Algorithmica (2004) 39: 275–286
DOI: 10.1007/s00453-004-1082-5

Algorithmica

Three-Dimensional Layers of Maxima[1]

Adam L. Buchsbaum[2] and Michael T. Goodrich[3]

Abstract. We present an $O(n \log n)$-time algorithm to solve the three-dimensional layers-of-maxima problem. This is an improvement over the prior $O(n \log n \log \log n)$-time solution. A previous claimed $O(n \log n)$-time solution due to Atallah et al. [2] has technical flaws. Our algorithm is based on a common framework underlying previous work, but to implement it we devise a new data structure to solve a special case of dynamic planar point location in a staircase subdivision. Our data structure itself relies on a new extension to dynamic fractional cascading that allows vertices of high degree in the control graph.

Key Words. Fractional cascading, Layers of maxima, Planar point location.

1. Introduction. A point $p \in \Re^d$ *dominates* another point $q \in \Re^d$ if each coordinate of p exceeds that of q. Given a set S of n points in $\Re^d$, the *maximum points* are those that are not dominated by any point in S. The *maxima set problem*, of finding all the maximum points in S, is a classic problem in computational geometry, dating back to the early days of the discipline [11]. Interestingly, the algorithm presented in the original paper by Kung et al. [11] is still the most efficient known method for solving this problem; it runs in $O(n \log n)$ time when $d = 2$ or 3 and $O(n \log^{d-2} n)$ time when $d \geq 4$. This problem has subsequently been studied in many other contexts, including solutions for parallel computation models [9], for point sets subject to insertions and deletions [10], and for moving points [7].

The *layers-of-maxima problem* iterates this discovery: after finding the maximum points, remove them from S and find the maximum points in the remaining set, iterating until S becomes empty. The iteration index in which a point is a maximum is defined to be its *layer*. More formally, for $p \in S$, $layer(p) = 1$ if p is a maximum point; otherwise, $layer(p) = 1 + \max\{layer(q) : q \text{ dominates } p\}$. The *layers-of-maxima problem*, which is related to the convex layers problem [4], is to determine $layer(p)$ for each $p \in S$, given S.

With some effort [1], the three-dimensional layers-of-maxima problem can be solved in time $O(n \log n \log \log n)$ using techniques from dynamic fractional cascading [13]. We sketch this result in Section 2. Atallah et al. [2] claim an $O(n \log n)$-time algorithm, but their presentation appears to have several problems. A simple, linear-time reduction from sorting gives an $\Omega(n \log n)$-time lower bound in the comparison model.

[1] An extended abstract of this work appears in *Proc. 10th Euro. Symp. on Algorithms*, Lecture Notes in Computer Science, vol. 2461, pp. 257–269, Springer-Verlag, Berlin, 2000. The second author was supported by DARPA Grant F30602-00-2-0509 and NSF Grant CCR-009806.
[2] AT&T Labs, Shannon Laboratory, 180 Park Ave., Florham Park, NJ 07932, USA. alb@research.att.com.
[3] Department of Information and Computer Science, University of California, Irvine, CA 92697-3425, USA. goodrich@ics.uci.edu.

Received June 26, 2003; revised October 15, 2003. Communicated by B. Chazelle.
Online publication February 9, 2004.

In this paper we give an $O(n \log n)$-time, $O(n \log n / \log \log n)$-space algorithm for the three-dimensional layers-of-maxima problem. Before we present our algorithm, however, we briefly outline how it relates to the prior algorithm claimed to run in $O(n \log n)$ time [2].

1.1. *Relation to the Prior Claim.* The previous algorithm [2] was based on the use of two new data structures. The first was a dynamic extension of the point-location structure of Preparata [14] to work in the context of staircase subdivisions, and the second was an extension of the biased search tree data structure of Bent et al. [3] to support finger searches and updates. The second structure was used as an auxiliary structure in the first, and both were analyzed by detailed case analyses. Unfortunately, there are crucial cases omitted in the original analyses of both of these structures, and filling in the details for these omitted cases appears to require increasing the running time to negate the claimed time bound of $O(n \log n)$ for the three-dimensional layers-of-maxima problem.

Our solution to the three-dimensional layers-of-maxima problem is also based on a dynamic data structure for staircase subdivisions. However, our structure is not an extension of Preparata's point-location approach, nor is it based on any biased data structures. Instead, our data structure exploits a new extension of dynamic fractional cascading, which may be of independent interest.

We sketch the basic approach for our algorithm in Section 2, which follows the space-sweeping framework that provided the basis for prior work [1], [2]. In Section 3 we present our new data structure to solve a special case of dynamic planar point location in a staircase subdivision, which is the key to the basic algorithm. In Section 4 we present an extension of dynamic fractional cascading that allows our data structure to achieve $O(\log n)$ amortized time per point in S, yielding an $O(n \log n)$-time algorithm for three-dimensional layers of maxima.

2. A Three-Dimensional Sweep Framework. Assume $S \subset \Re^3$. We use a three-dimensional sweep algorithm to solve the layers-of-maxima problem on S. Denote by $z(p)$ the z-coordinate of point p. Similarly define $x(p)$ and $y(p)$. Define $S_i(\ell) = \{p \in S : z(p) > i \wedge \mathit{layer}(p) = \ell\}$; $S_{-\infty}(\cdot)$ thus partitions S by layer, and the problem is to compute this partition. We process the points in S in order by decreasing z-coordinate, breaking ties arbitrarily.

INVARIANT 2.1. *When processing a point with z-coordinate i, points in $\bigcup_\ell S_i(\ell)$ are correctly labeled with their layers.*

For each layer, we maintain a subset of the points so far assigned to that layer. Define the *dominance region*, $D(p)$, of a point p to be the set of all points in $\Re^d$ that are dominated by p; for a set X of points, $D(X) = \bigcup_{p \in X} D(p)$. For a point $p \in \Re^3$, let $\pi(p)$ be the projection of p onto the (x, y) plane; for a set X of points, $\pi(X) = \bigcup_{p \in X} \{\pi(p)\}$.

In two dimensions the dominance region of a set of points is bounded by a *staircase*, which can be identified with its extremal points; see Figure 1(a). Denote by $M_i(\ell)$ the extremal points of $D(\pi(S_i(\ell)))$. That is, of the points so far assigned to each layer ℓ, $M_i(\ell)$ is the set of extremal points of the staircase induced by the dominance region of

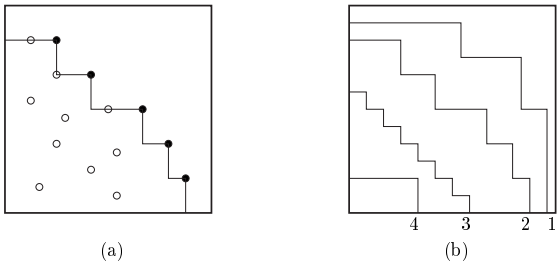

Fig. 1. In this and succeeding pictures, the x-dimension increases left-to-right, and the y-dimension increases bottom-to-top. (a) A set of points in the plane and the staircase bounding their dominance region. Extremal points are shown filled. (b) A staircase subdivision of the plane. Staircases are labeled by their corresponding layers.

their two-dimensional projections. We maintain $M_i(\ell)$ for each layer ℓ so far assigned a point, thereby dividing the two-dimensional plane into staircase regions; see Figure 1(b). We identify layers with their staircases.

Assume Invariant 2.1 is true before the first point with z-coordinate i is processed. We process the points $P_i = \{p \in S : z(p) = i\}$ as follows. First, we calculate $layer(p)$ for each $p \in P_i$. To do so, we identify where $\pi(p)$ lies in the staircase subdivision defined by $M_i(\cdot)$:

1. If $\pi(p)$ lies on or above staircase 1, then assign $layer(p) = 1$.
2. Otherwise, if $\pi(p)$ lies below the highest-numbered staircase, say s, then p is the first point assigned to $layer(p) = s + 1$.
3. Otherwise, $\pi(p)$ lies between two staircases, ℓ and $\ell + 1$, possibly lying on staircase $\ell + 1$; in this case, assign $layer(p) = \ell + 1$.

(See Figure 2(a).) Assume $M_i(\cdot)$ is correctly maintained. In the first case no previously processed point dominates p; in the other two cases, p is dominated by at least one point in $layer(p) - 1$, and p is not dominated by any point in layer $layer(p)$ or higher. Finally, no point q processed after p dominates p, because for each such q we know that $z(q) \leq z(p)$. Thus, the layer assignment maintains Invariant 2.1. We need only show

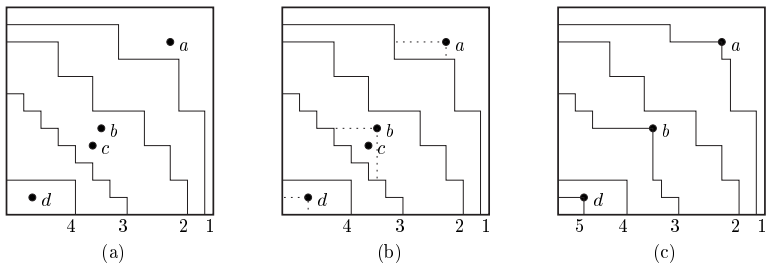

Fig. 2. (a) New points a, b, c, and d are assigned layers: $layer(a) = 1$; $layer(b) = layer(c) = 3$; $layer(d) = 5$. (b) Dotted lines show how a and b affect their staircases and d affects the plane. (c) After updating the staircases; c has no effect after b's insertion into staircase 3.

how to represent $M_i(\cdot)$ to allow testing of whether a point lies on or above some staircase and how to update $M_i(\cdot)$ to account for the points in P_i.

It suffices to maintain each $M_i(\ell)$ as a list of points ordered by decreasing y-coordinate (and hence increasing x-coordinate), including the sentinel points $(-\infty, \infty)$ and $(\infty, -\infty)$. This representation allows easy determination of whether point $\pi(p)$ lies on or above the staircase ℓ. Let $u, v \in M_i(\ell)$ such that $y(u) > y(p) \geq y(v)$.

1. $\pi(p)$ lies on staircase ℓ if (a) $x(p) = x(u)$ or (b) $x(u) < x(p) \leq x(v)$ and $y(p) = y(v)$;
2. otherwise, $\pi(p)$ lies above staircase ℓ if $x(p) > x(u)$;
3. otherwise, $\pi(p)$ lies below staircase ℓ.

Having determined layers for all points in P_i, we update $M_i(\cdot)$ as follows. In any order, consider each $p \in P_i$ in turn; let $\ell = \mathit{layer}(p)$. If $\pi(p)$ lies on or below staircase ℓ, no further action for p is necessary; $\pi(p)$ can lie below staircase ℓ at this point due to the addition of some other $q \in P_i$ that was previously processed into $M_i(\ell)$. Otherwise, $\pi(p)$ still lies above staircase ℓ and so becomes an extremal point defining the new staircase. In this case remove all points, if any, from $M_i(\ell)$ that are no longer extremal staircase points. Then insert $\pi(p)$ into its proper place in $M_i(\ell)$. Thus, $M_i(\cdot)$ becomes $M_j(\cdot)$ for the next z-coordinate j processed by the sweep algorithm.

To find the points in $M_i(\ell)$ that must be removed by the addition of $\pi(p)$, find $u, v \in M_i(\ell)$ such that $y(u) > y(p) \geq y(v)$. If $x(p) \geq x(v)$, then v is no longer extremal, so remove it from $M_i(\ell)$, and iterate at the successor of v in $M_i(\ell)$. If $x(p) < x(v)$, stop: v and its successors are still extremal. At the termination of this iteration, insert $\pi(p)$ as the successor of u in $M_i(\ell)$. See Figure 2(b), (c).

2.1. *Analysis.* Let $Q(N)$ be the time to determine the layer of a point. Let $R(N)$ be the time to determine the points u, v in the corresponding staircase that isolate where a new point begins to affect extremal points in that staircase. Let $I(N)$ and $D(N)$ be the times to insert and delete (resp.) each extremal point into/from a staircase. Each point p is the subject of one layer query and engenders at most one staircase transformation, to the staircase defining the dominance region of $\mathit{layer}(p)$. While the number of points deleted during this transformation can be large, over the course of the entire procedure each point is inserted and deleted at most once into/from its layer's staircase. The total time is thus $O(n(Q(N) + R(N) + I(N) + D(N)))$.

If we implement each $M_i(\ell)$ as a simple ordered list, then $Q(n) = O(\log^2 n)$: each query to determine where a point p lies with respect to a staircase takes $O(\log n)$ time, and we use binary search on the staircases to determine $\mathit{layer}(p)$. $R(n) = I(n) = D(n) = O(\log n)$, so the algorithm runs in $O(n \log^2 n)$ total time and $O(n)$ space.

We can improve the running time to $O(n \log n \log\log n)$ by using van Emde Boas trees [16] instead of ordered lists, but the space becomes $O(n^2)$. We can reduce the space back to $O(n)$ with the same $O(n \log n \log\log n)$ time bound using dynamic fractional cascading [13]. We omit the details of this implementation. Instead, we devise a more sophisticated representation for $M_i(\cdot)$, which itself uses an extension of dynamic fractional cascading, and which reduces the running time to $O(n \log n)$ but uses space $O(n \log n / \log\log n)$.

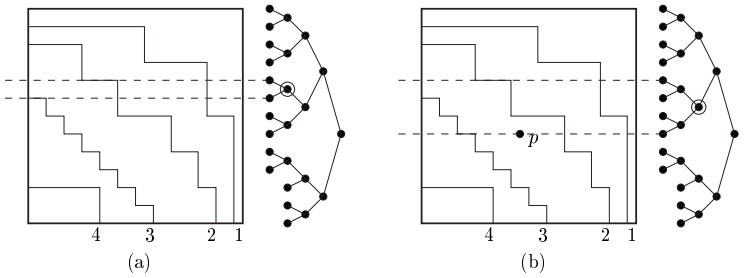

Fig. 3. A staircase subdivision with an associated search tree on the y-coordinates. Each of (a) and (b) depicts a slice (between the dashed lines) corresponding to the circled internal node. (a) Staircase 2 spans the slice, but staircase 1 does not. (b) Point p stabs staircase 2 in the slice but stabs none of the other staircases in that slice.

3. Improved Implementation. Since identifying the staircases that determine the layer of a new point is the time-consuming part in Section 2, we build a data structure to facilitate that query.

Let T be a search tree on the set of y-coordinates in S, i.e., each leaf an element of $\{y(p) : p \in S\}$. T is built a priori and remains unchanged throughout the algorithm. Each internal node u of T spans a *slice* of the y-coordinate space defined by the closed range $[\min(u), \max(u)]$, where $\min(u)$ (resp., $\max(u)$) is the minimum (resp., maximum) y-coordinate stored at a leaf descendant of u. Staircases pass through these slices; we say a staircase *enters* a slice at the top (maximum y-coordinate in the slice) and *exits* at the bottom (minimum y-coordinate in the slice). We say a staircase *spans* a slice if it enters and exits at different x-coordinates; staircases that simply pass vertically through a slice do not span that slice. See Figure 3(a). With each internal node u, we store the entry and exit x-coordinates of the staircases that span u's slice, associating with each such record the index of the corresponding staircase.

We say a point p *stabs* a staircase in a slice if an infinite vertical line drawn through p stabs a horizontal section of that staircase inside that slice. See Figure 3(b). We show how by testing a specific set of slices for stabbed staircases, we can identify the layer assigned to a new point. For some slice u and x-coordinate x let $pred_{entry(u)}(x)$ (resp., $pred_{exit(u)}(x)$) be the predecessor of x among u's entry (resp., exit) x-coordinates; let $succ_{exit(u)}(x)$ be the successor of x among u's exit x-coordinates; and let $lab(\cdot)$ be the index of the staircase with the corresponding entry/exit coordinates.

FACT 3.1. *Point p stabs a staircase in a slice if and only if the entry and exit points of the staircase fall on opposite sides of $x(p)$.*

LEMMA 3.2. *Assume p stabs a staircase in some slice u. The highest staircase stabbed by p in u, i.e., that with the lowest label, is staircase $lab(pred_{entry(u)}(x(p)))$; the lowest staircase stabbed by p in u, i.e., that with the highest label, is staircase $lab(succ_{exit(u)}(x(p)))$.*

PROOF. This follows from Fact 3.1 and that staircases do not cross. See Figure 4. □

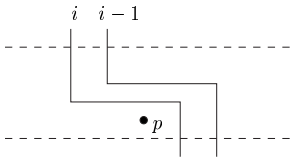

Fig. 4. A slice in which point p stabs staircases i and $i-1$. Because staircases do not cross, it must be that $i-1 = lab(pred_{entry(u)}(x(p)))$, or else p stabs a staircase higher than $i-1$ in the slice. Symmetrically, $i = lab(succ_{exit(u)}(x(p)))$, or else p stabs a staircase lower than i in the slice.

LEMMA 3.3. *Point p stabs a staircase in some slice u if and only if*

$$lab(pred_{entry(u)}(x(p))) \leq lab(succ_{exit(u)}(x(p))).$$

PROOF. The forward direction follows from Lemma 3.2. To see the reverse direction, assume $lab(pred_{entry(u)}(x(p))) = i \leq j = lab(succ_{exit(u)}(x(p)))$. Since staircases do not cross, the entry and exit points of both layers i and j must fall on opposite sides of $x(p)$; apply Fact 3.1. □

Lemma 3.3 implies that we can determine if a point stabs a staircase in a given slice in the time it takes to query some dictionary data structure. We use this lemma to determine $layer(p)$ for a new point p as follows (see Figure 5:)

1. Traverse the leaf-to-root path from $y(p)$ in T.
2. Find the nearest ancestor u whose slice contains a staircase stabbed by p. If $u = y(p)$, stop: p lies on a horizontal segment of a staircase. Otherwise, go to step (3).
3. Determine the nearest child of u whose slice contains a staircase stabbed by p. There must be some child as such, and it is not an ancestor of $y(p)$. Formally, let $c_1, \ldots, c_i$ be the children of u in top-down order (such that $\min(c_j) > \max(c_{j+1})$). Let c_j be the child of u that is an ancestor of $y(p)$. Find c_a such that p stabs a staircase in slice c_a and either (1) $a < j$ and there exists no b such that $a < b < j$ and p stabs a staircase in slice c_b; or (2) $a > j$ and there exists no b such that $a > b > j$ and p stabs a staircase in slice c_b.

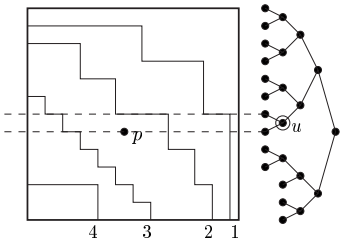

Fig. 5. The nearest ancestor, u, of leaf $y(p)$ whose slice contains a staircase stabbed by p is circled. The (one-dimensional) slice at the top child of u contains the stabbed staircase.

LEMMA 3.4. *If the procedure stops in step* (2), *then* $layer(p) = lab(pred_{entry(p)}(x(p)))$. *Otherwise, the procedure stops in step* (3). *In this case if* $a > j$, *then* $layer(p) = lab(pred_{entry(c_a)}(x(p)))$; *if* $a < j$, *then* $layer(p) = lab(succ_{exit(c_a)}(x(p))) + 1$.

PROOF. If (and only if) the procedure stops in step (2), then p lies on a horizontal segment of a staircase, which is identified by $pred_{entry(p)}(x(p))$.

Otherwise, if $a > j$, then c_a is below c_j. By assumption, p stabs a staircase in slice c_a. By Lemma 3.2, the highest such staircase is identified by $pred_{entry(c_a)}(x(p))$. Since p does not stab any staircase before stabbing this one, p lies between it and the next higher one, and so $layer(p) = lab(pred_{entry(c_a)}(x(p)))$.

If $a < j$, then c_a is above c_j. Arguing symmetrically, p must stab the staircase that is identified by $succ_{exit(c_a)}(x(p))$. Since p does not stab any staircase before stabbing this one, p lies between it and the next lower one, and so $layer(p) = lab(succ_{exit(c_a)}(x(p))) + 1$. □

3.1. *Updating* T. Recall the sweep algorithm framework from Section 2. Once the points in P_i have been assigned their layers, we must update the staircase extremal points. Let p be a new point assigned to layer ℓ. In addition to the new data structure in T, we still maintain each $M_i(\ell)$ as an ordered list to facilitate finding the points $u, v \in M_i(\ell)$ such that $y(u) > y(p) \geq y(v)$, indicating where $\pi(p)$ begins to affect the staircase. Assume $\pi(p)$ still lies above staircase ℓ, or else p engenders no further update.

While T itself remains static though this process, the sets of staircases spanning various slices change to reflect the addition of $\pi(p)$ to staircase ℓ. As in Section 2, iterate to find each v that is no longer extremal by the addition of $\pi(p)$ to staircase ℓ. Let v' be the first v encountered that remains in $M_i(\ell)$; i.e., after deleting the obviated v's and inserting $\pi(p)$, v' is $\pi(p)$'s successor in $M_i(\ell)$. The semi-closed range $[y(p), y(v'))$ (top-down) defines the extent of p's effect on $M_i(\ell)$, which we call p's *range of influence*. (In Figure 2(b) the dotted lines identify the ranges of influence for points a, b, and d.)

We must now update the slices at ancestors of $y(v)$ in T for each v deleted from $M_i(\ell)$. For each, traverse the $y(v)$-to-root path. For the slice at each node u on the path, there are four cases, depending on the relative positions of $y(p)$ and $\max(u)$ (the top of the slice) and those of $y(v')$ and $\min(u)$ (the bottom of the slice) (see Figure 6:)

1. $\max(u) > y(p), \min(u) \leq y(v')$. Both the top and bottom of the slices are outside p's range of influence. This and further ancestor slices are not affected by p, so terminate the traversal.
2. $\max(u) \leq y(p), \min(u) \leq y(v')$. The top of the slice is inside p's range of influence, but the bottom is outside. The entry point of staircase ℓ, which in this case is given by $pred_{entry(u)}(x(p))$, changes to $x(p)$, so delete the old entry point and insert the new one.
3. $\max(u) > y(p)$, $\min(u) > y(v')$. The bottom of the slice is inside p's range of influence, but the top is outside. The exit point of staircase ℓ, which in this case is given by $pred_{exit(u)}(x(p))$, changes to $x(p)$, so delete the old exit point and insert the new one.
4. $\max(u) \leq y(p)$, $\min(u) > y(v')$. Both the top and bottom of the slice are inside p's range of influence. Staircase ℓ no longer spans this slice. If it did before,

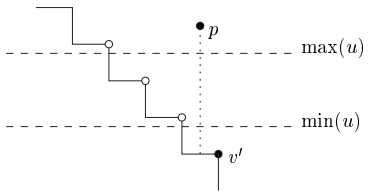

Fig. 6. A staircase, ℓ, being updated by the insertion of p. The dotted line identifies the range of p's influence: $[y(p), y(v'))$. The open circles are extremal points that must be removed from the staircase (and hence from which leaf-to-root traversals are applied). By the assumption that p belongs to layer ℓ, if the top of some slice u (indicated by dashed lines) is inside p's range of influence, then the entry point for staircase ℓ in that slice is given by $pred_{entry(u)}(x(p))$; otherwise, some other staircase would intercede between staircase ℓ and p. Similarly, if the bottom of the slice is inside p's range of influence, then the exit point for staircase ℓ in that slice is given by $pred_{exit(u)}(x(p))$.

$pred_{entry(u)}(x(p))$ and $pred_{exit(u)}(x(p))$ are both labeled ℓ and give the old entry and exit points, which are deleted. If $pred_{entry(u)}(x(p))$ and $pred_{exit(u)}(x(p))$ are not labeled ℓ, then staircase ℓ did not span this slice before, and no update is required.

Finally, perform the same operations on the slices encountered during a $y(p)$-to-root traversal of T. Correctness of this procedure follows from the fact that whenever a staircase spans a slice, there exists at least one extremal point of the staircase within the slice. That extremal point, having been the subject of such an update, caused entry and exit points to be recorded appropriately. Note that any slice whose top or bottom (or both) falls within p's range of influence is an ancestor of $y(p)$ or some $y(v)$ that was deleted above and hence gets updated appropriately.

3.2. *Analysis*. Straightforward implementations of T do not improve upon the results of Section 2. For example, if we construct T as a balanced, binary search tree and use red–black trees [8] to implement the entry and exit lists, then $Q(N) = O(\log^2 N)$ as before: $O(\log n)$ time at each level of T to perform a stabbing test using Lemma 3.3. $R(N) = O(\log n)$, and $I(N) = D(N) = O(\log^2 N)$ (argue as with $Q(N)$). Each point in S yields an element in at most one entry and exit list in each level of T. The algorithm thus runs in $O(n \log^2 n)$ time and $O(n \log n)$ space.

Using van Emde Boas trees [16] for the entry and exit lists reduces the running time to $O(n \log n \log\log n)$, but as each slice's entry and exit list requires $O(n)$ space, the total space increases to $O(n^2)$. Using Willard's q-fast tries [17] yields time $O(n \log^{1.5} n)$ and space $O(n \log n)$.

In general, however, if we set the arity of T to some d and use a dictionary that on N items admits predecessor and successor queries, insertions, and deletions in time $\mathcal{D}(N)$, then $Q(N) = O(\mathcal{D}(N)(\log_d N + d))$: $\mathcal{D}(N) \log_d N$ to find the first ancestor slice that contains a stabbed staircase, and $d \cdot \mathcal{D}(N)$ to find the nearest child slice containing the stabbed staircase. $R(N) = O(\log N)$ as before, and $I(N) = D(N) = O(\mathcal{D}(N) \log_d N)$. If $d = O(\sqrt{\log n})$ and $\mathcal{D}(N) = O(\log\log n)$, this would yield an $O(n \log n)$-time algorithm. Again we could use van Emde Boas trees [16], but the space would still be $O(n^2)$. Using Mehlhorn and Näher's [12] modification reduces the space to $O(n)$, but the time bound becomes randomized.

In the next section we show how to apply dynamic fractional cascading with this method to achieve $O(n \log n)$ time (worst case) and $O(n \log n / \log \log n)$ space.

4. Dynamic Fractional Cascading. Let $G = (V, E)$ be a *control graph* with a *catalog* $C(v) \subset \Re^+ = \Re \cup \{-\infty, \infty\}$ associated with each vertex $v \in V$ and a *range* $R(u, v) = R(v, u)$, which is an interval on the open real line, associated with each edge $\{u, v\} \in E$. The *local degree*, $d(v)$, of a vertex v is the maximum number of incident edges whose ranges contain any given element of $\Re^+$. That is, if $N_x(v) = \{u : \{v, u\} \in E \wedge x \in R(v, u)\}$, then $d(v) = \max\{|N_x(v)| : x \in \Re^+\}$. G has *locally bounded degree* d if $d \geq \max\{d(v) : v \in V\}$. Let $N = |V| + |E| + \sum_{v \in V} |C(v)|$.

A *generalized path* in G is a sequence $(v_1, \ldots, v_p)$ of vertices such that for each $1 < j \leq p$, there exists an edge $\{v_i, v_j\} \in E$ for some $1 \leq i < j$. Chazelle and Guibas [5] consider the problem of traversing a generalized path of G, at each vertex v determining the predecessor in $C(v)$ (or at each finding the successor) of a query value $x \in \Re^+$, given that $x \in R(e)$ for each edge e induced by the path. They design the *fractional cascading* data structure, which requires $O(N)$ space and answers each such query in time $O(p \log d + \log N)$ time, where p is the number of vertices in the path and G is of locally bounded degree d. They show how insertions and deletions from individual catalogs can be performed in $O(\log N)$ amortized time each, but then the query time degrades to $O(p \log d \log \log N + \log N)$.

Mehlhorn and Näher [13] show how to accommodate each update in $O(\log \log N)$ amortized time if the position of the update is known, i.e., given a pointer to the successor (or predecessor) of the element to be inserted or deleted. The query time becomes $O(p \log \log N + \log N)$. Their results assume that $d = O(1)$. Raman [15] notes that this result is easily extended to yield $O(p(\log d + \log \log N) + \log N)$ query time in the case of arbitrary d and with Dietz goes on [6], [15] to make the update time worst case $O(\log d + \log \log N)$.

The above extension of dynamic fractional cascading to arbitrary d uses an idea of Chazelle and Guibas that replaces each vertex of G by a uniform *star tree*. Using a more local technique, we can show the following slightly more general result. Let $(v_1, \ldots, v_p)$ denote a generalized path. Let $(\{u_2, v_2\}, \ldots, \{u_p, v_p\})$ be any sequence of edges that can be used to traverse the path; i.e., for each $1 < j \leq p$, $\{u_j, v_j\} \in E$ and there is some $1 \leq i < j$ such that $u_j = v_i$. Redefine $d(v)$ to be the (real) degree of v.

THEOREM 4.1. *A dynamic fractional cascading data structure can be built that uses $O(N)$ space, performs queries in time $O(p \log \log N + \log N + \sum_{i=2}^{p} (\log d(u_i) + \log d(v_i)))$, and accommodates insertions and deletions in $O(\log \log N)$ amortized time each, given a pointer to the predecessor or successor of the item to be inserted or deleted.*

Note that this allows a (constant) few vertices in the traversal of the path to be of arbitrarily high degree while still maintaining the overall time bound in Mehlhorn and Näher's original result.

PROOF OF THEOREM 4.1. Construct a new graph $G' = (V', E')$ as follows. For each $v \in V$, build a balanced binary tree T_v on $d(v)$ leaves. Let $\eta_v(i)$ be the ith neighbor

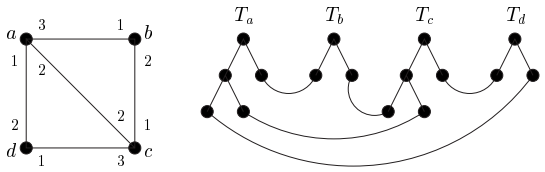

Fig. 7. A control graph G (left) and the associated new graph G' (right). Numbers on edges of G indicate an ordering of the neighbors of each vertex. Leaves of subtrees in G' are implicitly numbered left to right.

of v in G for some arbitrary but fixed ordering of neighbors; and let v_i denote the ith leaf of T_v in symmetric order. G' is comprised of all the trees T_v plus an edge $\{u_i, v_j\}$, connecting the appropriate leaves of T_u and T_v, for each edge $\{u, v\} \in E$ such that $\eta_u(i) = v$ and $\eta_v(j) = u$. (See Figure 7.) The degree of each vertex in G' is thus two. We maintain a dynamic fractional cascading data structure on G'. Each catalog $C(v)$ is stored at the root of the corresponding T_v. Since $|V'| \leq 4|E|$ and $|E'| \leq 5|E|$, $N' = |V'| + |E'| + \sum_{v \in V} |C(v)| = \Theta(N)$.

Now map paths in G to paths in G'. To start a query at $v \in V$, start at the root of T_v in V'. To traverse edge $\{u, v\}$ in G, traverse in G' the path from the root of T_u to that of T_v, via the appropriate leaves. (Record with each edge $\{u, v\} \in E$ the indices of the appropriate leaves in T_u and T_v.)

Mehlhorn and Näher [13, Lemma 2] show that traversing the new path takes $O(\ell)$ time, where $\ell = \log d(u) + \log d(v)$ is the length of the new path, while querying the catalogs $C(u)$ and $C(v)$ can still be done in $O(\log \log N)$ time each [13, Lemma 3], as can insertion and deletion (in $O(\log \log N)$ amortized time) into each catalog. □

It remains to combine this result with Raman's worst-case update technique.

4.1. *Application to Layers of Maxima*

THEOREM 4.2. *The three-dimensional layers-of-maxima problem can be solved in $O(n \log n)$ time and $O(n \log n / \log \log n)$ space.*

PROOF. We use the algorithm framework of Section 3. Let T be a static, balanced search tree of arity $\sqrt{\log n}$. (At most one internal child of every node has degree less than $\sqrt{\log n}$; the others have degree $\sqrt{\log n}$.) The depth of T is thus $O(\log n / \log \log n)$. The leaves of T correspond to the y-coordinates of points in S.

Now modify T as follows. Transform each internal node u into a length-3 path (u_1, u_2, u_3); let the parent of u_1 correspond to the parent of u, which will be similarly modified; and let the children of u_3 correspond to the children of u, which will be similarly modified unless they were leaves. That is, if v is the parent of u, v_3 becomes the parent of u_1. For each pair (u, v) of consecutive children in the original T, doubly link the corresponding nodes u_2 (u if it is a leaf) and v_2 (v if it is a leaf). See Figure 8. While T is no longer a tree, the notion of levels of T extends easily; in particular, the leaf-to-root paths maintain length $O(\log n / \log \log n)$.

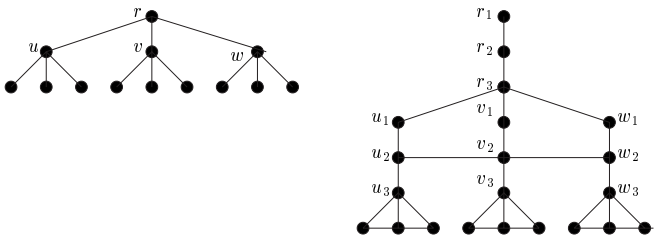

Fig. 8. A 3-ary tree (left) and the derived control graph (right). Expanding the internal nodes facilitates traversing the children of each node in the original tree.

Apply dynamic fractional cascading as per Theorem 4.1 with T as the control graph; for each original node u, store catalogs at the new node u_2 (or u if it is a leaf) to record slice u's entry and exit lists. $Q(N) = O(\log n)$: $O(\log \log n)$ time at each level of T to perform a stabbing test using Lemma 3.3; $O(\sqrt{\log n} \log \log n)$ time to find the appropriate stabbed child of the first stabbed ancestor, using the children links; and $O(\log \log n)$ time per level of T to traverse edges incident to nodes of degree $O(\sqrt{\log n})$. The transformation of the original T allows traversing the children of a node in $O(\sqrt{\log n} \log \log n)$ time, since they are linked at constant-degree nodes. $R(N) = O(\log n)$ if we continue to maintain ordered lists of extremal points for each staircase. $I(N) = D(N) = O(\log n)$: each update due to a point p is identified by the predecessor of $x(p)$ in the entry and exit lists of slices encountered in a leaf-to-root traversal of T, which we maintain while traversing the query path in T, so we can update the fractional cascading structure at a cost of $O(\log \log n)$ per level of T. The overall time is thus $O(n \log n)$.

Each point in S engenders an element in at most one entry and exit list in each level of T. The space is thus $O(n \log n / \log \log n)$. □

5. Conclusion. We have provided an $O(n \log n)$-time, $O(n \log n / \log \log n)$-space algorithm to solve the three-dimensional layers-of-maxima problem. Our algorithm uses a three-dimensional sweep on the input point set, maintaining a staircase subdivision of the plane with an improved dynamic fractional cascading data structure that accommodates vertices of high degree in the control graph.

While $\Omega(n \log n)$ is a lower bound on the time to solve this problem in the comparison model, it remains open to achieve that bound using linear or at least $o(n \log n / \log \log n)$ space as well as to provide bounds for higher dimensions. The static nature of the plane-sweep approach to solving this problem makes bootstrapping our solution for higher dimensions seem problematic.

Our algorithm framework employs a special case of dynamic point location in a staircase subdivision: namely, when all the y-coordinates are known a priori. It remains open to solve this problem in general. It is also open whether there are other applications of the dynamic fractional cascading extension.

Acknowledgement. We thank Rajeev Raman for helping set our dynamic fractional cascading extension in context with prior work.

References

[1] P. K. Agarwal. Personal communication, 1992.

[2] M. J. Atallah, M. T. Goodrich, and K. Ramaiyer. Biased finger trees and three-dimensional layers of maxima. In *Proc*. 10*th ACM Symp*. *on Computational Geometry*, pages 150–159, 1994.

[3] S. W. Bent, D. D. Sleator, and R. E. Tarjan. Biased search trees. *SIAM Journal on Computing*, 14(3):545–568, 1985.

[4] B. Chazelle. On the convex layers of a planar set. *IEEE Transactions on Information Theory*, IT-31:509–517, 1985.

[5] B. Chazelle and L. J. Guibas. Fractional cascading: I. A data structure technique. *Algorithmica*, 1(2)133–162, 1986.

[6] P. F. Dietz and R. Raman. Persistence, amortization and randomization. In *Proc*. 2*nd ACM-SIAM Symp*. *on Discrete Algorithms*, pages 78–88, 1991.

[7] P. G. Franciosa, C. Gaibisso and M. Talamo. An optimal algorithm for the maxima set problem for data in motion. In *Abstracts* 8*th European Workshop on Computational Geometry*, pages 17–21, 1992.

[8] L. J. Guibas and R. Sedgewick. A dichromatic framework for balanced trees. In *Proc*. 19*th IEEE Symp*. *on Foundations of Computer Science*, pages 8-21, 1978.

[9] J.-W. Jang, M. Nigam, V. K. Prasanna, and S. Sahni. Constant time algorithms for computational geometry on the reconfigurable mesh. *IEEE Transactions on Parallel and Distributed Systems*, 8(1):1–12, 1997.

[10] S. Kapoor. Dynamic maintenance of maximas of 2-d Point Sets. In *Proc*. 10*th ACM Symp*. *on Computational Geometry*, pages 140–149, 1994.

[11] H. T. Kung, F. Luccio, and P. Preparata. On finding the maxima of a set of vector. *Journal of the ACM*, 22(4):469-476, 1975.

[12] K. Mehlhorn and S. Näher. Bounded ordered dictionaries in $O(\log\log N)$ time and $O(n)$ space. *Information Processing Letters*, 35(4):183–189, 1990.

[13] K. Mehlhorn and S. Näher. Dynamic fractional cascading. *Algorithmica* 5(2):215–241, 1990.

[14] F. P. Preparata. A new approach to planar point location. *SIAM Journal on Computing*, 10(3):473-482, 1981.

[15] R. Raman. Eliminating Amortization: On Data Structures with Guaranteed Fast Response Time. Ph.D. thesis, University of Rochester, 1992.

[16] P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6(3):80–82, June 1977.

[17] D. E. Willard. New trie data structures which support very fast search operations. *Journal of Computer and System Sciences*, 28(3):379–394, 1984.