Discrete Comput Geom 2:113-126 (1987)

Linear Space Data Structures for Two Types of Range Search*

Bernard Chazelle[1] and Herbert Edelsbrunner[2]

[1] Department of Computer Science, Princeton University, Princeton, NJ 08544, USA

[2] Department of Computer Science, University of Illinois, Urbana, IL 61801, USA

Communicated by David Dobkin

Abstract. This paper investigates the existence of linear space data structures for range searching. We examine the *homothetic range search problem*, where a set S of n points in the plane is to be preprocessed so that for any triangle T with sides parallel to three fixed directions the points of S that lie in T can be computed efficiently. We also look at *domination searching* in three dimensions. In this problem, S is a set of n points in E^3 and the question is to retrieve all points of S that are dominated by some query point. We describe linear space data structures for both problems. The query time is optimal in the first case and nearly optimal in the second.

1. Introduction

Let S be a set of n points in d-dimensional Euclidean space E^d and let D be a domain of subsets of E^d called *ranges. Range searching* with respect to S and D refers to the task of preprocessing S so that for any $q \in D$, the subset of points of S that lie in q can be computed effectively. Typically, D is the set of all ranges patterned after some fixed shape, e.g., rectangles, disks, triangles in E^2, tetrahedra in E^3, etc. In all cases, the understanding is that the preprocessing is a one-shot operation whose cost is amortized over many queries. For this reason, it is traditional to measure the performance of a range search algorithm by means of $S(n)$, the storage required, and $Q(n)$, the time needed for answering any query. Let $S_q = S \cap q$ denote the set to be computed. Two important classes of range

* This research was conducted while the first author was with Brown University and the second author was with the Technical University of Graz, Austria. The first author was supported in part by NSF Grant MCS 83-03925.

searching need be distinguished. In *count-mode*, range searching involves computing only the cardinality of S_q, whereas in *report-mode*, every element of S_q is to be computed explicitly.

These two modes of operations often widely differ in complexity. One reason for the discrepancy comes from the opportunity offered in report-mode to amortize the search cost over the individual points of the output [2], [11], [16]. The existence of fairly efficient range search algorithms for a variety of problems motivates the following kind of questions. What problems can be solved within a given time and/or space complexity? In particular, what can be done—and how efficiently—if only linear storage is available? The main contribution of this paper is to propose a number of linear space algorithms for range search problems in E^2 and E^3. Before proceeding any further, let us introduce some terminology. *Homothetic range searching* in E^2 has the specifications: S is a set of n points in E^2; D is the set of triangles with sides parallel to three fixed directions. *Domination search* in E^3 refers to: S is a set of n points in E^3; D is the set $\{(-\infty, x]\times(-\infty, y]\times(-\infty, z] \mid x, y, z \in \Re\}$. We summarize our main results; k denotes the output size.

1. Homothetic range searching in E^2: $S(n)=O(n)$ and $Q(n)=O(k+\log n)$.
2. Domination search in E^3: $S(n)=O(n)$ and $Q(n)=O((k+1)\log n)$.
3. Domination search in E^3: $S(n)=O(n)$ and $Q(n)=O(k+\log^2 n)$.

The preprocessing time for these three solutions is, respectively, $O(n\log n)$, $O(n^2)$, and $O(n\log^2 n)$. The complexity class of interest in this work is characterized by the conditions: $S(n)=O(n)$ and $Q(n)=O(k+\log^c n)$, for some constant c. The main contribution of our work lies in the fact that neither of the problems listed above was known to be in this class before. The only (major) range searching algorithms previously proven in this class are:

1. D is the set of isothetic rectangles adjacent to a fixed line [16], [2] (recall that a figure is isothetic if it is made of edges parallel to the axes).
2. D is the set of halfplanes [7].
3. D is the set of trapezoids with two right angles adjacent to a fixed line [6] (note that the last two problems are special cases of this one).
4. D is the set of translates of a fixed convex range [5].

In [3] it is shown that in count-mode orthogonal range searching ($D=[x_1, x_2]\times[y_1, y_2]$) can be done in linear space and logarithmic time, but in report-mode a multiplicative factor of $\log^\varepsilon n$ must be included to either time or space. Another complexity class worthy of interest in this context is characterized by the conditions: $S(n)=O(n)$ and $Q(n)=O(k+n^\alpha)$, for some constant $\alpha<1$. The following problems have been shown to belong to this class: (1) S is a set of n points in E^2, D is the set of all triangles [18], [9]; (2) S is a set of n points in E^3, D is the set of all tetrahedra [19]. See also [8] for more general sets of problems in the complexity class in question.

All the results of this paper are based on an optimal solution to a *paper-stabbing problem*. Suppose that you have n sheets of paper attached to one corner of your desk; assume that sheets are different in size and shape but that none of them is completely hidden behind any other. A query comes as a needle which you poke through the first k sheets at an arbitrary point in the desk. The problem is to enumerate these sheets in optimal time and space. Our solution to this problem can be viewed as a generalization of McCreight's *priority search tree* [16]. Since the underlying structure is an acyclic directed graph instead of a tree, one might call it a *priority search dag.*

2. On a Paper-Stabbing Problem

Let (O, xy) be a Cartesian system of reference for the Euclidean plane. The coordinates of a point p are denoted (p_x, p_y). We say that a point p *dominates* a point q, a property denoted $q < p$, if and only if $q_x \le p_x$ and $q_y \le p_y$. Let $S = (p_1, \ldots, p_n)$ be a sequence of points $p_i = (x_i, y_i)$ satisfying the following:

Appearance Property: for any i, j, the relation $p_i < p_j$ implies $i < j$.

The appearance property is akin to topological sorting. Informally, it stipulates that applying the *painter's algorithm* to the rectangles $R_i = \{p \in E^2 | p < p_i\}$, in the order $i = n, \ldots, 1$, leaves each rectangle at least partly visible (Fig. 1). For any point $q \in E^2$ and any integer k $(1 \le k \le n)$, define $S_{q,k}$ to be the set of points in $\{p_1, \ldots, p_k\}$ dominating q. We formulate the *paper-stabbing problem* as follows:

Preprocess S so that for any q in E^2 and any integer k (with $1 \le k \le n$), the set $S_{q,k}$ can be computed efficiently.

Before proceeding with the detailed description of our solution, a word on the intuition behind it might be useful. Since only $O(n)$ space is allowed, there is little more we can do than form the planar graph of the visible parts of rectangles R_i (Fig. 1), and preprocess it for efficient point location [15], [13], [10] (i.e., retrieval of the face containing an arbitrary query point). This allows us to find the face containing q in logarithmic time. After this preliminary step, we will

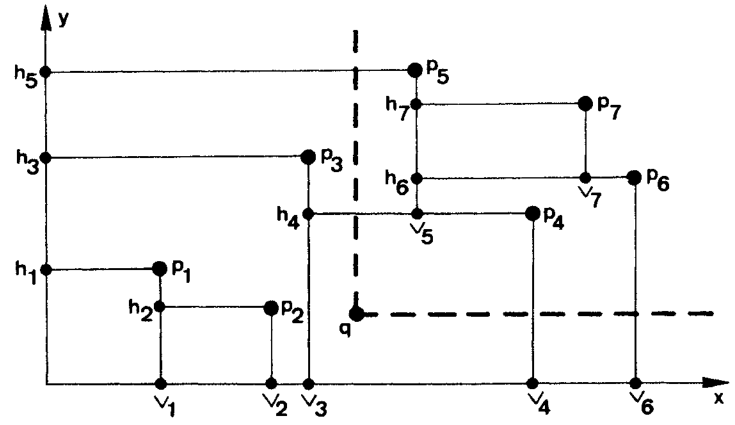

Fig. 1. The map for $(p_1, p_2, \ldots, p_7)$ and query point q.

attempt to cross through the subdivision both upward and rightward. Every edge in the subdivision corresponds to some point in S, so we might want to stop the traversal upon encountering points outside of the desired range (determined by k). This may fail to give all the points of $S_{q,k}$, so additional exploration based on some particular face ordering will be needed. We next substantiate this intuition.

2.1. *The Map of S, Its Properties, Its Construction*

Without loss of generality, assume that all points of S lie in the north-east quadrant, and for convenience that all x_i (resp. y_i) are pairwise distinct (ties, if any, can be broken arbitrarily). We define the *map* of S, denotes $\mathcal{M}(S)$, to be the isothetic planar subdivision obtained as follows (Fig. 1): for each $i=1,\dots,n$ in turn, extend a horizontal segment p_ih_i and a vertical segment p_iv_i from p_i, until hitting either another segment or one of the axes. More formally, let p_j be the point of S with maximal x-coordinate such that $j<i$ and $y_i<y_j$. If p_j exists, we have $h_{ix}=x_j$, else $h_{ix}=0$, where h_{ix} is the x-coordinate of point h_i. In all cases, $h_{iy}=y_i$. Similarly, $v_{ix}=x_i$; let p_l be the point of S with maximal y-coordinate such that $l<i$ and $x_i<x_l$. If p_l exists, we have $v_{iy}=y_l$, else $v_{iy}=0$. The point p_i is called the *anchor* of any edge on the segments p_ih_i or p_iv_i.

Lemma 1. *Let s be any vertical (resp. horizontal) line segment that does not pass through any point of S. Consider the bottom-up (resp. left-to-right) sequence of edges of $\mathcal{M}(S)$ intersected by s and let $(p_{i_1},\dots,p_{i_t})$ be the corresponding sequence of anchors. We necessarily have $i_1<\cdots<i_t$.*

Proof. Because of symmetry, we restrict ourselves to the "vertical" case. Let s_x be the x-coordinate of s. Since s does not contain any point of S, for each $1\le l\le t$, we have

$$h_{i_lx}\le s_x<x_{i_l}. \tag{1}$$

For the sake of contradiction, assume that for some l $(1\le l<t)$ we have $i_{l+1}<i_l$ (Fig. 2). From the appearance property of S and the definition of p_{i_l}, we have $p_{i_l}\not< p_{i_{l+1}}$ and $y_{i_l}<y_{i_{l+1}}$, hence $x_{i_l}>x_{i_{l+1}}$. But from the definition of h_{i_l} and $i_{l+1}<i_l$, this leads to $h_{i_lx}\ge x_{i_{l+1}}$ (Fig. 2), which contradicts (1). □

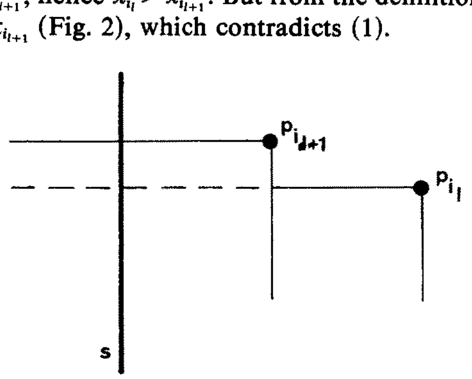

Fig. 2. $i_{l+1}<i_l$ is impossible by construction.

Next we show how to set up the map $\mathcal{M}(S)$. We assume that $\mathcal{M}(S)$ is represented by any of the standard structures for planar subdivisions: the DCEL (*doubly-connected-edge-list* representation [17] or the *quad-edge* structure [12]). Computing $\mathcal{M}(S)$ in $O(n \log n)$ time is elementary. The construction proceeds incrementally, by inserting each point $p_1, \ldots, p_n$ in this order. Let M_i be the subset of maxima in $S_i = \{p_1, \ldots, p_i\}$, i.e., $M_i = \{p \in S_i \mid p \not< q, \text{ for all } q \in S_i \backslash \{p\}\}$. We can represent M_i in a dynamic search tree, sorted by y-coordinates. Inserting p_{i+1} involves searching for y_{i+1} in the tree, computing h_{i+1}, traversing $\mathcal{M}(S_i)$ to find v_i, updating the map, deleting dominated points from the tree, and adding p_{i+1} to it. These operations are standard enough to make further elaboration unnecessary.

As will appear shortly, we need an efficient method for solving the following retrieval problem. Let q be a point in E^2 with $R_y = \{(x, y) \mid x = q_x \text{ and } y \geq q_y\}$ and $R_x = \{(x, y) \mid y = q_y \text{ and } x \geq q_x\}$. Let $I_y(q)$ (resp. $I_x(q)$) be the sequence of intersections between R_y (resp. R_x) and the edges of $\mathcal{M}(S)$, sorted by increasing y-coordinates (resp. x-coordinates). Preprocess $\mathcal{M}(S)$ so that for any $q \in E^2$, the points of $I_y(q)$ and $I_x(q)$ can be computed in $O(1)$ time per report, after $O(\log n)$ time preliminary work. A data structure, known as a *hive-graph*, has been described in [2] for solving precisely this problem. Roughly speaking, the idea is to refine the subdivision to allow efficient traversal in a preassigned direction. With the hive-graph, the point in $I_y(q)$ (resp. $I_x(q)$) are visited and reported in the correct order. It will be crucial later on to be able to stop this process at an arbitrary point, without paying the price for the remaining points in $I_y(q)$ (resp. $I_x(q)$). We will not detail the method here, but roughly speaking, it involves building a closely knit subdivision over the set of segments and preprocessing it for efficient point location. The preprocessing required by the algorithm takes $O(n \log n)$ time and the space used by the data structure is $O(n)$.

2.2. *Completing the Data Structure*

As we mentioned earlier, being able to cross through $\mathcal{M}(S)$ along vertical or horizontal segments may not be quite sufficient to solve the paper-stabbing problem. Consider query point q together with $k = 7$ in Fig. 1. Points p_4, p_5, and p_6 anchor edges which intersect the vertical and horizontal rays emanating from q; this is not true for p_7 which, however, is in $S_{q,7}$. To overcome this difficulty, we use a directed graph $G = (V, E)$, whose vertices are in one-to-one correspondence with the points of S and whose edges express the segment adjacencies in $\mathcal{M}(S)$. The graph G is defined as follows:

$$V = \{w_1, \ldots, w_n\}; \qquad E = \{(w_i, w_j) \mid h_j \in p_i v_i \text{ or } v_j \in p_i h_i\}$$

(Fig. 3). We adopt a node-based representation whereby each node w_i of G has associated with it a linked list $E(w_i)$ of outgoing edges. $E(w_i) = \{w_{i_1}, \ldots, w_{i_{k_i}}\}$, with $i_1 < \cdots < i_{k_i}$ and $(w_i, w_{i_1}), \ldots, (w_i, w_{i_{k_i}}) \in E$.

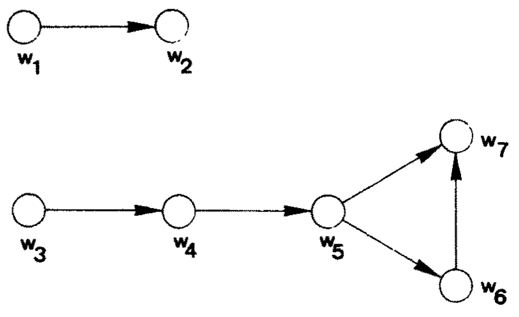

Fig. 3. Directed graph for the map in Fig. 1.

We complete the description of the data structure by mentioning that each edge in $\mathcal{M}(S)$ should have a pointer to its *supporting* node in G (an edge e is said to be *supported* by w_i iff it lies on either p_ih_i or p_iv_i). Of course, for any i ($1 \leq i \leq n$) point p_i should be retrievable in constant time from w_i. It is clear that G with all its required pointers can be computed in $O(n)$ time once $\mathcal{M}(S)$ is available in DCEL or quad-edge form. We omit the details. Next we list some of the salient properties of G.

Lemma 2. *G is acyclic and each node has indegree at most two.*

Proof. Whether h_i lies on p_jv_j or v_i lies on p_jh_j, the inequality $j < i$ holds, therefore G is acyclic. Since h_i and v_i lie on unique segments of $\mathcal{M}(S)$, the indegree is at most two. □

Note in passing that G is not necessarily connected (Fig. 3).

2.3. *The Query Algorithm*

We are now ready to describe the algorithm for computing $S_{q,l} = \{p_i \in S \mid q < p_i$ and $i \leq l\}$, given a query (q, l). For convenience, we assume that $q_x \neq x_1, \ldots, x_n$ and $q_y \neq y_1, \ldots, y_n$. Recall that $I_x(q)$ (resp. $I_y(q)$) is the ordered sequence of intersections between $\mathcal{M}(S)$ and the upward (resp. rightward) ray from q. Let p be any point of an edge e of $\mathcal{M}(S)$; we designate by $\alpha(p)$ the anchor of e. Let

$$J_x = \{w_i \in V \mid i \leq l \text{ and } p_i = \alpha(p) \text{ for some } p \text{ in } I_x(q)\}$$

and

$$J_y = \{w_i \in V \mid i \leq l \text{ and } p_i = \alpha(p) \text{ for some } p \text{ in } I_y(q)\}.$$

As a preliminary step, we compute J_x and J_y. This can be done optimally by using the hive-graph structure mentioned in Section 2.1. This is possible because, by virtue of Lemma 1, the order of reports corresponds to increasing indices, i.e., the anchors of points in $I_x(q)$ (resp. $I_y(q)$) form an increasing sequence of indices. As a result, each sequence of reports may stop as soon as an anchor p_j $(j > l)$ is discovered. After this operation, which necessitates $O(\log n + |J_x \cup J_y|)$ time, we are ready to explore the graph G.

Initially, $S_{q,l} = \emptyset$ and we define $J = J_x \cup J_y$. If $J = \emptyset$, terminate. Otherwise, mark every node in J. Then, as long as there are marked nodes in G, pick any of them, w_i, and perform the following steps.

Step 1. $S_{q,l} \leftarrow S_{q,l} \cup \{p_i\}$. Unmark w_i.

Step 2. Let $E(w_i) = \{w_{i_1}, \ldots, w_{i_\nu}\}$ be the set of nodes emanating from w_i and let i_m be the largest index less than l, i.e. $i_m = \max\{j \mid j \in \{i_1, \ldots, i_\nu\}$ and $j \le l\}$. If i_m is well defined, mark nodes $w_{i_1}, \ldots, w_{i_m}$.

Theorem 3. *Let S be a sequence of n points in the north-east quadrant which satisfies the appearance property. There is a data structure that takes $O(n \log n)$ time for construction and $O(n)$ space, such that for any query (q, l) the points in $S_{q,l}$ can be reported in $O(|S_{q,l}| + \log n)$ time. This is optimal.*

Proof. We successively establish the correctness of the method described above (part 1 below) and then analyze its performance (part 2 below).

Part 1. Given the organization of each set of outgoing edges in sorted lists, it suffices to show that for each $p_i \in S_{q,l} \backslash (J_x \cup J_y)$ there exists in G a (directed) path $w_{j_1}, \ldots, w_{j_\alpha}$, with $j_\alpha = i$, $w_{j_1} \in J_x \cup J_y$, $p_{j_1}, \ldots, p_{j_\alpha} \in S_{q,l}$. We prove this fact by induction on the ascending sequence of indices in $S_{q,l}$. The basis case being obvious, let $p_i \in S_{q,l} \backslash (J_x \cup J_y)$. Since p_i is not in J_x, v_i lies on $p_j h_j$ for some j $(1 \le j \le n)$, so G has an edge from w_j to w_i. Also, $v_{iy} > q_y$ implies that $q < p_j$, and since clearly $j < i$, we have $p_j \in S_{q,l}$. By induction hypothesis there exists a directed path from some w_{j_1} to w_j. This concludes the argument.

Part 2. The computation of J_x and J_y takes $O(\log n + |J_x \cup J_y|)$ time, as already observed. The remainder of the algorithm has a complexity proportional to the number of edges of G traversed. Let us call a *good* node, a node w_i such that $p_i \in S_{q,l}$, and a *bad* node, any other. Let H be the subgraph of G induced by the good nodes. From Lemma 1, it follows that for each good node w_i at most two bad nodes need to be visited. The running time of the algorithm is therefore proportional to the number of edges in H. This number is proportional to the number of vertices in H, since each vertex has indegree at most two (Lemma 2). □

3. Homothetic Range Searching

The *homothetic range search problem* refers to the case where the query domain D is the set of all polygons obtained by submitting a fixed simple m-gon to an arbitrary translation and an arbitrary scaling transformation. More precisely, let P be the simple m-gon. A query is specified by a pair (q, c), with $q \in E^2$ and c a positive real number; the *homothet* of P is the polygon $P_{q,c} = \{p \in E^2 |$ there is a point $v \in P$ such that $p_x = q_x + cv_x$ and $p_y = q_y + cv_y\}$. The input to the problem, denoted S as usual, is a set of n points in E^2 and the set to be computed is $S_q = S \cap P_{q,c}$. In the following, the number of sides of the query polygon, m, is taken to be a constant. We state our main result.

Theorem 4. *Let S be a set of n points in E^2. In $O(n \log n)$ preprocessing, it is possible to construct an $O(n)$ space data structure so that homothetic range searching with respect to S can be done in $O(k + \log n)$ query time, where k is the output size. The method is optimal.*

Proof. By triangulating the query polygon if necessary, one can always assume that P is a triangle. We set up a coordinate system such that two sides of P are parallel to the coordinate axes, and if P is translated so as to have its two sides collinear with the coordinate axes, then P is contained in the north-east quadrant (note that this system will not be orthogonal in general). We easily ensure that each point in S lies in the north-east quadrant. Let $ax + by + 1 = 0$ be an equation of the line passing through the third side of the triangle P. In $O(n \log n)$ time, sort the points of S according to their projections on a line perpendicular to this line. Let $S = (p_1, \ldots, p_n)$ be the resulting sequence; if $p_i = (x_i, y_i)$ we have $ax_1 + by_1 \leq \cdots \leq ax_n + by_n$. It is easy to see that S has the appearance property of the previous section, so it is possible to prepare the grounds for the paper-stabbing problem. Let ABC be the query triangle, with AB (resp. AC) parallel to the x (resp. y) axis (Fig. 4). Let H be the halfplane delimited by the line passing through BC and containing ABC, and let $k = |S \cap H|$; note that k is easily computed in $O(\log n)$ time. $S \cap ABC$ is exactly the output of the paper-stabbing problem on query input (A, k). We leave the claim of optimality as an exercise. $\square$

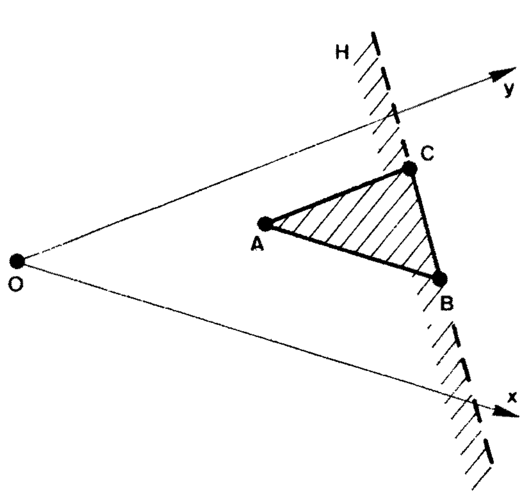

Fig. 4. Range query with triangle *ABC*.

4. The Domination Search Problem in E^3

We endow E^3 with a Cartesian system of coordinates (O, xyz). The notion of domination introduced in Section 2 generalizes easily to higher dimensions: a point $q \in E^3$ *dominates* a point $p \in E^3$, denoted $p \prec q$ iff $p_x \leq q_x$, $p_y \leq q_y$, and $p_z \leq q_z$. Let $S = \{p_1, \dots, p_n\}$ be a set of n points in E^3. *Domination searching* can be phrased as follows: preprocess S so that for any query $q \in E^3$ the set $S_q = \{p | p \in S \text{ and } p \prec q\}$ can be computed effectively. Let $S(n)$ and $Q(n)$ denote, respectively, the space and query time required by an algorithm for domination searching in E^3, and let k denote the output size. The best solution known so far achieves $S(n) = O(n \log n)$, $Q(n) = O(k + \log n)$ [11]. We will next describe two linear space data structures for this problem, one achieving $Q(n) = O((k+1) \log n)$ and the other $Q(n) = O(k + \log^2 n)$.

Let $p_i = (x_i, y_i, z_i)$. For convenience, we again assume that none of the three sets of coordinates has duplicates. The notion of *minima* is crucial to our approach. A point p_i is called a *minimum* of S if it does not dominate any other point in S. This definition carries over directly to E^2, so we may refer to minima in E^2 without further explanation. Assume for the time being that *each* point of S is a minimum. We put ourselves in the conditions of Theorem 3 by:

1. Defining a new relation $\succ$ as follows: $p \succ q$ iff $q \prec p$.
2. Relabelling the points of S so that $z_1 < \cdots < z_n$, and defining $S^* = ((x_1, y_1), \dots, (x_n, y_n))$ as a sequence of points in the plane.

Since each p_i is a minimum, it is immediate to see that the sequence S^* satisfies the appearance property with respect to $\succ$ (Section 2). Given a query q, we reduce the domination problem in E^3 to a paper-stabbing problem in which $\succ$ has been substituted for $\prec$. A query $q = (q_x, q_y, q_z)$ is transformed in $O(\log n)$ time into a query (q_x, q_y, k) for the paper-stabbing problem, with $k = |\{p_i \in S | z_i \leq q_z\}|$. We conclude with a result which will be the cornerstone of our ensuing developments.

Lemma 5. *Let S be a set of n points in E^3, all of which are minima. In $O(n \log n)$ preprocessing, it is possible to construct an $O(n)$ space data structure so that domination searching with respect to S can be done in $O(k + \log n)$ query time, where k is the output size. The method is optimal.*

4.1. *A Simple Linear-Space Solution*

From now on, S is taken to be an arbitrary set of n points in E^3. Recall that S_q, the set to be computed, consists of the points of S dominated by the query point q. In preprocessing, we compute the sequence of *layers* of S, denoted $(\mathscr{L}_1, \dots, \mathscr{L}_p)$. These layers are subsets of S obtained by removing the minima of S, computing the new set of minima, removing it, and so forth. Let $\mathscr{L}(S)$ denote the set of minima of S. The following algorithm provides a formal definition of layers, as well as a method for computing them:

```
$i \leftarrow 0$
while $S \neq \emptyset$
  begin
    $i \leftarrow i+1$
    $\mathscr{L}_i \leftarrow \mathscr{L}(S)$
    $S \leftarrow S \backslash \mathscr{L}_i$
  end
$p \leftarrow i$
```

Kung *et al.* [14] have shown how to compute the minima of a set of n points in E^3 in $O(n \log n)$ time. This leads to an $O(n^2 \log n)$ time, $O(n)$ space algorithm for computing the layers of S. This can be improved by resorting to a simpler, but more space-consuming, technique. Set up a directed graph over the points of S by placing an edge from p_i to p_j iff $p_i < p_j$. Removing all the sources of the graph gives $\mathscr{L}_1$ and iterating on this process gives $\mathscr{L}_2, \ldots, \mathscr{L}_p$ in $O(n^2)$ time and space. We omit the details.

Lemma 6. *For any i $(1 \leq i < p)$, $S_q \cap \mathscr{L}_{i+1} \neq \emptyset$ implies $S_q \cap \mathscr{L}_i \neq \emptyset$.*

Proof. Each point p in $\mathscr{L}_{i+1}$ dominates at least one point in $\mathscr{L}_i$. □

From Lemma 6, a possible line of attack follows trivially. We apply the result of Lemma 5 to $\mathscr{L}_1, \mathscr{L}_2, \ldots$ in turn, until we fail to report any point in S_q, at which stage the algorithm terminates. This leads to:

Theorem 7. *Let S be a set of n points in E^3. In $O(n^2)$ time and space, it is possible to construct an $O(n)$ space data structure so that domination searching with respect to S can be done in $O((k+1) \log n)$ query time, where k is the output size.*

4.2. *A More Efficient Algorithm*

We next show how a recursive strategy allows us to take the running time of the previous solution down to $O(k + \log^2 n)$. To be rigorous, this transformation constitutes an improvement only for values of $k = \Omega(\log n)$. Before proceeding with the description of the algorithm, we need to make a short digression. Let V be a set of n points in E^2, with each point being a minimum. Let $V_q = \{v \in V \mid v < q\}$ be defined for any point $q \in E^2$. Domination search in two dimensions calls for computing V_q efficiently, given any query point q. Of course, this problem can be solved optimally by application of Lemma 5. A much simpler solution is based on the following remark. Let $V = \{v_1, \ldots, v_n\}$ be given by increasing x-coordinates. The points of V_q always form a contiguous chain (possibly empty) of the form $\{v_i, v_{i+1}, \ldots, v_j\}$. Computing V_q can be done in optimal $O(j - i + \log n)$ time by searching for q_x in V (regarded for this purpose as a dictionary of x-coordinates).

Observation 8. Let S be a set of n points in E^2, all of which are minima. In $O(n \log n)$ preprocessing, it is possible to construct an $O(n)$ space data structure so that domination searching can be done in $O(k+\log n)$ query time, where k is the output size. Answering a query essentially involves searching for an item in a dictionary.

This last remark about the reduction of domination search in E^2 to a simple dictionary look-up is of great importance, as will be apparent below. Let us go back to our original problem, i.e., domination search in E^3.

The data structure, denoted $D(S)$, is a binary tree defined recursively as follows:

1. If $S=\emptyset$ then $D(S)$ is the empty tree.
2. Otherwise, let S^* be the projection of S onto the yz-plane. We define P as the set of minima of S^*, i.e., $P=\{p \in S^* \mid q \not< p$ for all $q \in S^* \setminus \{p\}\}$, and $M=\{p=(p_x, p_y, p_z) \in S \mid (p_x, p_y) \in P\}$ (clearly, M is a set of minima in E^3). It is then possible to build the data structures of Lemma 5 and Observation 8 for M and P, respectively, which we denote $H(S)$ and $E(S)$. Both data structures are assigned to the root r of $D(S)$. Define $V=S\setminus M$ and let $(p_{i_1}, \ldots, p_{i_m})$ be the sequence of points in V sorted by increasing x-coordinates. Let $l=\lfloor m/2 \rfloor$, $V_1=\{p_{i_1}, \ldots, p_{i_l}\}$, and $V_2=\{p_{i_{l+1}}, \ldots, p_{i_m}\}$: $D(V_1)$ (resp. $D(V_2)$) is assigned the left (resp. right) subtree of r. Since each point of S is represented only twice in $D(S)$, the storage required for the entire data structure is clearly $O(n)$. The data structures $E(S)$ and $H(S)$ will be referred to later on as the *easy* and *hard* structures, respectively. We are now ready to describe the query algorithm.

Step 1. Using $D(S)$ as a search tree, locate the leaf corresponding to p_i such that $x_i \le q_x < x_{i+1}$. Let $v_1, \ldots, v_h$ be the corresponding search path; v_1 is the root and v_h is the leaf that stores p_i (Fig. 5).

Step 2. Query the hard structure at each v_i $(1 \le i \le h)$.

Step 3. Let W be the sequence of left children of $(v_1, \ldots, v_{h-1})$ that are *not* nodes of the search path (this sequence is obtained by tracing the search path and recording the left child of each node witnessing a right turn—see square nodes in Fig. 5). Mark every node of W.

Step 4. While $D(S)$ has marked nodes, pick any, unmark it, and query its easy structure. If this leads to any report, mark its children (if any).

The correctness of the algorithm is based on a number of observations, simple enough to have their proofs omitted.

1. All the points in S_q are stored in $v_1, \ldots, v_h$ and in the subtrees rooted at the nodes of W.
2. With respect to the subset of S associated with any node that is either in W or is a descendant of a node in W, the problem to be solved is equivalent to domination search in a set of minima in E^2.

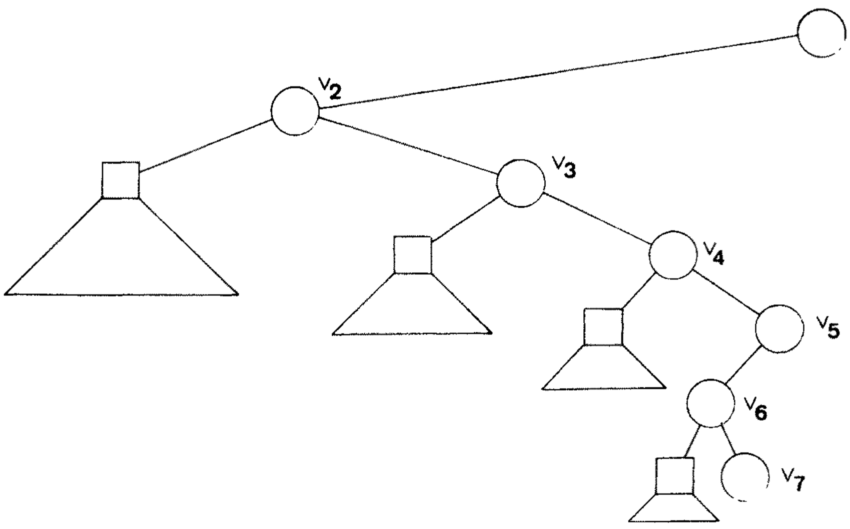

Fig. 5. Set $V=(v_1, v_2, \ldots, v_7)$ and nodes in W.

3. If the easy structure at v fails to report any point, no point stored in any descendant of v lies in S_q.

Let D_q be the subtrees of $D(S)$ visited during the computation and let $k=|S_q|$ be the output size. The algorithm takes $O(\log^2 n + k \log n)$ time, since (1) every node visited that is neither a leaf of D_q nor of the form v_j contributes at least one distinct entry to S_q; (2) each node visited requires $O(\log n)$ search time; (3) the number of nodes v_j is $O(\log n)$. This disappointing performance can be improved by exploiting the last remark of Observation 8. For the sake of clarity, a little background is necessary.

The notion of *fractional cascading*, developed in [6], is concerned with the problem of batching repeated binary searches. Let G be a connected graph whose maximum degree is bounded by a constant. With each node $w \in G$ is associated a dictionary $D(w)$ (i.e., an array of sorted numbers). Let m be the total size of all the dictionaries. Fractional cascading is a method for preprocessing G so that contiguous searches can be carried out in constant time. More precisely, if x has to be searched in $D(w_1), \ldots, D(w_t)$, where for each i, w_i is adjacent to some w_j $(j<i)$, this preprocessing allows us to do so in $O(\log m)$ time for $D(w_1)$ and then $O(1)$ time for each of the others, $D(w_2), \ldots, D(w_t)$. The interesting feature of fractional cascading is that its application increases the original size of the data structure by at most a constant factor. We also mention that the preprocessing can be done in linear time. The relevance of fractional cascading to the problem at hand is immediate. Since the easy structures are handled via a simple dictionary search, fractional cascading will allow us to handle all of them in $O(k)$ time after $O(\log n)$ preliminary work. Incidentally, note that the graph spanned by the nodes in W and their visited descendants is not connected, but these nodes

together with $\{v_1, \ldots, v_h\}$ do form a connected subgraph. Consequently, the fractional cascading scheme will have to visit the easy structures in V as well, in order to ensure the connectivity condition. This is not a problem, however, since there are only $O(\log n)$ such nodes, hence $O(\log n)$ spurious visits, at unit cost each. The preprocessing takes $O(n \log^2 n)$ time, since each node requires $O(p \log p)$ steps, where p is the number of points stored in the subtree of the node. We conclude:

Theorem 9. *Let S be a set of n points in E^3. In $O(n \log^2 n)$ preprocessing, it is possible to construct an $O(n)$ space data structure so that domination searching with respect to S can be done in $O(k + \log^2 n)$ query time, where k is the output size.*

It is possible to generalize Theorems 7 to 9 to higher dimensions. Every increase of one in dimension will result in the introduction of a factor of $\log n$ in both space and search time. The technique involves a canonical decomposition of the query into $O(\log n)$ queries of lesser dimensionality. The technique is due to Bentley [1]. It is standard and has been applied before on such numerous occasions that we will dispense with any further explanation.

Theorem 10. *Let S be a set of n points in E^d $(d > 2)$. In $O(n \log^{d-1} n)$ (resp. $O(n^2)$) preprocessing, it is possible to construct an $O(n \log^{d-3} n)$ space data structure so that domination searching with respect to S can be done in $O(k + \log^{d-1} n)$ (resp. $O(\log^{d-2} n + k \log n)$) query time, where k is the output size.*

5. Conclusions

The main contribution of this paper is the presentation of linear space data structures for two instances of range searching. In the case of three-dimensional domination, the query times achieved remain short of optimal. Can these be improved without sacrificing storage? Can the structures described in this paper be efficiently dynamized? Can they be made to perform better on the average?

Two of the techniques used in this work (paper-stabbing and recursive three-dimensional domination) follow an approach which might be compared with the *retrieve-and-explore* strategy of McCreight's priority search tree [16]. The idea is to set up a directed graph (dag and binary tree) and traverse it from each of a number of specified *source-nodes*, with the understanding that a nonsource node is visited only if its predecessor has been and that visit contributed at least one item to the output. This technique, based on some connectivity property of the encoding of each output set, seems fairly fundamental to retrieval problems (see also the solution for fixed radius circular range searching [5]). This raises the general question of studying how each output set can be encoded in a graph structure for a given range search problem: as demonstrated in [4] this seems a promising line of attack for proving lower bounds, especially in the pointer machine model.

References

1. J. L. Bentley, Multidimensional divide and conquer, *Comm. ACM* **23** (1980), 214-229.
2. B. Chazelle, Filtering search: a new approach to query-answering, *SIAM J. Comput.* **15** (1986), 703-724.
3. B. Chazelle, A Functional Approach to Data Structures and Its Use in Multidimensional Searching, Technical Report No. CS-85-16, Brown University, 1985. Preliminary version in *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science*, 165-174, 1985.
4. B. Chazelle, Lower bounds on the complexity of multidimensional searching, *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science*, 87-96, 1986.
5. B. Chazelle and H. Edelsbrunner, Optimal solutions for a class of point retrieval problems, *J. Symbolic Comput.* **1** (1985), 47-56.
6. B. Chazelle and L. J. Guibas, Fractional cascading: I. A data structuring technique, *Algorithmica* **1** (1986), 133-162.
7. B. Chazelle, L. J. Guibas, and D. T. Lee, The power of geometric duality, *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science*, 217-225, 1983. Also in *BIT* **25** (1985), 76-90.
8. D. P. Dobkin and H. Edelsbrunner, Space searching for interesting objects, *Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science*, 387-392, 1984.
9. H. Edelsbrunner and E. Welzl, Halfplanar range search in linear space and $O(n^{0.695})$ query time, *Inform. Process. Lett.*, to appear.
10. H. Edelsbrunner, L. J. Guibas, and J. Stolfi, Optimal point location in a monotone subdivision, *SIAM J. Comput.* **15** (1986), 317-340.
11. H. N. Gabow, J. L. Bentley, and R. E. Tarjan, Scaling and related techniques for geometry problems, *Proceedings of the 16th Annual ACM Symposium on Theory of Computation*, 135-143, 1984.
12. L. J. Guibas and J. Stolfi, Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams, *ACM Trans. Graphics* **4** (1985), 74-123.
13. D. G. Kirkpatrick, Optimal search in planar subdivisions, *SIAM J. Comput.* **12** (1983), 28-35.
14. H. T. Kung, F. Luccio, and F. P. Preparata, On finding the maxima of a set of vectors, *J. Assoc. Comput. Mach.* **22** (1975), 469-476.
15. R. J. Lipton and R. E. Tarjan, Applications of a planar separator theorem, *SIAM J. Comput.* **9** (1980), 615-627.
16. E. M. McCreight, Priority search trees, *SIAM. J. Comput.* **14** (1985), 257-276.
17. D. E. Muller and F. P. Preparata, Finding the intersection of two convex polyhedra, *Theoret. Comput. Sci.* **7** (1978), 217-236.
18. D. E. Willard, Polygon retrieval, *SIAM J. Comput.* **11** (1982), 149-165.
19. F. F. Yao, A 3-space partition and its applications, *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, 258-263, 1983.

Received April, 1986, and in revised form November 25, 1986.