'C)$ is obtuse, therefore the normal to CC' passing through B' intersects CC' at some point, denoted $J_{B'}$. Let D be the intersection of BC with the line passing through $B'J_{B'}$. Since $CJ_{B'}D$ forms a right triangle, $\angle(DB', DB)$ is obtuse, so the normal to BB' passing through D intersects BB' at some point $D' \in BB'$. This completes the partitioning of ABC into six right triangles $\{AI_BB, AI_BB', BD'D, B'D'D, DJ_{B'}C, B'J_{B'}C\}$, each with one (non-hypothenuse) side parallel to a line in Δ. The initial splits make 18 a trivial upper bound on the number of pieces in the partition. ∎

4. Computing Algebraic Predicates

Hopcroft has posed the following problem: Given n lines and n points in the plane, check whether any line passes through any the points. One could also ask slightly more esoteric questions such as: given n points in E^3, are 5 of them cospherical or do 17 of them have their mass center coincide with one of the points? In lower dimensions (as in Hopcroft's problem) there is always hope for efficient, ad hoc methods. In higher dimensions, however, it is clear that only fairly general lines of attack can bear fruit. We will propose a general approach to this problem, reminiscent of the "Four Russians" algorithm for boolean matrix multiplication [AHU]. The approach is inspired by a technique of Yao [Y]. The general question which we ask is: *Given n geometric objects in arbitrary dimensions, do k of them interact in a prespecified manner?*

This general class of questions can be easily formalized. Let $S = \{V_1, \ldots, V_k\}$ be a collection of k sets of the form $V_i = \{v_1^{(i)}, \ldots, v_{p_i}^{(i)}\}$. Each $v_j^{(i)}$ is a vector $\in Q^{c_i}$ of the form $v_j^{(i)} = (x_{j,1}^{(i)}, \ldots, x_{j,c_i}^{(i)})$. The set S is the *input* and the quantity $n = \sum_{1 \leq i \leq k} p_i c_i$ is the *input size*. Let $Q_d(y_{1,1}, \ldots, y_{1,c_1}, y_{2,1}, \cdots, y_{2,c_2}, \cdots, y_{k,1}, \cdots, y_{k,c_k}, z_1, \ldots, z_h)$ be a rational λ–variate polynomial of degree d; $\lambda = h + \sum_{1 \leq i \leq k} c_i$. If $w_i = (y_{i,1}, \ldots, y_{i,c_i})$, we use the abbreviation $Q_d(w_1, \ldots, w_k; z_1, \ldots, z_h)$. In the following, the quantities $d, k, c_1, \ldots, c_k, h$ are considered to be constants.

Problem K: Does there exist a k-tuple $(j_1, \ldots, j_k) \in \Pi_{1 \leq i \leq k}\{1, \ldots, p_i\}$ and an h-tuple $(z_1, \ldots, z_h) \in \Re^h$ such that $Q_d(v_{j_1}^{(1)}, \ldots, v_{j_k}^{(k)}; z_1, \ldots, z_h) = 0$?

There is a trivial $O(n^k)$ solution involving the testing of all possible k-tuples $(j_1, \ldots, j_k)$. Each test involves checking if a polynomial of constant degree, with rational coefficients and a constant number of variables, has real zeros. As is well-known [Ta] this can be decided effectively. The following result asserts that it is always possible to do (a little) better. Before proceeding, we now observe that the previous problems can be easily rephrased as instances of Problem K (one will also see the role of the z_i's, which might not be obvious at first sight).

1. Let $\{a_i x + b_i y + c_i = 0 \mid 1 \leq i \leq n\}$ and $\{(x_i, y_i) \mid 1 \leq i \leq n\}$ be respectively n lines and n points in the plane. Whether a point passes through a line is equivalent to Problem K for: $k = 2$, $V_1 = \{(a_1, b_1, c_1), \ldots, (a_n, b_n, c_n)\}$, $V_2 = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, $h = 0$, and $Q_2(y_{1,1}, y_{1,2}, y_{1,3}, y_{2,1}, y_{2,2}) = y_{1,1}y_{2,1} + y_{1,2}y_{2,2} + y_{1,3}$.
2. Let $\{p_i = (x_i, y_i, z_i) \mid 1 \leq i \leq n\}$ be n points in E^3. Whether 5 points are cospherical can be expressed by the sentence: there exist i, j, k, l, m such that $P(p_i, i, p_j, j, p_k, k, p_l, l, p_m, m)$ is true. $P(\ldots)$ is the first-order sentence: "$i \neq j$, $i \neq k$, $i \neq l$, $i \neq m$, $j \neq k$, $j \neq l$, $j \neq m$, $k \neq l$, $k \neq m$, $l \neq m$ and there exist reals a, b, c, d such that $(x_i - a)^2 + (y_i - b)^2 + (z_i - c)^2 = d^2$, $(x_j - a)^2 + (y_j - b)^2 + (z_j - c)^2 = d^2$, etc." Each statement of the form $i \neq j$ can be written as "there exists x such that $x(i - j) - 1 = 0$." To have both $A = 0$ and $B = 0$ one will write $A^2 + B^2 = 0$. This leads to an equivalent form to the previous statement: "there exist reals $a, b, c, d, \chi_1, \ldots, \chi_{10}$ such that $Q_4(p_i, i, p_j, j, p_k, k, p_l, l, p_m, m; a, b, c, d, \chi_1, \ldots, \chi_{10}) = 0$. Therefore we will have $V_i = \{(x_1, y_1, z_1, i), \ldots, (x_n, y_n, z_n, i)\}$, for $i = 1, \ldots, 5$, and $(z_1, \ldots, z_h) = (a, b, c, d, \chi_1, \ldots, \chi_{10})$.

Theorem 5. Problem K can be solved in $O(n^{k-\epsilon})$ time, for some real $\epsilon > 0$.

Proof: Consider the m $(h + c_k)$-variate polynomials of the form $Q_d(v_{j_1}^{(1)}, \ldots, v_{j_{k-1}}^{(k-1)}, x_1, \ldots, x_{c_k}; z_1, \ldots, z_h)$, where $x_1, \ldots, x_{c_k}$, $z_1, \ldots, z_h$ are the variables and $(j_1, \ldots, j_{k-1}) \in \Pi_{1 \leq i \leq k-1}\{1, \ldots, p_i\}$; note that $m \leq n^{k-1}$. Divide up this set of polynomials into subsets of size roughly α, denoted $W_1, \ldots, W_t$ $(t = O(n^{k-1}/\alpha))$. In [Ch2] a method is described for preprocessing a set of polynomials so that determining whether a query vector is a subvector of a zero of one of them can be done in logarithmic time. More precisely, let $F = \{P_1, \ldots, P_\alpha\}$ be a family of α fixed-degree r-variate polynomials with rational coefficients (and r a constant), and let $S = \{x \in E^r \mid \Pi_{1 \leq i \leq \alpha} P_i(x) \neq 0\}$. In $O(\alpha^{O(1)})$ time and space, it is possible to compute a set of algebraic points, one in each connected region of S, as well as set up a data structure for computing the predicate $[\exists i\ (1 \leq i \leq \alpha) \mid P_i(q) = 0]$, for any $q \in E^r$. In $O(\log \alpha)$ time, the algorithm will return an index i such that $P_i(q) = 0$ if such an index is to be found, otherwise it will return the algebraic point associated with the unique region of S that contains q. Furthermore, we can adapt the data structure so that for any $(q_1, \ldots, q_l)$, we can determine in logarithmic time whether there exist $(q_{l+1}, \ldots, q_r)$ such that

for some i, we have $P_i(q_1, \ldots, q_r) = 0$. Applying this method to each set $W_1, \ldots, W_t$ requires $O(n^{k-1}\alpha^{O(1)})$ preprocessing time. On the other hand, this will allow us to test the p_k vectors of V_k in $O(p_k t \log \alpha) = O((n^k \log \alpha)/\alpha)$ overall query time. Setting α so as to balance preprocessing and query times leads to a time complexity of $O(n^{k-\epsilon})$, for some $\epsilon > 0$. ∎

5. Conclusions

The aim of this work has been to present various techniques for (partially) circumventing the difficulty of searching implicit sets without expanding the search domain in full. Our approach has been almost exclusively algorithmic (except for the triangle problem). We feel that more geometric insights will be needed to produce further improvements in this area. The work in [W,EW,HW,YY] is illustrative of the kind of geometric and topological facts needed for future breakthroughs. We close with an open problem which seems to elude the techniques presented in this paper: organize two sets of numbers X and Y so that for any number q the pair $(x, y) \in X \times Y$ that minimizes $|x + y - q|$ can be found efficiently.

REFERENCES

[AHU] Aho, A.V., Hopcroft, J.E., Ullman, J.D. *The design and analysis of computer algorithms*, Reading, MA, Addison-Wesley, 1974.

[BS] Bentley, J.L., Shamos, M.I. *Divide-and-conquer in multidimensional space*, Proc. 8th Annu. ACM Symp. on Theory of Comput. (1976), pp. 220–230.

[Bl] Blum, M., Floyd, R.W., Pratt, V.R., Rivest, R.L., Tarjan, R.E. *Time bounds for selection*, JCSS, 7 (4), pp. 448–461, 1972.

[Ch1] Chazelle, B. *Filtering search: A new approach to query-answering*, SIAM J. on Comput., 15 (3), pp. 703–724, Aug. 1986.

[Ch2] Chazelle, B. *Fast searching in a real algebraic manifold with applications to geometric complexity*, Proc. CAAP'85, Berlin, West-Germany, LNCS, Springer-Verlag, pp. 145–156, March 1985.

[Ch3] Chazelle, B. *New techniques for computing order statistics in Euclidean space*, Proc. ACM Symp. on Comput. Geometry, June 1985, pp. 125–134.

[CG] Chazelle, B., Guibas, L.J. *Fractional cascading: II. Applications*, Algorithmica, 1 (2), pp. 163–191, 1986.

[CGL] Chazelle, B., Guibas, L.J., Lee, D.T. *The power of geometric duality*, BIT, 25 (1), 1985, pp. 76–90.

[CW] Chin, F., Wang, C.A. *Minimum vertex distance between separable convex polygons*, Info. Proc. Lett., 18 (1984), pp. 41–45.

[Co1] Cole, R. *Searching and storing similar lists*, J. of Algorithms, 7, pp. 202–220, 1986.

[Co2] Cole, R. *Slowing down sorting networks to obtain faster sorting algorithms*, Proc. of 25th Annu. IEEE Symp. on Foundations of Comput. Sci., Singer Island, FL, pp. 255–259, 1984.

[CY] Cole, R., Yap, C.K. *Geometric retrieval problems*, Information and Control, Vol. 63, Nos.1–2, pp. 39–57, Oct/Nov 1984.

[DL] Dobkin, D.P., Lipton, R.J. *Multidimensional searching problems*, SIAM J. on Comput. 5 (2), pp. 181–186, 1976.

[EOS] Edelsbrunner, H., O'Rourke, J., Seidel, R. *Constructing arrangements of lines and hyperplanes with applications*, SIAM J. on Computing, Vol. 15, No. 2, pp. 341–363, May 1986.

[EW] Edelsbrunner, H., Welzl, E. *Halfplanar range search in linear space and $O(n^{0.695})$ query time*, Rep. F111, Inst. Inform. Proc., Tech. Univ. Graz, Austria, 1983.

[FJ1] Frederickson, G.N., Johnson, D.B. *The complexity of selection and ranking in $X + Y$ and matrices with sorted columns*, JCSS, 24 (1982), pp. 197–208.

[FJ2] Frederickson, G.N., Johnson, D.B. *Finding kth paths and p-centers by generating and searching good data structures*, J. of Alg., 4 (1983), pp. 61–80.

[FJ3] Frederickson, G.N., Johnson, D.B. *Generalized selection and ranking: sorted matrices*, SIAM J. on Comput., 13 (1), pp. 14–30, 1984.

[GM] Galil, Z., Megiddo, N. *A fast selection algorithm and the problem of optimum distribution of effort*, J. of ACM, 26, pp. 58–64, 1979.

[HW] Haussler, D., Welzl, E. *Epsilon-nets and simplex range queries*, Proc. 2nd Annu. ACM Symp. on Computational Geometry, pp. 61–71, June 1986.

[JM] Johnson, D.B., Mizoguchi, T. *Selecting the k-th element in $X + Y$ and $X_1 + X_2 + \ldots + X_m$*, SIAM J. on Comput., 7 (2), pp. 147–153, 1978.

[MT] McKenna, M., Toussaint G.T. *Finding the minimum vertex distance between two disjoint convex polygons in linear time*, Tech. Rep. SOCS–83–6, McGill University, April 1983.

[M] Meggido, N. *Applying parallel computation algorithms in the design of serial algorithms*, J. of ACM, pp. 852–865, 1983.

[Me] Meggido, N., Tamir, A., Zemel, E., Chandrasekaran, R. *An $O(n \log^2 n)$ algorithm for the kth longest path in a tree with applications to location problems*, SIAM J. on Comput., 10 (2), pp. 328–337, May 1981.

[MA] Mirzaian, A., Arjomandi, E. *Selection in $X + Y$ and matrices with sorted rows and columns*, Info. Process. Lett., 20, pp. 13–17, 1985.

[PH] Preparata, F.P., Hong, S.J. *Convex hulls of finite sets of points in two and three dimensions*, Comm. ACM, 20, 2 (1977), pp. 87–93.

[PS] Preparata, F.P., Shamos, M.I. *Computational geometry*, New York: Springer-Verlag, 1985.

[Sa] Salowe, J.S. *Efficient geometric selection in the plane*, M.Sc. Thesis, Rutgers University, Aug. 1985.

[Sh] Shamos, M.I. *Geometry and statistics: problems at the interface*, Algorithms and complexity: new directions and recent results, ed. J.F. Traub, Academic Press, New York, pp. 251–280, 1976.

[Ta] Tarski, A. *A decision method for elementary algebra and geometry*, Univ. of Calif. Press, 1948, 2nd Ed., 1951.

[T] Toussaint, G.T. *An optimal algorithm for computing the minimum vertex distance between two crossing convex polygons*, Proc. 21st Allerton Conf. on Comm. Control and Comput. (1983), pp. 457–458.

[W] Willard, D.E. *Polygon retrieval*, SIAM J. Comp., 11 (1982), pp. 149–165.

[Y] Yao, A.C. *On constructing minimum spanning tree in k-dimensional space and related problems*, SIAM J. Comput. 11 (4), 1982, pp. 721–736.

[YY] Yao, A.C., Yao, F.F. *A general approach to d-dimensional geometric queries*, 17th Ann. ACM Symp. on Theory of Computing, Providence, RI, May 1985, pp. 163–168.

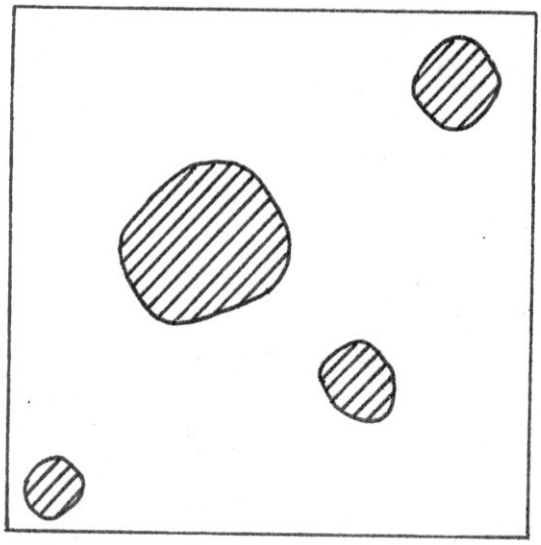


Figure 1

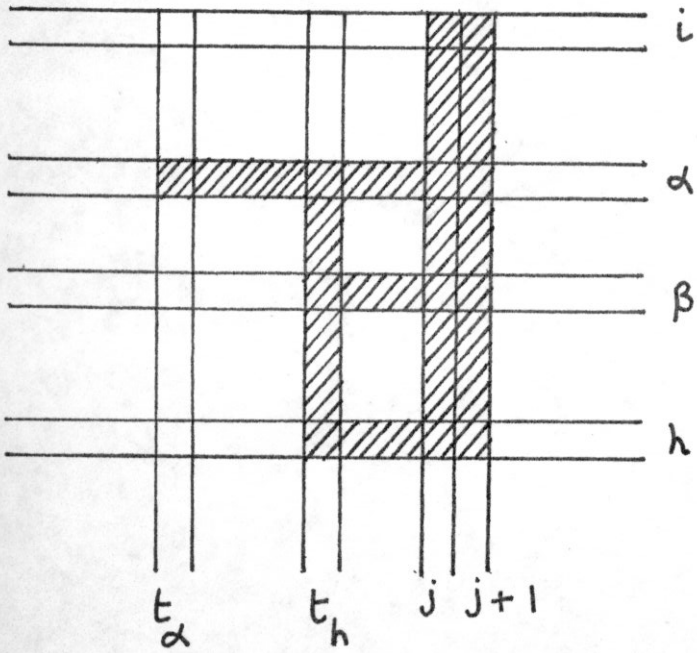


Figure 2

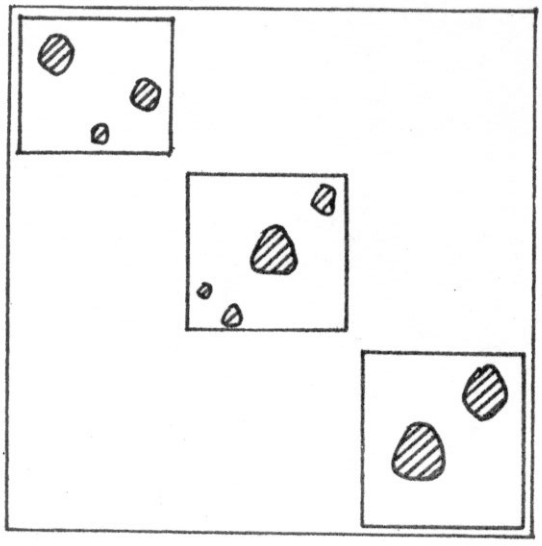


Figure 3

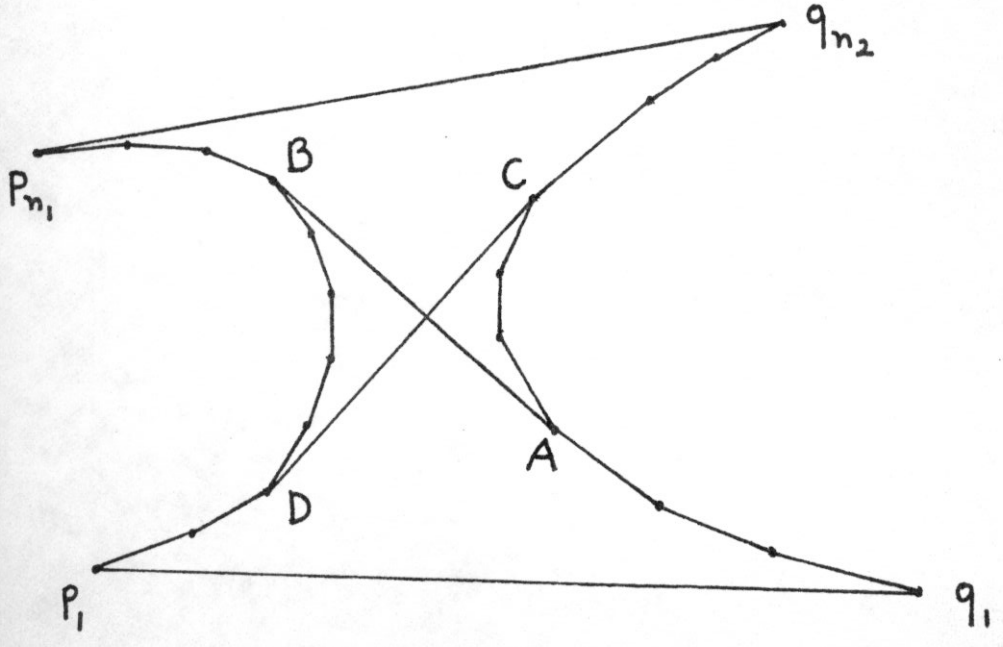


Figure 4

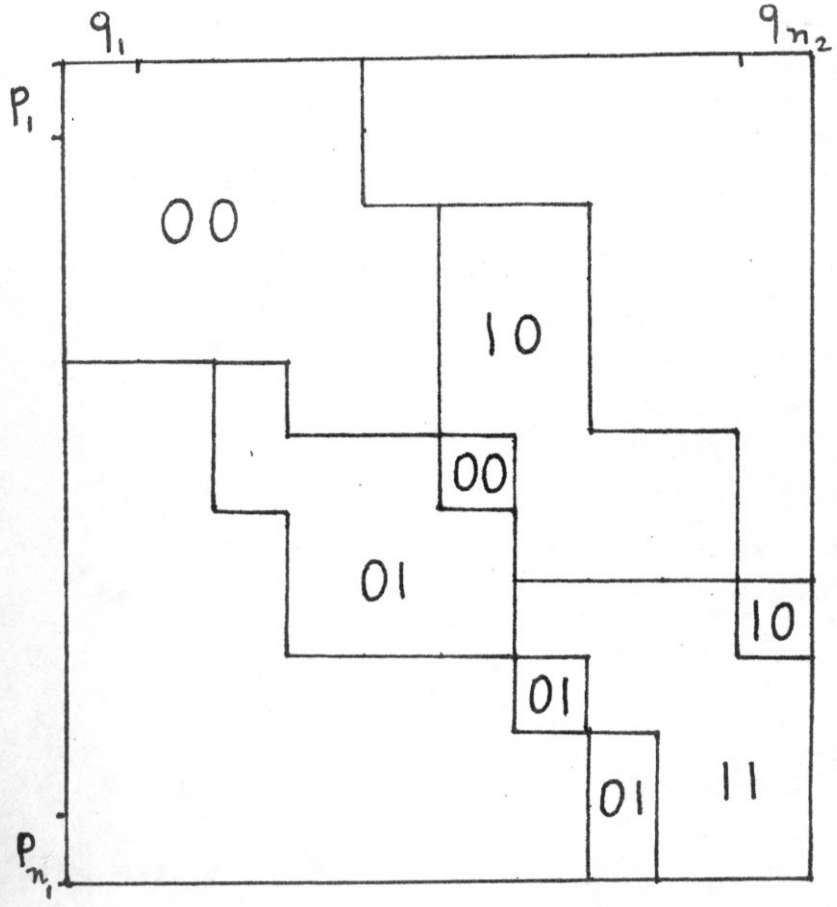
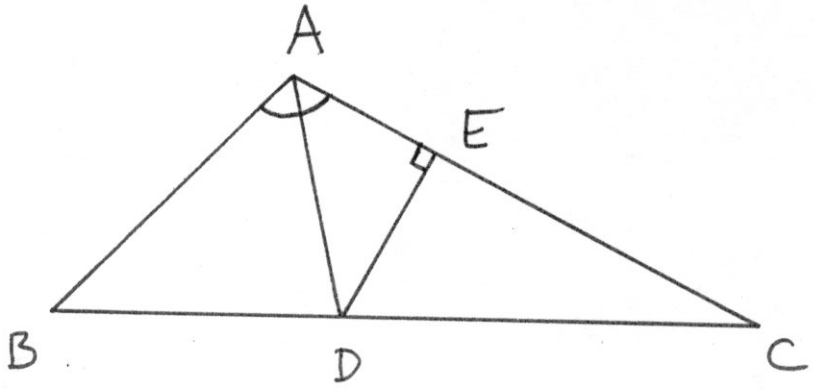
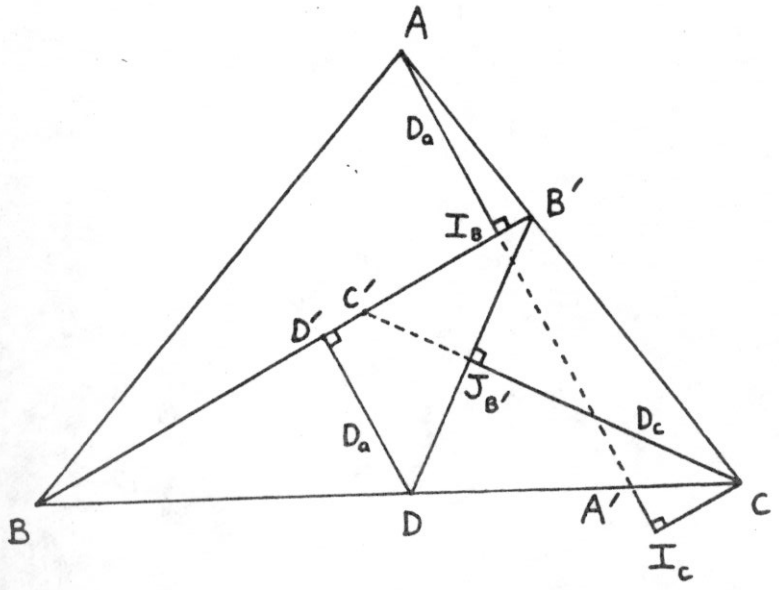


Figure 5



A)



B)

Figure 6