J. Symbolic Computation (1985) **1**, 47–56

Optimal Solutions for a Class of Point Retrieval Problems

B. CHAZELLE† AND H. EDELSBRUNNER‡

† *Department of Computer Science, Brown University, Providence, Rhode Island* 02912, *U.S.A. and*
‡*Institutes of Information Processing, Technical University Graz, Schiesstattg.* 4*a*,
A–8010 *Graz, Austria*

(*Received* 23 *July* 1984)

Let P be a set of n points in the Euclidean plane and let C be a convex figure. We study the problem of preprocessing P so that for any query point q, the points of P in $C+q$ can be retrieved efficiently. If constant time suffices for deciding the inclusion of a point in C, we then demonstrate the existence of an optimal solution: the algorithm requires $O(n)$ space and $O(k+\log n)$ time for a query with output size k. If C is a disk, the problem becomes the well-known *fixed-radius neighbour problem*, to which we thus provide the first known optimal solution.

1. Introduction

Let P be a set of n points in the Euclidean plane E^2, and let C be a convex figure. We study the complexity of the following problem: preprocess P so that for any query translate $C_q = C+q$ of C the points in $P \cap C_q$ can be retrieved efficiently. Intuitively, a query corresponds to an arbitrary displacement of C without rotation. We demonstrate the existence of a solution that is optimal in space and time, provided that C satisfies certain weak computational conditions. Specifically, we describe a data structure that requires $O(n)$ space and $O(k+\log n)$ time to answer a query with output size k. The only assumption necessary to the validity of the algorithm is that constant time suffices for deciding whether a point is contained in C. A few other primitive operations must be assumed for the sake of preprocessing. If such operations can be executed in constant time, the data structure can be constructed in $O(n^2)$ time.

The generality of the setting allows a uniform solution of several problems which have been treated separately in the past. If C is a disk, the problem becomes the well-known *fixed-radius neighbour problem* (Bentley & Maurer, 1979; Chazelle, 1983*a*; Chazelle *et al.*, 1984). The best solution to this problem achieves optimal retrieval time at the cost of $O(n(\log n \log\log n)^2)$ space (Chazelle *et al.*, 1984), but also handles queries with non-fixed radius. If C is a triangle or a rectangle then we have restricted versions of the *triangular* and *orthogonal range search* problems (Knuth, 1973; Edelsbrunner *et al.*, 1982; Chazelle, 1983*b*; Cole & Yap, 1983; Edelsbrunner & Welzl, 1983; Willard, 1985). Again, optimal retrieval time is achieved only with superlinear space. Other shapes which C may assume include ellipses or hybrid convex figures bounded by a constant number of analytic curves. We look at the special case where C is a convex m-gon and m is considered a

The first author was supported in part by NSF grants MCS 83–03925 and the Office of Naval Research and the Defense Advanced Research Projects Agency under contract N00014–83–K–0146 and ARPA Order No. 4786.

variable of the problem. For this case, we describe a solution requiring $O(n+m)$ space and $O(k+\log n \log m)$ time to compute a k-point answer.

2. The Geometric and Computational Backdrop

In this section we introduce relevant geometric notions and address the computational assumptions we have to make.

Let E^2 denote the Euclidean plane and endow it with a system of Cartesian coordinates x and y. The directions determined by the x and y axes are referred to as *horizontal* and *vertical*, respectively. Let A be a subset of E^2. We assume that the reader is familiar with the concepts of interior int$A$, closure cl$A$ and boundary bd$A$. For two points $a=(a_x, a_y)$ and $b=(b_x, b_y)$, we have $a+b=(a_x+b_x, a_y+b_y)$, and for a real λ, $\lambda a=(\lambda a_x, \lambda a_y)$. These operations are naturally extended to subsets A, B of E^2, i.e.

$$A+B=\{a+b \mid a \in A, b \in B\} \quad \text{and} \quad \lambda A=\{\lambda a \mid a \in A\}.$$

For any point q, $A+q=A+\{q\}$ is called a *translate* of A and is denoted A_q. A is *convex* if for any points a_1 and a_2 in A, the point $\lambda a_1+(1-\lambda)a_2$ lies in A for each λ such that $0 \leqslant \lambda \leqslant 1$. The smallest convex set that contains A is called the *convex hull* of A, denoted conv$A$. The convex hull of $\{a, b\}$ is called a *segment.*

The model of computation is the standard RAM with infinite real arithmetic—a traditional assumption in computational geometry. Let C be a convex closed figure with non-empty interior. We leave C essentially unspecified and therefore must make a minimum number of assumptions on the primitive operations allowed with respect to C. First, we consider the intersection of the boundaries of two translates of C. We define

$$S(v, w)=\text{bd}(-C)_v \cap \text{bd}(-C)_w,$$

for two points v and w in E^2. By convexity of C, $S(v, w)$ is either empty or consists of at most two possibly degenerate segments, and thus can be represented in a constant amount of space. This concept is naturally extended to the case where v and w are infinitesimally close: this gives

$$S(v, w)=S(v, l)=\text{bd}(-C)_v \cap \text{bd}(-C+l),$$

for l the line that contains v and w (see Fig. 1 for an illustration of the two cases).

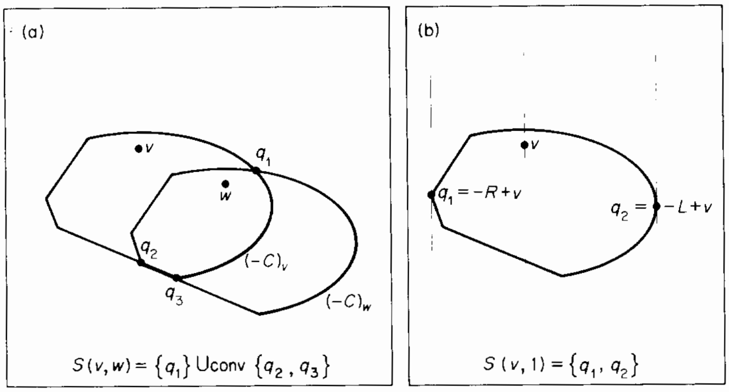

Fig. 1. Intersection of boundaries of two translates.

We call C *computable* if (i) constant time suffices to test for any point p in E^2 whether or not p is contained in C, and (ii) constant time suffices to compute $S(v, w)$ for any two, potentially infinitesimally close, points v and w in E^2. In the remainder of this section, we elaborate on the primitive operations needed and introduce the notion of *silo*. Let L (resp. R) in bd$C$ be the point with minimal (resp. maximal) x-coordinate, and maximal y-coordinate if not unique.

LEMMA 1. *If C is computable, L and R can be determined in constant time.*

PROOF. Let l be the vertical line through the origin $O=(0,0)$. Since C is computable, $S(O, l)$ and the lower endpoints a and b of the two vertical segments that constitute $S(O, l)$ can be determined in constant time. Let $a_x < b_x$; we then have $L=-b$ and $R=-a$.

We immediately derive

LEMMA 2. *Let C be computable, let p be a point in E^2 and l be the vertical line through p. Constant time suffices to decide whether*

1. *C is to the right of l,*
2. *$C \cap l \neq \emptyset$ and* (2.1) *p is above C,* (2.2) *p is contained in C,* (2.3) *p is below C,*
3. *C is to the left of l.*

PROOF. Since C is computable, case 2.2 can be distinguished from the other cases in constant time. If case 2.2 does not apply, then compute L and R (Lemma 1) and observe that C contains conv$\{L, R\}$. If $p_x < L_x$ (resp. $p_x > R_x$) then case 1 (resp. case 3) applies. Otherwise, case 2.1 (resp. case 2.3) applies if p_y is greater (resp. smaller) than the y-coordinate of $l \cap$ conv$\{L, R\}$.

The algorithmic part of this paper uses the so-called *silos* as substitutes for $-C$. Let v be a point and $r(v)$ be the vertical *ray* with v as lower endpoint. The figure $-C+r(v)$ is termed the *silo* $T(v)$ of v (Fig. 2). Obviously, $T(v)$ is the set of points q such that C_q intersects $r(v)$. bd$T(v)$ consists of two vertical rays $-L+r(v)$ and $-R+r(v)$, and the lower

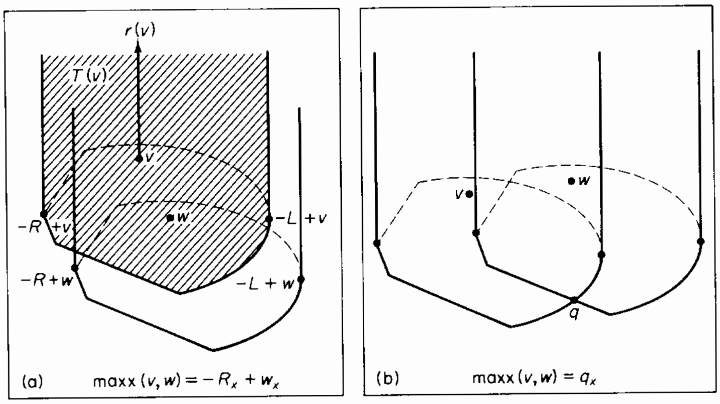

Fig. 2. Intersection of two silos.

bd$(-C)_v$. The silos of two points v, w are translates of each other. Therefore,
$\cap\, \mathrm{bd}T(w)$ is empty or consists of a single point, segment, or ray, provided that
. Let maxx(v, w) be the maximal x-coordinate of a point in $\mathrm{bd}T(v) \cap \mathrm{bd}T(w)$. The
g technical result will be of interest in Section 4.

3. *For any pair of points v, w in E^2 with $v_x \neq w_x$ and* $\mathrm{bd}T(v) \cap \mathrm{bd}T(w) \neq \emptyset$,
. w) can be computed in constant time.

Assume wlog that $v_x < w_x$. Three cases must be distinguished:

$R+w$ is below $(-C)_v$ (this is equivalent to "v is above C_{-R+w}" or "$v+R-w$ is
ove C" and, by Lemma 2, is decidable in constant time). Then $-R+w$ is below
(v) and maxx$(v, w) = -R_x + w_x$ (Fig. 2(a)).
$L+v$ is below $(-C)_w$. Then maxx$(v, w) = -L_x + v_x$ by the same reasoning as in
se 1.
therwise, maxx(v, w) is the x-coordinate of the rightmost point of $S(v, w)$ which lies
low conv$\{-L+u, -R+u\}$, for $u = v, w$ (Fig. 2(b)).

3. Clearing the Way for the Main Algorithm

P be a set of points in E^2 and C a computable figure: C is closed, convex, and has
npty interior. To retrieve $P \cap C_q$, for query point q, we store subsets of points in
te structures. These subsets are defined by a regular decomposition of E^2 into
logram. This section describes the particular decomposition and shows the
nce of silos.

L and R be the extreme points of C as introduced in the previous section. The
nt $s = \mathrm{conv}\{L, R\}$ partitions C into two figures: C_a contains the points above and on
C_b the points below s. Since C is computable, so are C_a and C_b. The algorithm
ers each figure in turn. Because of symmetry, we may assume that C is now C_a
$s \subseteq \mathrm{bd}C$.

M be a point in $C \backslash s$ that allows a line through M to be both parallel to s and
nt to C (Fig. 3). L, R, and M induce the decomposition of E^2 in the following way
$= \frac{1}{2}(R-L)$ and $Y = \frac{1}{2}(M-N)$, with N the vertical projection of M onto s. W
e the coordinate system so that point $(i, j) = iX + jY$, and cal
$p = (p_x, p_y) | i \leqslant p_x < i+1,\ j \leqslant p_y < j+1$ and i, j integers$\}$ a *cell*. $\mathcal{G} = \{c_{ij}\}$ for a
rs i, j is a decomposition of E^2 (Fig. 3). Note that M can be determined in constan

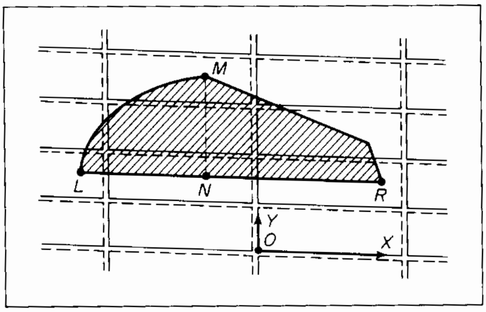

Fig. 3. Decomposing C and E^2.

time, and that the new coordinates of a given point can be computed in constant time. The limited interaction between the cells of $\mathcal{G}$ and translates of C justifies the introduction of $\mathcal{G}$. We have

LEMMA 4. *For any q in E^2, C_q intersects at most nine cells of $\mathcal{G}$.*

PROOF. By construction of $\mathcal{G}$, C_q intersects cells of three consecutive rows and columns of $\mathcal{G}$. Nine cells lie in the intersection of three rows and three columns.

If C_q intersects a cell c, the intersection always is of a particular kind. Let N, E, S, W denote the four edges of bd$c$ with the natural association of north–above, east–right, south–below, and west–left. We say that C_q is D-grounded if $C_q \cap D$ equals the orthogonal projection of $C_q \cap$ cl$c$ onto D, for $D = N, E, S, W$. C_q is said to be *grounded* if it is D-grounded for at least one assignment of D to N, E, S or W. We have

LEMMA 5. *Let q be a point in E^2 and c a cell of $\mathcal{G}$ such that $C_q \cap c \neq \emptyset$. C_q is grounded with respect to c.*

PROOF. Let T be the triangle with vertices $L+q$, $R+q$ and $M+q$. Evidently, $T \subseteq C_q$ and C_q is D-grounded if T is. Define $s_1 = \text{conv}\{L+q, R+q\}$, $s_2 = \text{conv}\{R+q, M+q\}$, and $s_3 = \text{conv}\{M+q, L+q\}$. Observe that

if T is not S-grounded then int$c \cap s_1 \neq \emptyset$,
if T is not E-grounded then int$c \cap s_2 \neq \emptyset$,
if T *is not* W-grounded then int$c \cap s_3 \neq \emptyset$.

So T is not grounded only if int$c$ intersects each edge of T. We prove that this is not possible: to obtain int$c \cap s_1 \neq \emptyset$, s_1 must be above S. The line supporting N then intersects T in a segment longer than one. As a consequence, N cannot intersect s_2, say, and so neither can E.

Note that we proved more than is asserted in Lemma 5: C_q is guaranteed to be D-grounded for at least one assignment of D to E, S, or W.

Lemmas 4 and 5 form the basis of the algorithm. The idea is to identify the at most nine cells that intersect C_q, and for each cell retrieve the points of P in C_q. Lemma 5 guarantees that for each cell the interaction with C_q is grounded. Each non-empty cell c (that is, $c \cap P \neq \emptyset$) is equipped with a data structure $\Delta_D(c \cap P)$, for each direction D from $\{N, E, S, W\}$. A high-level description of the algorithm for answering a query C_q follows:

```
Step 1: Determine the non-empty cells in $\mathcal{G}$ that intersect $C_q$.
Step 2: For each such cell $c$ do:
    Step 2.1: Find an assignment of $D$ to $N, E, S, W$ such that $C_q$ is $D$-grounded with
              respect to $c$.
    Step 2.2: Use $\Delta_D(c \cap P)$ to retrieve the points in $c \cap P \cap C_q$.
```

Next we sketch possible implementations of Steps 1 and 2.1 and relegate Step 2.2 to Section 4.

COMPUTING INTERSECTING CELLS

After some preliminary sorting, the non-empty cells c_{ij} in $\mathcal{G}$ can be computed in $O(n \log n)$ time, with $n = |P|$. These cells can be stored in a linear array, lexicographically sorted with respect to (i,j). Using binary search (Lemma 2), $O(\log n)$ time suffices to determine the rows that contain cells which intersect C_q, and then to identify all cells that intersect C_q.

COMPUTING GROUND ORIENTATION

Let C_q be a query translate of C and let c_{ij} be a cell in $\mathscr{G}$ which intersects C_q. The following procedure can be used to determine an assignment of D to E, S or W such that C_q is D-grounded.

Case 1: C contains $(i,j)-q$ or $(i+1,j)-q$; then set $D:=S$.
Case 2: C contains $(i,j+1)-q$; then set $D:=W$.
Case 3: C contains $(i+1,j+1)-q$; then set $D:=E$.

4. The Main Retrieval Algorithm

The four (or three) data structures to be assigned to a non-empty cell c are built according to the same procedure; so we limit our presentation to S-grounded queries. Recall the subproblem to be solved: given a non-empty cell c and a query point q such that C_q is S-grounded with respect to c, retrieve all points in $P \cap c \cap C_q$. The basic approach is to specify the locus of queries q yielding the same answer. Exhaustive characterisation of the regions is prohibitively expensive, however, but the philosophy of *filtering search* can be applied to keep down the storage requirement (Chazelle, 1983*b*). In a nutshell, this notion involves amortising the search over the output size.

Although straightforward, the following facts are fundamental for our approach.

OBSERVATION 6. *Point p lies in C_q if and only if q lies in $(-C)_p$.*

Define $P_c = P \cap c$. We have

LEMMA 7. *Let c be a cell such that C_q is S-grounded, and let p be a point of P_c. Then p is in C_q if and only if q is in $T(p)$.*

PROOF. If $p \in C_q$ then $q \in (-C)_p \subseteq T(p)$. Conversely, if $q \in T(p)$ then $p \in -T(-q) = C_q + r$, with $r = \{(0, r_y) | r_y \leqslant 0\}$. By S-groundedness of C_q, $c \cap C_q + r = c \cap C_q$ and $p \in C_q$.

Lemma 7 suggests replacing each point p of P_c by silo $T(p)$ and reporting all silos that contain q (Fig. 4). We develop this idea next: let $P_c = \{p_1, \ldots, p_m\}$, with the ordering such that p_i precedes p_j in lexicographical order if $i<j$. Since the difference in x-coordinates of any two points in P_c is less than one,

$$U = \bigcup_{1 \leqslant i \leqslant m} T(p_i)$$

is connected. We call $L(P_c) = \mathrm{bd}\, U$ the *layer* of P_c. By definition of $T(p)$, $L(P_c)$ is an unbounded connected curve, monotone in x, i.e. any vertical line intersects $L(P_c)$ in at most one (possibly degenerate) segment (Fig. 4). $L(P_c)$ can be considered to consist of a sequence of "edges"; for p_i in P_c, we call

$$L(P_c) \cap \mathrm{bd}\, T(p_i) - \left(\bigcup_{1 \leqslant j < i} \mathrm{bd}\, T(p_j) \right)$$

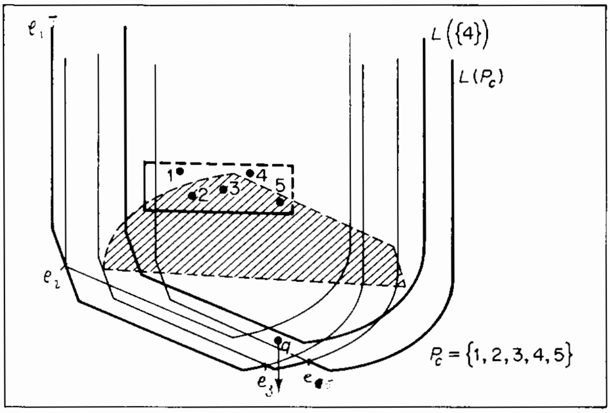

Fig. 4. Nested layers of silos.

the *edge* e_i of P_c. Obviously, two distinct edges cannot properly intersect and

$$L(P_c) = \bigcup_{1 \leq i \leq m} e_i.$$

Intuitively, e_i is the contribution of $T(p_i)$ to $L(P_c)$. If some part of $L(P_c)$ is contained in $\mathrm{bd}T(p_j)$, for several j, then the smallest j is chosen. If e_i is empty, then p_i is called *redundant* and we define $\mathrm{ext}P_c = \{p \in P_c | p \text{ non-redundant}\}$. We describe salient properties of $L(P_c)$ and detail its construction.

OBSERVATION 8. *Let* $e_{k_1}, \ldots, e_{k_t}$ $(t = |\mathrm{ext}P_c|)$ *be the sequence of edges on* $L(P_c)$ *from left to right.*

(i) $maxx(p_{k_i}, p_{k_{i+1}})$ *is the x-coordinate of the right endpoint of* e_{k_i} *and the left endpoint of* $e_{k_{i+1}}$, *for* $1 \leq i \leq t-1$ *(see also Lemma* 3).

(ii) $k_i < k_{i+1}$, *for* $1 \leq i \leq t-1$.

For the construction of $L(P_c)$, we choose to represent its edge-list in a linear array A, so that the indices of the edges in sorted order appear from left to right. The construction takes the points in the order of their indices (so we assume that an initial sorting phase has been previously performed). When p_i is processed, $L(\{p_1, \ldots, p_{i-1}\})$ is treated as a stack: a possibly empty sequence of points at the end of A is deleted and p_i is possibly added. A formal description of the algorithm follows:

```
$A[1] := 1$; $top := 1$;
**for** $i := 2$ **to** $m$ **do**
  **unless** $p_i$ and $p_{A[top]}$ share the $x$-coordinate **then**
    **while** $top > 1$ **and** $maxx(p_{A[top-1]}, p_{A[top]}) \geq maxx(p_{A[top]}, p_i)$ **do**
      $top := top - 1$
    **endwhile**
    $top := top + 1$; $A[top] := i$
  **endunless**
**endfor**
```

It is obvious that $L(P_c)$ can be constructed in $O(m)$ time. The following result is crucial for the ensuing developments.

LEMMA 9. *Let $L(P_c)=(e_{k_1},\ldots,e_{k_t})$, C_q a translate of C that is S-grounded with respect to c, and $r=\{(q_x,r_y)|r_y\leqslant q_y\}$.*

(i) *r intersects $L(P_c)$ if and only if $C_q\cap P_c$ is non-empty.*

(ii) *Let $e_{k_l}\cap r\neq\emptyset$; then p_{k_l} lies in C_q.*

(iii) *There are indices i and j, with $i\leqslant l\leqslant j$, such that $C_q\cap \mathrm{ext}P_c=\{p_{k_i},\ldots,p_{k_j}\}$.*

PROOF. Assertions (i) and (ii) follow directly from Lemma 7. To prove (iii), assume that C_q contains p_{k_i} and p_{k_j} but not p_{k_m} for some $i<m<j$. By Lemma 7, q lies in $T=T(p_{k_i})\cap T(p_{k_j})$. Since p_{k_m} is not redundant, $T\subseteq T(p_{k_m})$ which implies that C_q contains p_{k_m}—a contradiction.

If $L(P_c)=(e_{k_1},\ldots,e_{k_t})$ and C_q are given, the points in $C_q\cap \mathrm{ext}P_c$ can be retrieved in $O(\log k_t)$ time as follows:

Step 1: Using binary search, determine l such that e_{k_l} intersects with r, the vertical ray falling down from q (if $1<l<k_t$ then $\mathrm{maxx}(p_{k_{l-1}},p_{k_l})<q_x\leqslant \mathrm{maxx}(p_{k_l},p_{k_{l+1}})$).

Step 2: If l exists then set $i:=j:=l$. While $p_{k_{i-1}}$ (resp. $p_{k_{j+1}}$) is in C_q, report it and set $i:=i-1$ (resp. $j:=j+1$).

We are now ready to describe the underlying data structure $\Delta_S(P_c)$. Essentially, $\Delta_S(P_c)$ is the (nested) family $\mathscr{L}(P_c)=(L_1,\ldots,L_k)$ of *layers* of P_c defined as follows: Let $P_{c,1}=P_c$ and $P_{c,k}=P_{c,k-1}-\mathrm{ext}P_{c,k-1}$, for $k>1$. Let K be the largest index such that $P_{c,K}\neq\emptyset$, and $L_k=L(P_{c,k})$ for $1\leqslant k\leqslant K$. $\mathscr{L}(P_c)$ necessitates $O(m)$ space and can be easily constructed in $O(m^2)$ time. Furthermore,

OBSERVATION 10. *Let $\mathscr{L}(P_c)=(L_1,\ldots,L_K)$ be the family of layers of P_c and let C_q be S-grounded with respect to c. Let r be the vertical ray falling down from q. Then $r\cap L_i\neq\emptyset$ only if $r\cap L_j\neq\emptyset$, for $j<i$.*

This remark leads to a procedure for retrieving the points in $C_q\cap P_c$.

```
i := 1; stop := false;
repeat determine edge e of L_i that intersects r.
  if e exists then
    report the points in C_q ∩ extP_{c,i} and set i := i + 1
      else set stop := true
  endif
until stop
```

The complexity of the procedure is $O(k\log m)$, where k is the number of points reported. $O(k+\log m)$ can be achieved by applying the *hive-graph* of Chazelle (1983*b*) to $\mathscr{L}(P_c)$. This graph connects L_i with L_{i-1}, for $2\leqslant i\leqslant K$, in such a way that

(i) the knowledge of the edge in L_{i-1} that intersects r allows us to find the intersecting edge in L_i in constant time,

(ii) $O(m)$ space suffices to store $\mathscr{L}(P_c)$.

To describe the application of the hive-graph, it is convenient to change the representation of a layer L_i: in addition to $(k_1,\ldots,k_t)$ we store the sequence B_i of x-coordinates $(\mathrm{maxx}(p_{k_1},p_{k_2}),\ldots,\mathrm{maxx}(p_{k_{t-1}},p_{k_t}))$, $k_t=|\mathrm{ext}P_{c,i}|$. B_i subdivides the x-axis

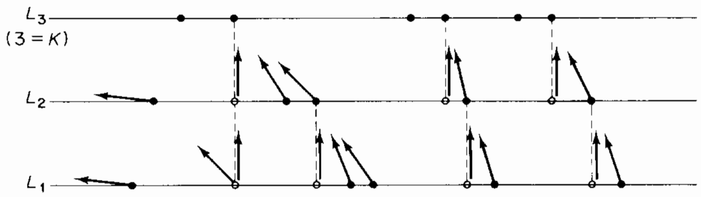

Fig. 5. Connecting layers with the hive-graph.

into t non-overlapping intervals, each belonging in the obvious manner to an index in $\{k_1, \ldots, k_t\}$. The hive-graph is obtained by the following procedure:

```
**set** $B_k^* := B_{ki}$
**for** $i := K-1$ **to** 1 **do**
    pick every other $x$-value in $B_{i+1}^*$ and merge the chosen values with $B_i$. Let $B_i^*$ be
    the resulting list. For each value $a$ in $B_i^*$, set up a pointer to the largest $b$ in $B_{i+1}^*$
    with $b \leqslant a$.
**endfor**
```

Figure 5 illustrates the connections between schematic layers that are introduced by the hive-graph. It is a straightforward exercise to verify that the hive-graph satisfies properties (i) and (ii) above. Details can also be found in Chazelle (1983*b*). All this leads to the main result of this paper.

THEOREM 11. *Let P be a set of n points in E^2 and C a convex computable figure. There exists a data structure that stores P in $O(n)$ space such that $O(k+\log n)$ time suffices to retrieve all points of P lying inside a query translate C_q. The data structure can be constructed in $O(n^2)$ time.*

We list a number of immediate consequences.

COROLLARY 12. *For n points in E^2, $O(n)$ space and $O(k+\log n)$ time suffice to retrieve the k points within a fixed radius of an arbitrary query point.*

It seems worthwhile to note that for the fixed radius problem C is a disk, thus making the partition into an upper and lower part unnecessary since both parts lead to the same decomposition of E^2. Another interesting special case arises when C is a convex polygon with m edges and m is considered a parameter of the problem. In this case, C violates the requirement of being computable. This can be fixed by allowing the primitive operations to take $O(\log m)$ time (this is motivated by the existence of a trivial algorithm for detecting whether a point lies inside a convex m-gon in $O(\log m)$ time). Our method yields

COROLLARY 13. *Let C be a convex polygon with m edges in E^2. A set P of n points can be stored in $O(n+m)$ space such that $O(k+\log m \log n)$ time suffices to retrieve the k points of P lying inside a query translate of C.*

5. Discussion

The problems examined in this paper fall into a general class involving a fixed set of objects and d-dimensional queries. A query is d-dimensional if it can be specified by d

parameters; in this paper $d = 2$. By contrast, more difficult problems like orthogonal or triangular range search are respectively four- and six-dimensional. Studying the complexity of retrieval problems along this taxonomy appears to be of great methodological value. In general, can the complexity of query problems be asserted within a theory based on the dimensionality of queries?

Let us briefly recall the main tools used in this paper:

1. transforming the problem,
2. using a decomposition of E^2 into cells and introducing silos,
3. exploiting layers of silos to speed up searching,
4. connecting layers into a hive-graph.

All of these ideas have appeared—in most cases separately—in the computational geometry literature: layers and transformations were exploited to give an optimal solution to the case where halfplanes are query ranges (Chazelle *et al.*, 1983). There, too, the structure shared the connection of layers into a hive-graph. The notion of silos is closely related to the somehow dual concept of α-hull developed in Edelsbrunner *et al.* (1983). Finally, using decompositions into cells seems almost as old as the field itself (Knuth, 1973).

References

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

Chazelle, B. (1983*a*). An improved algorithm for the fixed-radius neighbor problem. *Inform. Process. Lett.* **16,** 193–198.

Chazelle, B. (1983*b*). Filtering search: A new approach to query-answering. *Proc. 24th Ann. Symp. Found. Comp. Sci.*, 122–132.

Chazelle, B., Cole, R., Preparata, F. P., Yap, C. K. (1984). *New Upper Bounds for Neighbor Searching.* Technical Report CS–84–11. Providence: Brown University.

Chazelle, B., Guibas, L. J., Lee, D. T. (1983). The power of geometric duality. *Proc. 24th Ann. Symp. Found. Comp. Sci.*, 217–225.

Cole, R., Yap, C. K. (1983). Geometric retrieval problems. *Proc. 24th Ann. Symp. Found. Comp. Sci.*, 112–121.

Edelsbrunner, H., Kirkpatrick, D. G., Maurer, H. A. (1982). Polygonal intersection search. *Inform. Process. Lett.* **14,** 74–79.

Edelsbrunner, H., Kirkpatrick, D. G., Seidel, R. (1983). On the shape of a set of points in the plane. *IEEE Trans. Inform. Theory*, **IT-29,** 551–559.

Edelsbrunner, H., Welzl, E. (1983). *Halfplanar Range Search in Linear Space and $O(n^{0.695})$ Query Time.* Report F111, Institutes of Information Processing, Tech. Univ., Graz, Austria.

Knuth, D. E. (1973). *Sorting and Searching—The Art of Computer Programming*, III. Chapter 6.5. Reading: Addison-Wesley.

Willard, D. E. (1985). New data structures for orthogonal queries. *SIAM J. Comput.* (in press).