SIAM J. COMPUT.
Vol. 18, No. 4, pp. 811-830, August 1989

FULLY DYNAMIC POINT LOCATION IN A MONOTONE SUBDIVISION*

FRANCO P. PREPARATA† AND ROBERTO TAMASSIA‡

Abstract. In this paper a dynamic technique for locating a point in a monotone planar subdivision, whose current number of vertices is n, is presented. The (complete set of) update operations are insertion of a point on an edge and of a chain of edges between two vertices, and their reverse operations. The data structure uses space $O(n)$. The query time is $O(\log^2 n)$, the time for insertion/deletion of a point is $O(\log n)$, and the time for insertion/deletion of a chain with k edges is $O(\log^2 n+k)$, all worst-case. The technique is conceptually a special case of the chain method of Lee and Preparata and uses the same query algorithm. The emergence of full dynamic capabilities is afforded by a subtle choice of the chain set (separators), which induces a total order on the set of regions of the planar subdivision.

Key words. point location, planar subdivision, monotone polygon, dynamic data structures, on-line algorithm, computational geometry, analysis of algorithms

AMS(MOS) subject classifications. 68U05, 68Q25

1. Introduction.

A fundamental and classical problem in computational geometry, *planar point location*, is an instrument in a wide variety of applications. In a geometric context, it is naturally viewed as the extension to two dimensions of its one-dimensional analogue: search in a total ordering. It is formulated as follows. Given a planar subdivision **P** with n vertices (a planar graph embedded in the plane with straight-line edges), determine to which region of **P** a query point q belongs. The repetitive use of **P** and the on-line requirement on the answers call for a preprocessing of **P** that may ease the query operation, just as sorting and binary search intervene in one-dimensional search. The history of planar point location research spans more than a decade and is dense in results; the reader is referred to the extensive literature on this subject: [DL], [EKA], [EGS], [Ki], [LP], [LT], [P1], [P2], [PS], and [ST].

Most of the past research on the topic has focused on the *static* case of planar point location, where the planar subdivision **P** is fixed. For this instance of the problem, several practical techniques are available today (e.g., [EKA], [EGS], [LP], [P1], [ST]), some of which are provably optimal in the asymptotic sense [EGS], [ST]. The analogy with one-dimensional search, for which both static and dynamic optimal techniques have long been known, naturally motivates the desire to develop techniques for *dynamic* planar point location, where the planar subdivision can be modified by insertions and deletions of points and segments. In this setting, the three traditional measures of complexity for the static problem—preprocessing time, space for the data structure, and query time—are supplemented by measures of pertinent update times, and preprocessing time is no longer relevant if the data structure is dynamically built.

Work on dynamic point location is a rather recent undertaking. Overmars [O] proposed a technique for the case where the n vertices of **P** are given, the boundary of each region has a bounded number of edges, and only edges can be easily inserted or deleted. The basic entities used in Overmars's method are the edges themselves;

* Received by the editors January 20, 1988; accepted for publication (in revised form) November 23, 1988. This work was supported in part by National Science Foundation grant ECS-84-10902. A preliminary version of this paper was published in the Proceedings of the 29th IEEE Symposium on Foundations of Computer Science, pp. 558-567, 1988.

† Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801.

‡ Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801. Present address, Department of Computer Science, Brown University, Providence, Rhode Island 02912.

each edge currently in **P** is stored in a segment tree defined on the fixed set of vertex abscissae, and the edge fragments assigned to a given node of the segment tree form a totally ordered set and are therefore efficiently searchable. This approach yields $O(n \log n)$ space, and $O(\log^2 n)$ query and update times (worst-case). Note that these measures are unrelated to the current number of edges in **P**. This technique can be extended to support insertions and deletions of vertices in $O(\log^2 n)$ amortized time, at the expense of deploying a rather complicated and not very practical data structure.

Another interesting dynamic point location technique, allowing both edge and vertex updates, is presented by Fries, Mehlhorn, and Naeher [F], [FMN], [M]. Their approach achieves $O(n)$ space, $O(\log^2 n)$ query time, and $O(\log^4 n)$ amortized update time. If only insertions are considered, the update time is reduced to $O(\log^2 n)$ (amortized) [M, pp. 135-143]. This technique is suited for general subdivisions, and is based on the static point location algorithm of Lee and Preparata [LP].

In a recent paper [ST], Sarnak and Tarjan indicated as one of the most challenging problems in computational geometry the development of a fully dynamic point location data structure whose space and query time performance are of the same order as that of the best known static techniques for this problem. In this paper we make significant progress toward this goal, as expressed by the following theorem, which represents the central result of this paper.

THEOREM A. *Let* **P** *be a monotone planar subdivision with n vertices. There exists a dynamic point location data structure with* $O(n)$ *space requirement and* $O(\log^2 n)$ *query time that allows for insertion/deletion of a vertex in time* $O(\log n)$ *and insertion/deletion of a chain of k edges in time* $O(\log^2 n + k)$, *all time bounds being worst-case.*

It must be underscored that our method allows for arbitrary insertions and deletions of vertices and edges, the only condition being that monotonicity of the subdivision be preserved. The restriction to monotone subdivisions prevents the achievement of a more general goal. However, the class of monotone subdivisions is very important in applications, since it includes convex subdivisions, triangulations, and rectangular dissections. Moreover, from a methodological standpoint, the presented dynamic technique does not use bizarre data structures and is based on the same geometric objects, the separating chains, that yielded the first practical, albeit suboptimal, point location technique of Lee and Preparata [LP], and later the practical and optimal algorithm of Edelsbrunner, Guibas, and Stolfi [EGS].

The paper is organized as follows. In § 2 we review the technical background and precisely formulate the problem. We recall the static point location technique of Lee and Preparata that consists essentially of performing binary search on a set of monotone chains, called separators, sorted from left to right. Also, we introduce a complete repertory of update operations.

In § 3 we define a total ordering $<$ on the set of regions of **P** and define a subclass of monotone subdivisions, called regular subdivisions, for which the ordering $<$ is induced by the left-to-right adjacency of the regions. We show that the separators of a regular subdivision have a simple structure that allows for an efficient dynamic maintenance. We then transform an arbitrary subdivision **P** into a regular subdivision **P*** by (virtually) duplicating the edges along some monotone chains, called "channels." The regions of **P***, called "clusters," are sets of regions of **P** (consecutive in the order $<$) connected by channels. The formal underpinning of the order $<$ can be found in the theory of planar *st*-graphs [LEC], [TP] and planar lattices [KR].

In § 4, we present the dynamic technique. The data structure is based on organizing the separators of **P*** into a balanced tree, called "separator tree." We show that the insertion/deletion of a chain causes the ordering $<$ of the regions of **P** to be modified in a simple way. Namely, the transformation consists of partitioning the sorted sequence of regions into four subsequences and swapping the two middle ones. Regarding **P***, we show that only $O(1)$ channels are affected by the update. These properties are the basis of the algorithms for the insertion/deletion of chains. At the end of the section we describe the simpler algorithms for the insertion/deletion of vertices.

Finally, § 5 concludes the paper with some open problems.

2. Preliminaries. The geometric objects to be defined below are readily motivated if we view a point of the plane as the central projection of a point of a hemisphere to whose pole this plane is tangent.

A *vertex* v in the plane is either a finite point or a point at infinity (the latter is the projection of a point on the hemisphere equator). An *edge* $e=(u, v)$ is the portion of the straight line between u and v, with the only restriction that u and v be not points at infinity associated with the same direction. Thus e is either a segment or a straight-line ray, but not a whole straight-line. When both u and v are at infinity, then e is an edge at infinity, i.e., a portion of the line at infinity (the projection of an arc of the equator). A (*polygonal*) *chain* γ is a sequence $(e_i : e_i = (v_i, v_{i+1}),\ i=1, \cdots, p-1)$ of edges; it is *simple* if nonself-intersecting; it is *monotone* if any line parallel to the x-axis intersects γ in at most a point or a segment. The notions of "left" and "right" are referred to a bottom-up orientation of the involved entity (a chain, or, later on, a separator, an edge, etc.). A *simple polygon* r is a region of the plane delimited by a simple chain with $v_p = v_1$, called the *boundary* of r. Note that r could be unbounded; in this case the boundary of r contains one or more edges belonging to the line at infinity. A polygon r is monotone if its boundary is partitionable into two monotone chains γ_1 and γ_2, called the *left chain* and *right chain* of r, respectively (see Fig. 1). Chains γ_1 and γ_2 share two vertices referred to as $HIGH(r)$ and $LOW(r)$, respectively, with $y(HIGH(r)) > y(LOW(r))$. In other words, $HIGH(r)$ and $LOW(r)$ are, respectively, points of maximum and minimum ordinates in polygon r; each is unique unless it is the extreme of either a horizontal edge or an edge at infinity, in which case the selection between the two edge extremes is arbitrary.

A *monotone subdivision* **P** is a partition of the plane into monotone polygons called the *regions* of **P** (see Fig. 2(a)). It is easily realized (see also [EGS]) that a monotone subdivision of **P** is determined by a planar graph G embedded in the plane whose edges are either segments or rays of straight lines (referred to as a "planar

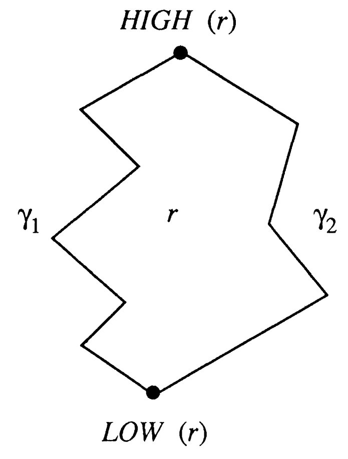


FIG. 1. *Nomenclature for a monotone polygon. Left chain*: γ_1; *right chain*: γ_2.

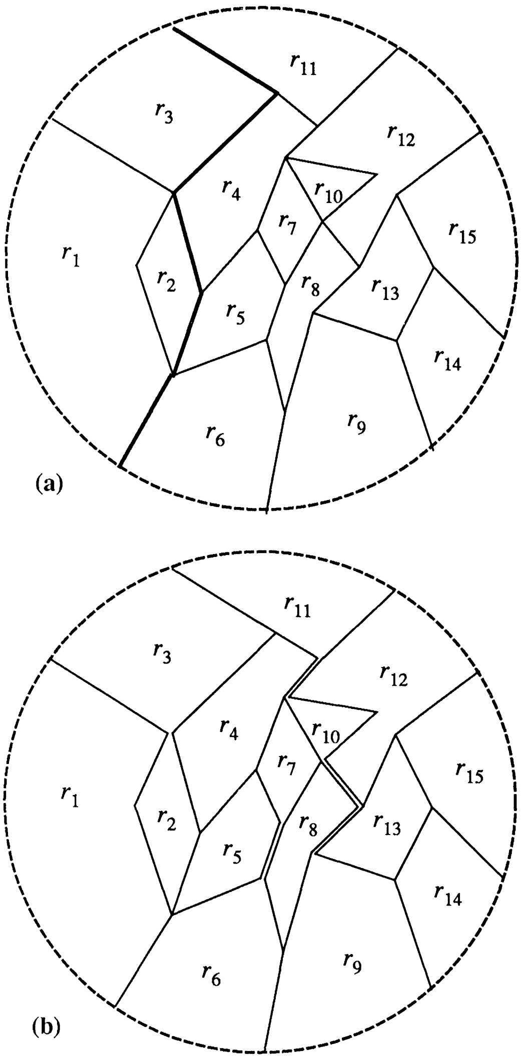


FIG. 2. (a) *A monotone subdivision* **P**. - - - *line at infinity*; ——— *separator.* (b) *The regular subdivision* **P*** *obtained from* **P** *by forming all maximal clusters.*

straight-line graph" in [LP]): edges, vertices, and chains of G are referred to as edges, vertices, and chains of **P**. The vertices of **P** are both the finite ones and those at infinity. This ensures the validity of the well-known Euler's formula and its corollaries:

$$|E| = O(|V|), \quad |R| = O(|V|),$$

where V, E, and R, respectively, are the set of vertices, edges, and regions of **P**.

Given a monotone subdivision **P**, a *separator* σ of **P** is a monotone chain $(v_1, \cdots, v_p)$ of **P** with the property that v_1 and v_p are points at infinity (hence, each horizontal line intersects a separator either in a point or in a segment). A separator of **P** is illustrated with bold line segments in Fig. 2(a). Given separators σ_1 and σ_2 of **P**, we say that σ_1 *is to the left of* σ_2, denoted $\sigma_1 < \sigma_2$, if, for any horizontal line l intersecting σ_1 and σ_2 in distinct points, l intersects σ_1 to the left of σ_2. A *partial subdivision* is the portion of a monotone subdivision contained between two distinct separators σ_1

and σ_2, with $\sigma_1 < \sigma_2$. A *complete family of separators* Σ for **P** is a sequence $(\sigma_1, \sigma_2, \cdots, \sigma_t)$ with $\sigma_1 < \sigma_2 < \cdots < \sigma_t$, such that every edge of **P** is contained in at least one separator of Σ. Notice that from Euler's formula $t = O(n)$. As shown in [LP], every monotone subdivision admits a complete family of separators.

Given a complete family of separators Σ for **P**, it is well known [LP] how to use Σ to perform planar point location in **P**. If n is the number of vertices of **P**, then in time $O(\log n)$ we can decide on which side of a separator the query point q lies; applying this operation as a primitive, a bisection search on Σ determines, in time $O(\log t \cdot \log n)$, two consecutive separators between which lies q. This process can be adapted or supplemented to determine the actual region r to which q belongs.

Since Σ is used in a binary search fashion, each separator is assigned to a node of a binary search tree, called the *separator tree* **T**. With a minor abuse of language, we call "node σ" the node of **T** to which σ has been assigned. An edge e of **P** belongs, in general, to a nonempty interval $(\sigma_i, \sigma_{i+1}, \cdots, \sigma_j)$ of separators. Let node σ_k, $i \leqq k \leqq j$ be the least common ancestor of nodes $\sigma_i, \sigma_{i+1}, \cdots, \sigma_j$; then e is called a *proper edge* of σ_k and is stored only once at node σ_k. We denote by $proper(\sigma_k)$ the set of proper edges of σ_k, i.e., the edges of σ_k stored at node σ_k. This yields $O(n)$ storage space while guaranteeing the correctness of the technique (see [EGS], [LP]). Note that edges whose extremes are both at infinity need not be stored.

We now illustrate that a planar subdivision **P** can be constructed by an appropriate sequence of the following operations.

INSERTPOINT $(v, e; e_1, e_2)$:
Split the edge $e = (u, w)$ into two edges $e_1 = (u, v)$ and $e_2 = (v, w)$ by inserting vertex v.

REMOVEPOINT $(v; e)$:
Let v be a vertex of degree 2 whose incident edges, $e_1 = (u, v)$ and $e_2 = (v, w)$, are on the same straight line. Remove v and replace e_1 and e_2 with edge $e = (u, w)$.

INSERTCHAIN $(\gamma, v_1, v_2, r; r_1, r_2)$:
Add the monotone chain $\gamma = (v_1, w_1, \cdots, w_{k-1}, v_2)$, with $y(v_1) \leqq y(v_2)$, inside region r, which is decomposed into regions r_1 and r_2, with r_1 and r_2, respectively, to the left and to the right of γ, directed from v_1 to v_2.

REMOVECHAIN $(\gamma; r)$:
Let γ be a monotone chain whose nonextreme vertices have degree 2. Remove γ and merge the regions r_1 and r_2 formerly on the two sides of γ into region r. (The operation is allowed only if the subdivision **P**$'$ so obtained is monotone.)

With the above repertory of operations, we claim that a monotone subdivision **P** can always be transformed into a monotone subdivision **P**$'$ having either fewer vertices or fewer edges. Then by $O(n)$ such transformations, we obtain the trivial subdivision whose only region is the entire plane (bounded by the line at infinity).

Indeed, let σ be a separator of **P**, and imagine traversing σ from $-\infty$ to $+\infty$. Let $\deg^-(v)$ and $\deg^+(v)$ denote the numbers of edges incident on v and lying in the halfplanes $y < y(v)$ and $y > y(v)$, respectively. (For simplicity, we assume here that no edge is horizontal. In the general case, a horizontal edge is conventionally directed from left to right.) Let $(v_1, \cdots, v_p)$ be the sequence of finite vertices of σ with $\deg^+(v_i) + \deg^-(v_i) \geqq 3$, as encountered in the traversal of σ. If this sequence is empty, the entire chain can be trivially removed. Therefore, assume $p \geqq 1$. If there are two consecutive vertices v_i and v_{i+1} such that $\deg^+(v_i) \geqq 2$ and $\deg^-(v_{i+1}) \geqq 2$, then the chain γ (of degree-2 vertices) between v_i and v_{i+1} can be deleted by *REMOVECHAIN* while preserving monotonicity. Suppose that there are no two such vertices. If

$\deg^-(v_1) \geqq 2$, then the portion of σ from $-\infty$ to v_1 can be deleted. Otherwise, $\deg^-(v_1) = 1$, and the preceding conditions give rise to the following chain of implications:

$$(\deg^-(v_1)=1) \Rightarrow (\deg^+(v_1) \geqq 2) \Rightarrow (\deg^+(v_2) \geqq 2) \Rightarrow \cdots \Rightarrow (\deg^+(v_p) \geqq 2);$$

the latter shows that the portion of σ from v_p to $+\infty$ can be deleted. This establishes our claim.

When all finite vertices have disappeared, the resulting subdivision consists of a closed chain of edges whose union is the line at infinity. Removal of the vertices at infinity completes the transformation. Since all of the above operations are reversible, this shows that any monotone subdivision **P** with n vertices can be constructed by $O(n)$ operations of the above repertory.

THEOREM 1. *An arbitrary planar subdivision* **P** *with n vertices can be assembled starting from the empty subdivision by a sequence of O(n) INSERTPOINT and INSERTCHAIN operations, and can be disassembled to obtain the empty subdivision by a sequence of O(n) REMOVEPOINT and REMOVECHAIN operations.*

Although the above operations are sufficient to assemble and disassemble any monotone subdivision, the following operation is also profitably included in the repertory.

MOVEPOINT $(v; x, y)$:

Translate a degree-2 vertex v from its present location to point (x, y). (The operation is allowed only if the subdivision **P**$'$ so obtained is monotone and topologically equivalent to **P**.)

3. Ordering the regions of a monotone subdivision. Let **P** be a monotone subdivision, and assume for simplicity that none of its finite edges is horizontal. Given two regions r_1 and r_2 of **P**, we say that r_1 is *left-adjacent to* r_2, denoted $r_1 \ll r_2$, if r_1 and r_2 share an edge e, and any separator of **P** containing e leaves r_1 to its left and r_2 to its right. Note that relation $\ll$ is trivially antisymmetric. But we can also show Lemma 1.

LEMMA 1. *Relation $\ll$ on the regions of* **P** *is acyclic.*

Proof. Assume $r_1 \ll r_2 \ll \cdots \ll r_k \ll r_1$. Let Σ be a complete family of separators and let $\{\sigma_1, \cdots, \sigma_{k-1}\} \subseteq \Sigma$ be such that σ_i separates r_i and r_{i+1}, so that $\sigma_1 < \sigma_2 < \cdots < \sigma_{k-1}$. If $\sigma \in \Sigma$ leaves r_k to its left and r_1 to its right, we have $\sigma_{k-1} < \sigma$ and $\sigma < \sigma_1$, a contradiction since the separators are ordered. □

Thus, the transitive closure of $\ll$ is a partial order, referred to as "*to the left of*," and denoted by $\rightarrow$. Specifically, $r_1 \rightarrow r_2$ if there is a path from r_1 to r_2 in the directed graph of the relation $\ll$ on the set of regions. Correspondingly, given two regions r_1 and r_2 of **P**, we say that r_1 *is below* r_2, denoted $r_1 \uparrow r_2$, if there is a monotone chain from $HIGH(r_1)$ to $LOW(r_2)$. Obviously, $\uparrow$ is a partial order on the set of regions. The following lemma shows that these two partial orders are complementary, and it is the geometric counterpart of a topological property of planar lattices [B, ex. 7(c), p. 32], [Ka], [KR].

LEMMA 2. *Let r_1 and r_2 be two regions of a monotone subdivision* **P**. *Then one and only one of the following holds*:

$$r_1 \rightarrow r_2, \quad r_2 \rightarrow r_1, \quad r_1 \uparrow r_2, \quad r_2 \uparrow r_1.$$

Proof. Let σ_L be the leftmost separator that contains the left chain of the boundary of r_1 and, analogously, let σ_R be the rightmost separator containing the right chain of the boundary of r_1. These separators partition **P** into five portions, each a partial subdivision: one of them is r_1 itself, and the others are denoted L, R, B, and T (see Fig. 3). Now, we consider four mutually exclusive cases for r_2, one of which must occur:

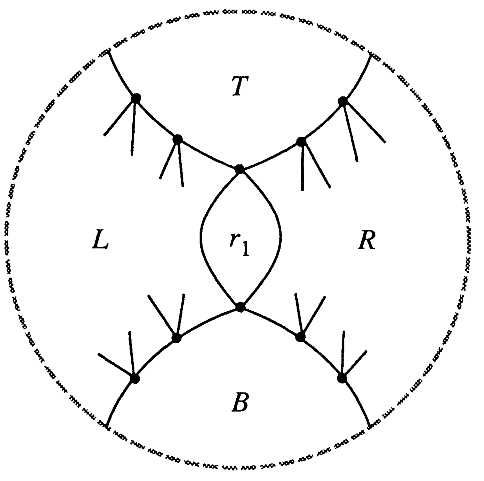


FIG. 3. **P** *partitioned into five portions, each a partial subdivision* (*for the proof of Lemma* 2).

(1) $r_2 \in L$. Consider any sequence of regions $(r'_1, \cdots, r'_s)$ such that $r_2 \to r'_1$, $r'_i \to r'_{i+1}$ for $i = 1, \cdots, s-1$, and the right chain of r'_s has a nonempty intersection with σ_L. If the right chain of r'_s has an edge that is also on the left boundary of r_1, then $r_2 \to r_1$. Otherwise, by the definition of σ_L, there is a sequence $r'_{s+1}, \cdots, r'_p$ of regions such that $r'_j \to r'_{j+1}$ for $j = s, \cdots, p-1$, and $r'_p \to r_1$. Thus, in all cases, $r_2 \to r_1$.

(2) $r_2 \in R$. Arguing as in (1), we establish $r_1 \to r_2$.

(3) $r_2 \in B$. Since $LOW(r_1)$ is the highest ordinate vertex in B, there is a monotone chain from any vertex in B to $LOW(r_1)$, and, in particular, from $HIGH(r_2)$ to $LOW(r_1)$. Thus $r_2 \uparrow r_1$.

(4) $r_2 \in T$. Arguing as in (3), we establish $r_1 \uparrow r_2$. □

We say that r_1 *precedes* r_2, denoted $r_1 < r_2$, if either $r_1 \to r_2$ or $r_1 \uparrow r_2$.

THEOREM 2. *The relation* $<$ *on the regions of* **P** *is a total order.*

As an example, the region subscripts in Fig. 2(a) reflect the order $<$.

DEFINITION 1. A *regular subdivision* is a monotone subdivision having no pair (r_1, r_2) of regions such that $r_1 \uparrow r_2$.

For example, in Fig. 2(a) we have $r_9 \uparrow r_{10}$, which shows that the illustrated monotone subdivision is not regular. An example of regular subdivision is given in Fig. 4(a).

The significance of regular subdivisions is expressed by Theorem 3 below. It is easily realized that there is a unique complete family $\Sigma = (\sigma_1, \cdots, \sigma_t)$ of separators for a regular subdivision **P**. By the definition of separator, all regions to the left of σ_j precede all those to its right in the order $<$. Let **T** be a separator tree for the above family Σ. Recalling the rule for storing the edges of separator σ in **T**, as reviewed in § 2, we have Theorem 3.

THEOREM 3. *In a regular subdivision* **P**, *the edges of proper*(σ) *in* **T** *form a single chain* (*see Fig.* 4(b)).

Proof. Assume for a contradiction that σ contains a chain γ that is the bottom-to-top concatenation of three nonempty chains γ_1, γ_2, and γ_3, where γ_1 and γ_3 consist of proper edges of σ, and γ_2 contains no such edge. Let v_1 and v_2 be the bottom and top vertices, respectively, of γ_2, and let $e' \in \gamma_1$ and $e'' \in \gamma_2$ be the edges of σ incident on v_1. Since $e'' \notin proper(\sigma)$, we must have $e'' \in proper(\sigma')$, where node σ' is an ancestor of node σ in **T**. We claim there is a region r_1 such that $v_1 = HIGH(r_1)$. Otherwise, each separator containing e''—and so σ'—also contains e', contrary to the hypothesis

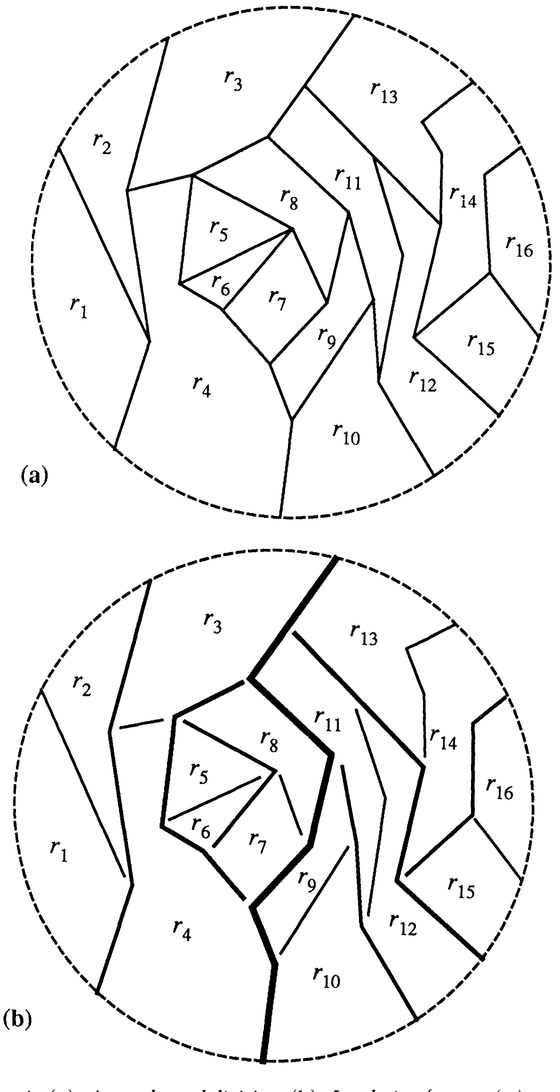


FIG. 4. (a) *A regular subdivision.* (b) *Its chains* $\{proper(\sigma): \sigma \in \Sigma\}$.

that node σ is closest to the root among the nodes whose separator contains e'. Analogously, we show that there is a region r_2 for which $v_2 = LOW(r_2)$. Since γ_2 is a monotone chain from $HIGH(r_1)$ to $LOW(r_2)$, then $r_1 \uparrow r_2$, whence a contradiction. □

This theorem shows that a regular subdivision has a particularly simple separator tree. In the next section we shall show that the property expressed by Theorem 3 is crucial for the efficient dynamization of the chain method for point location. We now slightly generalize the notion of region in a way that will enable us to show that any monotone subdivision embeds a unique regular subdivision. We say that two regions r_1 and r_2 consecutive in $<$ with $r_1 \uparrow r_2$ are *vertically consecutive.* We then have Lemma 3.

LEMMA 3. *If* r_1 *and* r_2 *are two vertically consecutive regions of a monotone subdivision* **P**, *then the monotone chain from* $HIGH(r_1)$ *to* $LOW(r_2)$ *is unique.*

Proof. The lemma holds trivially if $HIGH(r_1) = LOW(r_2)$. Thus, assume the contrary. Since $r_1 \uparrow r_2$, there is a monotone chain γ from $HIGH(r_1)$ to LOW (r_2).

Suppose now, for a contradiction, that there is a monotone chain γ' from $HIGH\ (r_1)$ to $LOW\ (r_2)$ distinct from γ. Then $\gamma \cup \gamma'$ defines the boundary of a partial subdivision that contains at least one region of **P**. For any region r inside this partial subdivision, there are (possibly empty) chains from $HIGH(r_1)$ to $LOW(r)$ and from $HIGH(r)$ to $LOW(r_2)$, so that $r_1 \uparrow r \uparrow r_2$, contrary to the hypothesis that r_1 and r_2 are consecutive in $<$. □

DEFINITION 2. Given two vertically consecutive regions, r_1 and r_2, in **P**, with $r_1 \uparrow r_2$, the unique monotone chain from $HIGH(r_1)$ to $LOW(r_2)$ is called a *channel.*

LEMMA 4. *All channels are pairwise vertex-disjoint.*

Proof. Assume, for a contradiction, that there are two channels γ_1 and γ_2 that are not vertex-disjoint, where γ_1 connects regions r_1 and r_2, γ_2 connects regions r_3 and r_4, and $r_1 < r_2 < r_3 < r_4$ (see Fig. 5). Since γ_1 and γ_2 share a vertex x, there is a chain from $HIGH(r_3)$ to $LOW(r_2)$ consisting of the portion of γ_2 from $HIGH(r_3)$ to x, and the portion of γ_1 from x to $LOW(r_2)$. Hence, we have $r_3 < r_2$, a contradiction. □

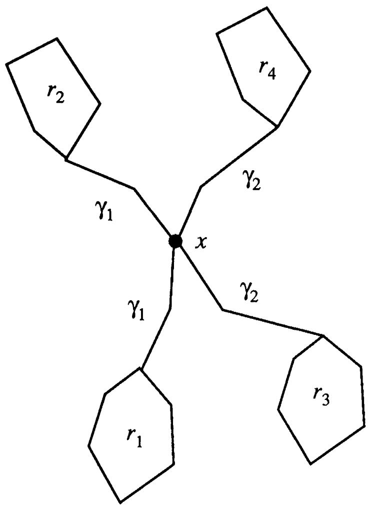


FIG. 5. *Two channels γ_1 and γ_2 that are not vertex-disjoint, where γ_1 connects regions r_1 and r_2, γ_2 connects regions r_3 and r_4, and $r_1 < r_2 < r_3 < r_4$ (for the proof of Lemma* 4).

Given two vertically consecutive regions r_1 and r_2, with $r_1 \uparrow r_2$, we imagine duplicating the channel from r_1 to r_2 and view the measure-zero region delimited by the two replicas as a degenerate polygon joining r_1 and r_2 and merging them into a new region $r_1 \cup r_2$ (see Fig. 6). Clearly, we can merge in this fashion any sequence of vertically consecutive pairs. This is formulated in the following definition.

DEFINITION 3. *Clusters* are recursively defined as follows:

(1) An individual region r is a cluster;

(2) Given two vertically consecutive clusters χ_1 and χ_2, with $\chi_1 \uparrow \chi_2$, their union is a cluster χ, denoted $\chi_1\text{-}\chi_2$ (the horizontal bar denotes the channel).

A *maximal cluster* χ is one that is not properly contained in any other cluster.

The unique subdivision resulting by forming all maximal clusters of **P** is denoted **P***. Figure 2(b) illustrates the regular subdivision **P*** corresponding to the subdivision **P** of Fig. 2(a). Note the clusters $r_2\text{-}r_3$, $r_6\text{-}r_7$, and $r_9\text{-}r_{10}\text{-}r_{11}$.

The above definition leads us to a convenient string notation for the order $<$, as well as for the cluster structure where appropriate. Normally, we shall use lowercase

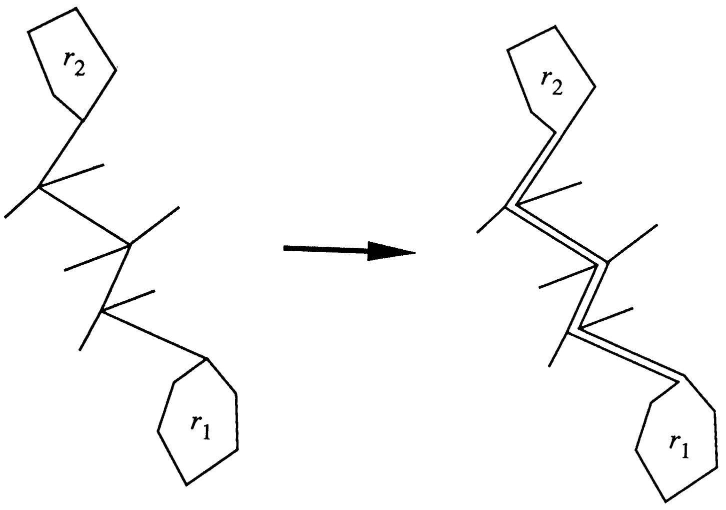


FIG. 6. *Creation of a channel between two vertically consecutive regions.*

roman letters for individual regions, lowercase Greek letters for clusters, and uppercase roman letters otherwise (i.e., for collections of consecutive regions not forming a single cluster). Specifically, we have:

(1) A cluster (possibly, a region) is a string.

(2) Given two strings A and B, such that the rightmost cluster of A and the leftmost cluster of B are consecutive (contiguity), then AB is a string.

A subdivision may be represented by means of its structural decomposition. For example, the subdivision of Fig. 2(b) is described by the following string:

$$r_1\chi_1 r_4 r_5 \chi_2 r_8 \chi_3 r_{12} r_{13} r_{14} r_{15},$$

where $\chi_1 = r_2\text{-}r_3$, $\chi_2 = r_6\text{-}r_7$, and $\chi_3 = r_9\text{-}r_{10}\text{-}r_{11}$.

Later we will find it convenient to explicitly indicate that two consecutive regions r_1 and r_2 may or may not form a cluster. We shall denote this with the string notation $r_1\text{--}r_2$, where "--" means "potential channel."

We conclude this section with the following straightforward observation.

THEOREM 4. *The subdivision* $\mathbf{P}^*$ *obtained by forming all maximal clusters of a monotone subdivision* $\mathbf{P}$ *is regular.*

Note that in the transformation of $\mathbf{P}$ to $\mathbf{P}^*$ only the edges of channels are duplicated. By Lemma 4, each edge is duplicated at most once, thereby ensuring that the number of edges remains $O(n)$.

4. Dynamic point location in a monotone subdivision.

4.1. Data structure. In the following description, we assume that all sorted lists are stored as *red-black trees* [GS], [T]. We recall the following properties of red-black trees that are important in the subsequent time complexity analyses.

(1) Only $O(1)$ rotations are needed to rebalance the tree after an insertion/deletion.

(2) The data structure can be used to implement concatenable queues. Operations *SPLICE* and *SPLIT* of concatenable queues take $O(\log n)$ time and need $O(\log n)$ rotations each for rebalancing.

The search data structure consists of a main component, called the *augmented separator tree*, and an auxiliary component, called the *dictionary*. The *augmented* separator tree **T** has a primary and secondary structure. The *primary structure* is a separator tree for $\mathbf{P}^*$, i.e., each of its leaves is associated with a region of $\mathbf{P}^*$ (a maximal cluster of **P**), and each of its internal nodes is associated with a separator of $\mathbf{P}^*$. (The left-to-right order of the leaves of the primary structure of **T** corresponds to the order $<$ on the regions of $\mathbf{P}^*$.) The *secondary structure* is a collection of lists, each realized as a search tree. Specifically, node σ points to the list $proper(\sigma)$ sorted from bottom to top, and the leaf associated with cluster χ (briefly called "leaf χ") points to the list $regions(\chi)$ of the regions that form cluster χ, also sorted from bottom to top.

Given two regions r_1 and r_2 consecutive in $<$, the separator σ between r_1 and r_2 is associated with the least common ancestor of the leaves associated with the respective clusters of r_1 and r_2 in **T**. By the definition of separator tree, the edges of σ are stored in the secondary structures of nodes along the path from node σ to the root of **T**; by Theorem 3, in a regular subdivision each extreme vertex of $proper(\sigma)$ splits $proper(\sigma')$, for some ancestor node σ' of node σ, into two chains. More precisely, the following simple lemma, stated without proof, makes explicit the allocation of the edges of σ to the nodes of **T**.

LEMMA 5. *Let σ be a separator of $\mathbf{P}^*$, and $\sigma_1, \cdots, \sigma_h$ be the sequence of nodes of* **T** *on the path from the root* $(=\sigma_1)$ *to σ. Then*

$$\sigma = (\alpha_1, \alpha_2, \cdots, \alpha_h, proper(\sigma), \beta_h, \beta_{h-1}, \cdots, \beta_1),$$

where α_i and β_i are (possibly empty) subchains of $proper(\sigma_i)$, $i = 1, \cdots, h$.

To dynamically maintain the channels, it is convenient to keep two representatives e' and e'' of each edge e that are created when e is inserted into **P**. If e does not belong to a channel, e' and e'' are joined into a *double edge* and belong to the same $proper(\sigma)$. If instead e is part of a channel, then e' and e'' are *single edges* and belong to distinct $proper(\sigma')$ and $proper(\sigma'')$. In the latter case e' and e'' are on the boundary of the same cluster χ so that nodes σ' and σ'' are on the path from leaf χ to the root of **T**. Therefore we represent $proper(\sigma)$ by means of two lists: $strand1(\sigma)$ and $strand2(\sigma)$. List $strand1(\sigma)$, called *primary strand*, stores a representative for each edge of $proper(\sigma)$ in bottom-to-top order. List $strand2(\sigma)$, called *secondary strand*, stores a representative for each double edge of $proper(\sigma)$ in bottom-to-top order.

Moreover, associated with each chain $proper(\sigma)$ there are two boolean indicators $t(\sigma)$ and $b(\sigma)$, corresponding, respectively, to the topmost and bottommost vertices of $proper(\sigma)$. Specifically, let σ' be the ancestor of σ such that the topmost vertex of $proper(\sigma)$ is an internal vertex of the chain $proper(\sigma')$ (for the special case where the topmost vertex of σ is at infinity, we let σ' be the father of σ). We define $t(\sigma) = left$ if σ is to the left of σ', and $t(\sigma) = right$ if σ is to the right of σ'. Parameter $b(\sigma)$ is analogously defined.

The *dictionary* contains the sorted lists of the vertices, edges, and regions of **P**, each sorted according to the alphabetic order of their names. With each vertex v we store pointers to the representatives of v in the (at most two) chains $proper(\sigma)$ and $proper(\sigma')$ of which v is a nonextreme vertex. With each edge e we store pointers to the two representatives of e in the data structure. Finally, with each region r we store the vertices $HIGH(r)$ and $LOW(r)$, and a pointer to the representative of r in the list $regions(\chi)$ such that χ is the maximal cluster containing r. The dynamic maintenance of the dictionary in the various operations can be trivially performed in $O(\log n)$ time, and will not be explicitly mentioned in the following.

To analyze the storage used by the data structure, we note the following: the primary structure of **T** has $O(n)$ nodes, since there are $O(n)$ regions (by Euler's formula) and therefore $O(n)$ separators; the secondary structure of **T** also has size

$O(n)$, since there are $O(n)$ edges in $\{proper(\sigma): \sigma \in \mathbf{T}\}$ and $O(n)$ regions in $\{regions(\chi): \chi \in \mathbf{T}\}$ (again, by Euler's formula); the auxiliary component has one record of bounded size per vertex, edge, and region. Therefore, we conclude with the following theorem.

THEOREM 5. *The data structure for dynamic point location has storage space* $O(n)$.

Note that the above data structure is essentially *identical* with the one originally proposed for the static version of the technique [LP]. What is remarkable is that the single-chain structure of the proper edges of any given separator, due to our specific choice of the separator family, is the key for the emergence of full dynamic capabilities.

We now show that the property expressed by Theorem 3 allows us to establish an important dynamic feature of the data structure. According to standard terminology, a *rotation at node* μ of a binary search tree is the restructuring of the subtree rooted at μ so that one of the children of μ becomes the root thereof. A rotation is either *left* or *right* depending on whether the right or left child of μ becomes the new root, respectively. We then have Lemma 6.

LEMMA 6. *A rotation at a node of* $\mathbf{T}$ *can be performed in* $O(\log n)$ *time.*

Proof. Without loss of generality, we consider a left rotation as illustrated in Fig. 7. Clearly, the separators stored at nodes outside the subtree rooted at σ_2 in Fig. 7(a) are not affected by the rotation, nor are those in the subtrees rooted at σ_1, σ_3, and σ_5. Hence, the only alterations involve separators σ_2 and σ_4. If $\sigma_4 \cap proper(\sigma_2) = \emptyset$, then the modification is trivial. Suppose then that $\sigma_4 \cap proper(\sigma_2) \neq \emptyset$. In this case the set $\sigma_4 \cap proper(\sigma_2)$ forms either the initial or the final segment of the chain $proper(\sigma_2)$, or both, for otherwise, regular subdivision $\mathbf{P}^*$ would contain vertically consecutive regions. Thus the update is accomplished by: (1) splitting $proper(\sigma_2)$ into $\gamma_2 = \sigma_4 \cap proper(\sigma_2)$ and its relative complement γ_1: (2) splicing γ_2 with $proper(\sigma_4)$ to form the updated $proper(\sigma_4)$; and (3) setting the updated $proper(\sigma_2)$ equal to γ_1. Note that the extreme vertices of $proper(\sigma_4)$ are obtained in time $O(1)$, and the splitting vertices of $proper(\sigma_2)$ are determined in time $O(\log n)$. Since data structures for $proper(\sigma_2)$ and $proper(\sigma_4)$, (i.e., the red-black trees associated with lists *strand*1 and *strand*2) are also concatenable queues, the splitting and splicing operations are executed in time $O(\log n)$ as well. The parameters $t(\sigma)$ and $b(\sigma)$ for the resulting separators are updated in $O(1)$ time by means of straightforward rules. □

Hereafter, the red-black tree $\mathbf{T}$ is assumed to be balanced. The rest of this section is devoted to the discussion of the algorithms to perform searches, insertions, and deletions.

4.2. Query. To perform point location search for a query point q, we use essentially the same method as in [LP]. The search consists of tracing a path from the root to a leaf χ of $\mathbf{T}$. At each internal node σ we discriminate q against separator σ. Three cases may occur:

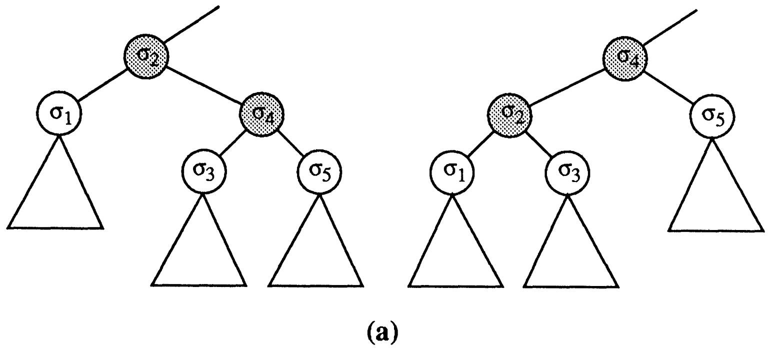
(1) $q \in \sigma$: we return the edge of σ that contains q and stop;

(2) q is to the left of σ: we proceed to the left child of σ;

(3) q is to the right of σ: we proceed to the right child of σ.

Once we reach a leaf χ, we know that q belongs to a region of χ. Since the regions of χ are sorted from bottom to top, such region is determined by searching in the list $regions(\chi)$. The above technique can be viewed as a "horizontal" binary search in the set of separators of $\mathbf{P}^*$, followed by a "vertical" binary search in the set of regions of the leaf χ.

Let e be the edge of σ whose vertical span contains $y(q)$. When $e \in proper(\sigma)$, the discrimination of q against σ is a conventional search in *strand*$1(\sigma)$. When



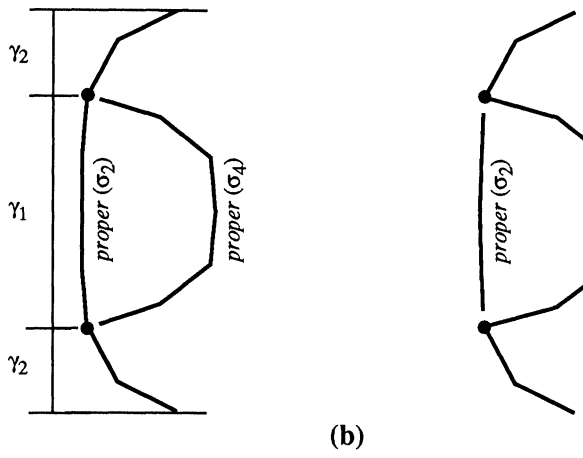


FIG. 7. *Illustration of a (left) rotation. Shaded regions are nodes involved in the update.* (a) *Separator tree.* (b) *Chains of proper edges.*

$e \notin proper(\sigma)$ then we use the pair $(t(\sigma), b(\sigma))$: for example, when e is above $proper(\sigma)$, if $t(\sigma) = left$, then q is discriminated to the right of σ, and to its left otherwise. (This is a minor variant of the criterion adopted in [LP].) The case when e is below $proper(\sigma)$ is treated analogously. This simple analysis confirms that the time spent at each node is $O(\log n)$. We have Theorem 6 [LP].

THEOREM 6. *The time complexity of the query operation is* $O(\log^2 n)$.

4.3. Insertion. We shall first show that the effect of operation $INSERT\text{-}CHAIN(\gamma, v_1, v_2, r; r_1, r_2)$ on the order $<$ of the regions of **P** can be expressed as a syntactical transformation between the strings expressing the order before and after the update. The situation is illustrated in Fig. 8.

On the boundary of r there are two distinguished vertices: $HIGH(r_1)$ and $LOW(r_2)$. Note that $HIGH(r_1) = HIGH(r)$ if v_2 is on the right chain of the boundary of r (and similarly $LOW(r_2) = LOW(r)$ if v_1 is on the left chain). Thus, in general, $HIGH(r_1)$ is on the left chain of the boundary of r, and $LOW(r_2)$ is on the right chain. Using the string notation introduced in § 3, let L and R be the strings corresponding to the

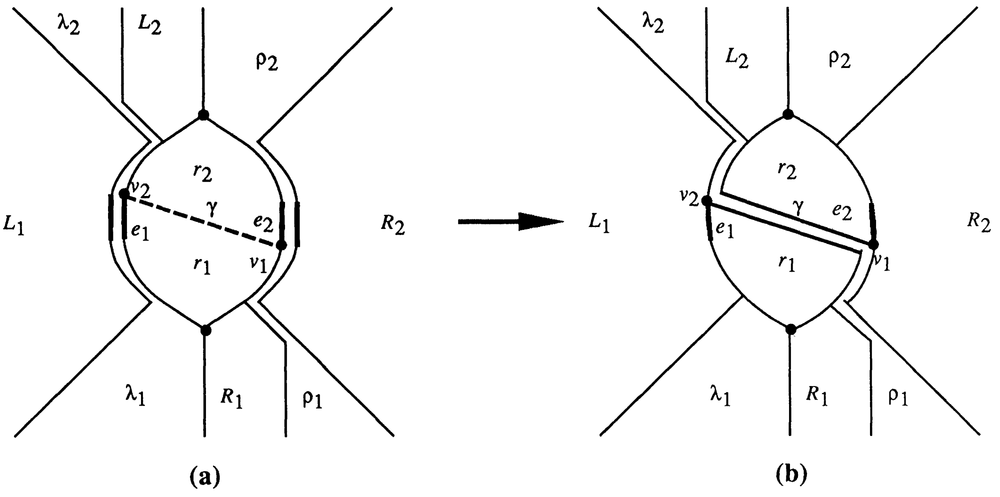


FIG. 8. (a) *Canonical partition of subdivision* $\mathbf{P}^*$ *with reference to region r and vertices* v_1 *and* v_2. (b) *The restructured subdivision after the insertion of chain* γ *between* v_1 *and* v_2.

regions that, respectively, precede and follow r in $<$. Thus, the subdivision $\mathbf{P}^*$ is described by the string LrR.

Let e_1 be the edge of $\mathbf{P}^*$ on the left boundary of r incident on $HIGH(r_1)$ from below, and let χ be the maximal cluster on the left of e_1. In general, this cluster consists of two portions, χ_1 and χ_2 (such that χ_1-χ_2), where χ_2 consists exactly of the regions q' of χ for which $y(LOW(q')) \geqq y(HIGH(r_1))$. Thus, we have $L = L'\chi L''$. We now distinguish three cases and define substrings λ_1, λ_2, L_1, and L_2 as follows.

(1) $\chi_2 \neq \varnothing$. Let $\lambda_1 = \chi_1$, $\lambda_2 = \chi_2$, $L_1 = L'$, $L_2 = L''$, so that $L = L_1\lambda_1$-λ_2L_2.

(2) $\chi_2 = \varnothing$. Let q be the region preceding r (note q could form a cluster with r). We further distinguish:

  (2.1) $y(LOW(q)) \geqq y(HIGH(r_1))$. In this case we let $L = L_1\lambda_2L_2$, where λ_2 is the maximal cluster immediately following χ.

  (2.2) $y(LOW(q)) < y(HIGH(r_1))$. In this case we let $L = L_1\lambda_1$--, where λ_1 is the rightmost maximal cluster of L (but not necessarily a maximal cluster in $\mathbf{P}^*$).

The three cases are conveniently encompassed by the notation

$$L = L_1\lambda_1\text{--}\lambda_2L_2.$$

Note that some of the symbols may denote empty strings.

Analogously, string R can be reformulated as

$$R = R_1\rho_1\text{--}\rho_2R_2$$

with straightforward meanings of the symbols.

Thus, in general, for any given region r and choice of v_1 and v_2 on its boundary, we have the following canonical string decomposition of $\mathbf{P}^*$:

$$L_1\lambda_1\text{--}\lambda_2L_2rR_1\rho_1\text{--}\rho_2R_2.$$

The corresponding partition of the subdivision is illustrated in Fig. 8(a). Examples of configurations corresponding to cases (2.1) and (2.2) are shown in Fig. 9. Namely, Fig. 9(a) shows case (2.1) for L and case (1) for R, Fig. 9(b) shows case (2.2) for L and R, and Fig. 9(c) shows case (2.2) for L and case (1) for R.

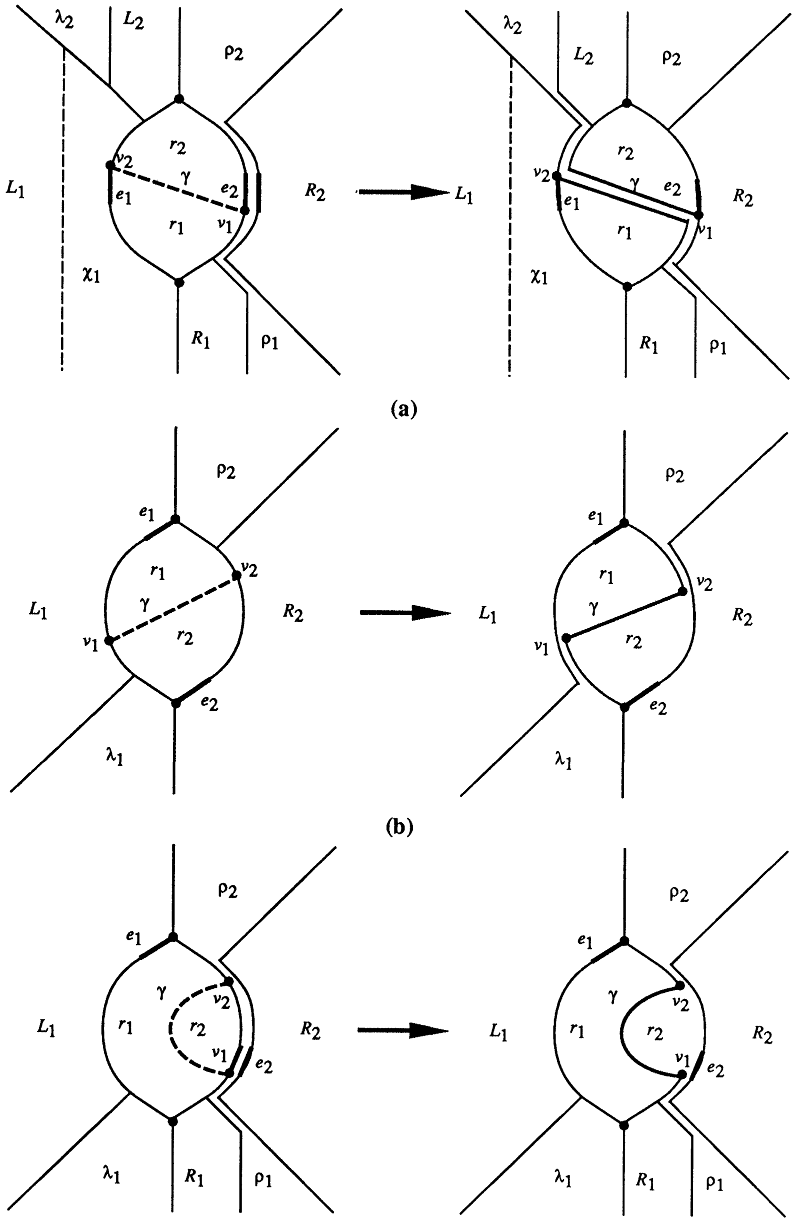


FIG. 9. *Special cases of the structural partition of Fig.* 8. (a) *Case* (2.1) *for L and case* (1) *for R.* (b) *Case* (2.2) *for both L and R.* (c) *Case* (2.2) *for L and case* (1) *for R.*

We now investigate the rearrangement of this order caused by the insertion of chain γ into r. Referring to Fig. 8(b), it is immediately observed that the order after the update is as follows:

$$L_1 < \lambda_1 < r_1 < R_1 < \rho_1 < \lambda_2 < L_2 < r_2 < \rho_2 < R_2.$$

To obtain the string description of the updated subdivision we must determine whether any new channel has been created. Any such channel can only arise in correspondence with a new adjacency caused by the update, specifically for the following pairs: (λ_1, r_1), (ρ_1, λ_2), and (r_2, ρ_2). The channel from λ_1 to r_1 exists only if $y(HIGH(\lambda_1)) \leqq y(LOW(r_1))$, and analogously for the channel from r_2 to ρ_2. Instead, since $y(HIGH(\rho_1)) < y(LOW(\lambda_2))$, the cluster $\rho_1\text{-}\lambda_2$ always exists. Therefore, the order caused by the insertion of γ is represented by the string

$$L_1\lambda_1\text{--}r_1R_1\rho_1\text{-}\lambda_2L_2r_2\text{--}\rho_2R_2.$$

(In purely syntactic terms, this transformation corresponds to rewriting r as $r_2 - r_1$ and then exchanging substrings $r_1R_1\rho_1$ and $\lambda_2L_2r_2$.) This is summarized as follows.

THEOREM 7. *Let $L_1\lambda_1\text{--}\lambda_2L_2rR_1\rho_1\text{--}\rho_2R_2$ be the string description of the order of* $\mathbf{P}^*$, *where L_1, r, and R_2 are nonempty. After operation INSERTCHAIN$(\gamma, v_1, v_2, r; r_1, r_2)$ the new order is described by $L_1\lambda_1\text{--}r_1R_1\rho_1\text{-}\lambda_2L_2r_2\text{--}\rho_2R_2$.*

The algorithm for the *INSERTCHAIN*$(\gamma, v_1, v_2, r; r_1, r_2)$ operation implements the syntactical transformation of the string description by decomposing the subdivision $\mathbf{P}$ into its components L_1, λ_1, λ_2, L_2, r, R_1, ρ_1, ρ_2, and R_2, which are subsequently reassembled according to the new order given by Theorem 7.

To formally describe the algorithm, we denote by $\mathbf{P}(S)$ the partial subdivision associated with a string S of consecutive regions of $\mathbf{P}$. We can represent $\mathbf{P}(S)$ with essentially the same data structure described in § 4.1, and we denote with $\mathbf{T}(S)$ the augmented separator tree for S. Note that $\mathbf{T}(S)$ does not store the edges which form the boundary of S (in the same way as $\mathbf{T}$ does not store the edges at infinity). Partial subdivisions can be cut and merged with the same rules as for the decomposition and concatenation of the corresponding strings.

Let $\mathbf{P}(S_1)$, $\mathbf{P}(S_2)$, and $\mathbf{P}(S)$ be partial subdivisions such that $S = S_1S_2$. We show in the following how to *merge* $\mathbf{T}(S_1)$ and $\mathbf{T}(S_2)$ into $\mathbf{T}(S)$, and how to *cut* $\mathbf{T}(S)$ to produce $\mathbf{T}(S_1)$ and $\mathbf{T}(S_2)$. The merge operation needs also the separator σ forming the common boundary between the two (open) partial subdivisions $\mathbf{P}(S_1)$ and $\mathbf{P}(S_2)$; σ is represented by its primary and secondary strands. The cut operation returns the separator σ. These operations can be implemented by means of the following six primitives.

PROCEDURE *MERGE*$1(S_1, \sigma, S_2; S)$. (It merges partial subdivisions $\mathbf{P}(S_1)$ and $\mathbf{P}(S_2)$, with $S_1 \rightarrow S_2$; σ is the separator between $\mathbf{P}(S_1)$ and $\mathbf{P}(S_2)$.)

(1) Construct a separator tree $\mathbf{T}(S)$ for $\mathbf{P}(S)$, by placing σ at the root, and making $\mathbf{T}(S_1)$ and $\mathbf{T}(S_2)$ the left and right subtrees of σ, respectively.
($\mathbf{T}(S)$ is a legal separator tree for $\mathbf{P}(S)$, but might be unbalanced.)

(2) Rebalance $\mathbf{T}(S)$ by means of rotations.

PROCEDURE *MERGE*$2(\chi_1, \alpha, \chi_2; \chi)$. (It merges partial subdivisions $\mathbf{P}(\chi_1)$ and $\mathbf{P}(\chi_2)$ such that $\chi_1 \uparrow \chi_2$ into $\mathbf{P}(\chi)$, where $\chi = \chi_1\text{-}\chi_2$, and α is the channel between γ_1 and χ_2.)

(1) Separate the two strands of α, and make the secondary strand become a new primary strand.

(2) Splice *regions*(χ_1) and *regions*(χ_2) to form *regions*(χ).

LEMMA 7. *Operations MERGE$1(S_1, \sigma, S_2; S)$ and MERGE$2(\chi_1, \alpha, \chi_2; \chi)$ have time complexity $O(\log^2 n)$ and $O(\log n)$, respectively.*

Proof. The time bound for operation *MERGE*2 follows immediately from the properties of concatenable queues. With regard to *MERGE*1, Step (1) consists of forming $\mathbf{T}(S)$ by joining the primary structures of $\mathbf{T}(S_1)$ and $\mathbf{T}(S_2)$ through node σ, which takes $O(1)$ time. Since we use red-black trees, we can rebalance $\mathbf{T}(S)$ with

$O(\log n)$ rotations [GS], [T, pp. 52–53]. By Lemma 6, each such rotation takes $O(\log n)$ time, so that the total time complexity is $O(\log^2 n)$. □

PROCEDURE $CUT1(S, \chi_1, \chi_2; S_1, \sigma, S_2)$. (It cuts partial subdivision $\mathbf{P}(S)$ into $\mathbf{P}(S_1)$ with rightmost cluster χ_1 and $\mathbf{P}(S_2)$ with leftmost cluster χ_2, such that $\chi_1 \rightarrow \chi_2$, and also returns the separator σ between $\mathbf{P}(S_1)$ and $\mathbf{P}(S_2)$.)

(1) Find the node σ of $\mathbf{T}(S)$ that is the least common ancestor of leaves χ_1 and χ_2.
(2) Perform a sequence of rotations to bring σ to the root of $\mathbf{T}(S)$, where after each rotation we rebalance the subtree of σ involved in the rotation, namely, the left subtree for a left rotation and the right subtree for a right rotation (see Fig. 10).
(3) Set $\mathbf{T}(S_1)$ as the left subtree of σ and $\mathbf{T}(S_2)$ as the right subtree of σ. Return the chain *proper*(σ).

PROCEDURE $CUT2(\chi; \chi_1, \alpha, \chi_2)$. (It cuts partial subdivision $\mathbf{P}(\chi)$ into $\mathbf{P}(\chi_1)$ and $\mathbf{P}(\chi_2)$, where $\chi = \chi_1\text{-}\chi_2$ and α is the channel between χ_1 and χ_2.)

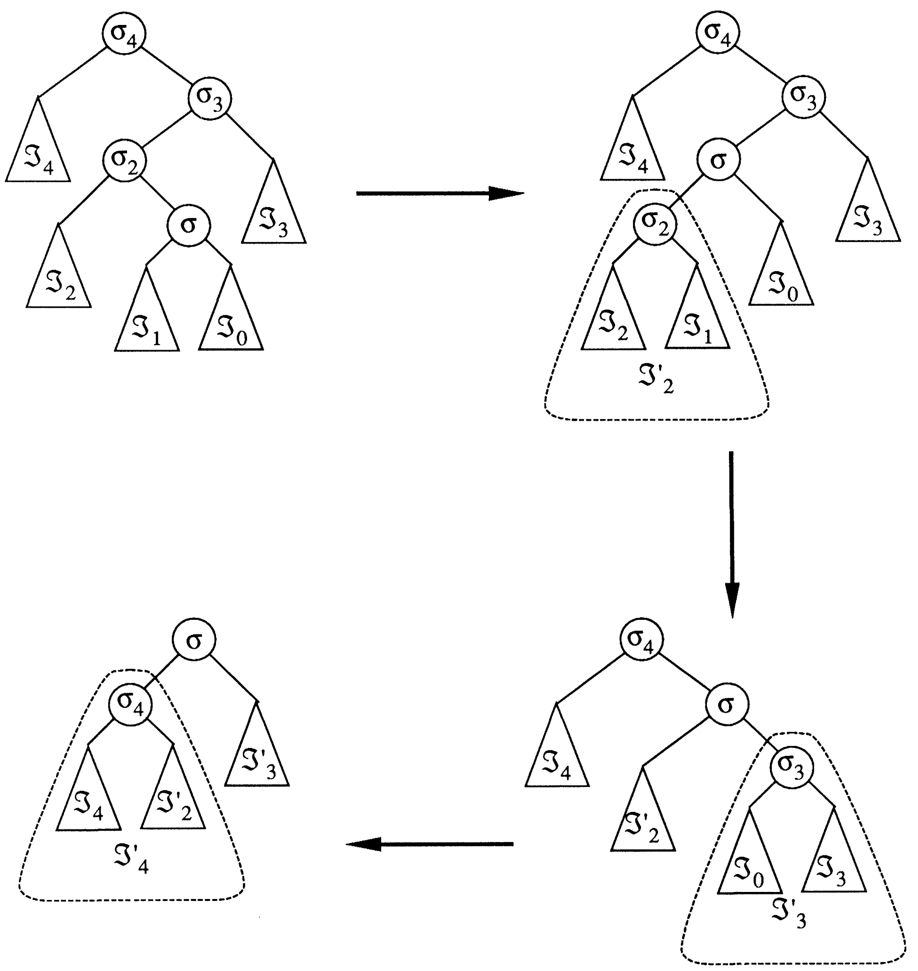


FIG. 10. *Example for step* (2) *of Procedure CUT*1.

(1) Join the two previously separated strands of α, so that the rightmost one becomes the secondary strand of the other.
(2) Split *regions*(χ) into *regions*(χ_1) and *regions*(χ_2).

LEMMA 8. *Operations* $CUT1(S, \chi_1, \chi_2; S_1, \sigma, S_2)$ *and* $CUT2(\chi; \chi_1, \gamma, \chi_2)$ *have time complexity* $O(\log^2 n)$ *and* $O(\log n)$, *respectively.*

Proof. The time bound for operation $CUT2$ is immediate. With regard to operation $CUT1$, step (1) takes $O(\mathit{height}(\mathbf{T}(S))) = O(\log n)$ time. In step (2), we perform no more than $\mathit{height}(\mathbf{T}(S))$ rotations to bring σ to the root. After each such rotation, we have to rebalance a subtree $\mathbf{T}'$, whose left and right subtrees are already balanced, so that the number of rotations required for rebalancing is proportional to the difference of height of the subtrees of $\mathbf{T}'$. Such differences form a sequence whose sum is proportional to $\mathit{height}(\mathbf{T}(S))$ [GS], [T, p. 53]. By Lemma 6, each such rotation takes $O(\log n)$ time, so that the total time complexity is $O(\log^2 n)$. □

PROCEDURE *FINDLEFT*$(e; \chi)$. (It finds the cluster χ to the left of edge e. If e is part of a channel, then χ is the cluster that contains such channel.)
(1) Perform a point location search for (any point of) edge e. The search will stop at a node σ of $\mathbf{T}$ that stores (a representative of) e.
(2) If e is a double edge of σ (i.e., e does not belong to a channel), resume the point location search in the left subtree of σ and return the leaf χ where the search terminates. (This corresponds to searching for a point p^- immediately to the left of edge e.)
(3) Otherwise (i.e., e is a single edge of σ and belongs to a channel) resume the point location search in both subtrees of σ. (This corresponds to searching for points p^- and p^+ immediately to the left and right of edge e, respectively.) One of the two searches, say the left one, will terminate in a leaf, while the other search, say the right one, will stop at a node σ' that stores the other representative of e. (Recall that the two nodes storing e are on the path from leaf χ to the root of $\mathbf{T}$.) We resume the search in the left subtree of σ' and return the leaf χ where the search terminates. (The case where the right search out of σ terminates in a leaf is analogous.)

PROCEDURE *FINDRIGHT*$(e; \chi)$. (It finds the cluster χ to the right of edge e. If e is part of a channel, then χ is the cluster that contains such channel.)
(Analogous to *FINDLEFT*.)

LEMMA 9. *Operations* *FINDLEFT*$(e; \chi)$ *and* *FINDRIGHT*$(e; \chi)$ *have each time complexity* $O(\log^2 n)$.

Proof. Since each edge has two representatives, there are at most two nodes of $\mathbf{T}$ where we proceed to both children. Hence, we visit a total of $O(\log n)$ nodes, spending $O(\log n)$ time at each node. □

The complete algorithm for operation *INSERTCHAIN*$(\gamma, v_1, v_2, r; r_1, r_2)$ is as follows.

ALGORITHM *INSERTCHAIN*$(\gamma, v_1, v_2, r; r_1, r_2)$.
(1) Find regions q and s immediately preceding and following r, respectively; also, find clusters χ_L and χ_R by means of *FINDLEFT*$(e_1; \chi_L)$ and *FINDRIGHT*$(e_2; \chi_R)$. From these obtain λ_1, λ_2, ρ_1, and ρ_2.
(2) Perform a sequence of $CUT1$ and $CUT2$ operations to decompose $\mathbf{P} = \mathbf{P}(L_1\lambda_1\text{--}\lambda_2L_2rR_1\rho_1\text{--}\rho_2R_2)$ into $\mathbf{P}(L_1)$, $\mathbf{P}(\lambda_1)$, $\mathbf{P}(\lambda_2)$, $\mathbf{P}(L_2)$, $\mathbf{P}(r)$, $\mathbf{P}(R_1)$, $\mathbf{P}(\rho_1)$, $\mathbf{P}(\rho_2)$, and $\mathbf{P}(R_2)$. The primary and secondary strands returned by each such

operation, which form the boundaries of the above partial subdivisions, are collected into a list **L**.

(3) Construct the primary and secondary strands of chain γ and add them to **L**.

(4) Destroy $\mathbf{P}(r)$ and create $\mathbf{P}(r_1)$ and $\mathbf{P}(r_2)$.

(5) Test for channels λ_1-r_1 and r_2-ρ_2, and perform a sequence of *MERGE*1 and *MERGE*2 operations to construct the updated subdivision $\mathbf{P}(L_1\lambda_1\text{--}r_1R_1\rho_1\text{-}\lambda_2L_2r_2\text{--}\rho_2R_2)$. The separators and channels needed to perform each such merge are obtained by splitting and splicing the appropriate strands of **L**.

THEOREM 8. *The time complexity of operation INSERTCHAIN*$(\gamma, v_1, v_2, r; r_1, r_2)$, *where* γ *consists of k edges, is* $O(\log^2 n + k)$.

Proof. In step (1), finding q and s takes $O(\log n)$ time. In fact, q is either in the cluster of r or in the cluster immediately preceding the one of r, and analogously for s. By Lemma 9, finding χ_L and χ_R takes $O(\log^2 n)$ time. The remaining computation of λ_1, λ_2, ρ_1, and ρ_2 can be done in $O(\log n)$ time. By Lemma 8, step (2) takes $O(\log^2 n)$ time. Note that the list **L** has $O(1)$ elements. Step (3) can be clearly performed in time $O(k)$. Step (4) takes $O(1)$ time since r, r_1, and r_2 are single-region structures. In step (5), testing for channels λ_1--r_1 and r_2--ρ_2 takes $O(1)$ time. Since the list **L** has $O(1)$ elements, we can construct in $O(\log n)$ time the separators and channels needed for each merge operation of step (5). By Lemma 7 the total time for such merges is $O(\log^2 n)$. □

With regard to the *INSERTPOINT* operation, we locate the edge e in the dictionary, and replace each of the two representatives of e in the data structure with the chain (e_1, v, e_2). This corresponds to performing two insertions into sorted lists, so that we have Theorem 9.

THEOREM 9. *The time complexity of operation INSERTPOINT*$(v, e; e_1, e_2)$ *is* $O(\log n)$.

A similar argument shows Theorem 10.

THEOREM 10. *The time complexity of operation MOVEPOINT*$(v; x, y)$ *is* $O(\log n)$.

4.4. Deletion. The transformations involved in a *REMOVECHAIN* operation are exactly the reverse of the ones for the *INSERTCHAIN* operation. We observe that all the updates performed in the latter case are totally reversible, which establishes Theorem 11.

THEOREM 11. *The time complexity of operation REMOVECHAIN*$(\gamma; r)$, *where* γ *consists of k edges, is* $O(\log^2 n + k)$.

The same situation arises with respect to the *INSERTPOINT* and *REMOVEPOINT* operations, so that we have Theorem 12.

THEOREM 12. *The time complexity of operation REMOVEPOINT*$(v; e)$ *is* $O(\log n)$.

Theorem A stated in § 1 results from the combination of the above Theorems 5, 6, 8, 9, 11, and 12.

5. Conclusion and open problems. The above technique represents a reasonably efficient solution of the dynamic point location problem. It requires no new sophisticated or bizarre data structures, and it appears eminently practical.

It remains an open problem whether $O(\log n)$ optimal performance is achievable for query/update times; in particular, whether the technique of fractional cascading, which achieved optimality for its suboptimal static predecessor [EGS], can also be successfully applied to the presented technique.

Another challenging open question is to extend our technique to general planar subdivisions.

REFERENCES

[B] G. Birkhoff, *Lattice Theory*, American Mathematical Society Colloquium Publications, Vol. 25, American Mathematical Society, Providence, RI, 1979.

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

[EKA] M. Edahiro, I. Kokubo, and T. Asano, *A new point-location algorithm and its practical efficiency—Comparison with existing algorithms*, ACM Trans. Graphics, 3 (1984), pp. 86-109.

[EGS] H. Edelsbrunner, L. J. Guibas, and J. Stolfi, *Optimal point location in a monotone subdivision*, SIAM J. Comput., 15 (1986), pp. 317-340.

[F] O. Fries, *Zerlegung einer planaren Unterteilung der Ebene und ihre Anwendungen*, M.S. thesis, Inst. Angew. Math. und Inform., Univ. Saarlandes, Saarbrücken, FRG, 1985.

[FMN] O. Fries, K. Mehlhorn, and S. Naeher, *Dynamization of geometric data structures*, in Proc. ACM Symposium on Computational Geometry, 1985, pp. 168-176.

[GS] L. J. Guibas and R. Sedgewick, *A dichromatic framework for balanced trees*, in Proc. 19th Annual IEEE Symposium on Foundations of Computer Science, 1978, pp. 8-21.

[Ka] T. Kameda, *On the vector representation of the reachability in planar directed graphs*, Inform. Process. Lett., 3 (1975), pp. 75-77.

[KR] D. Kelly and I. Rival, *Planar lattices*, Canadian J. Math., 27 (1975), pp. 636-665.

[Ki] D. G. Kirkpatrick, *Optimal search in planar subdivisions*, SIAM J. Comput., 12 (1983), pp. 28-35.

[LP] D. T. Lee and F. P. Preparata, *Location of a point in a planar subdivision and its applications*, SIAM J. Comput., 6 (1977), pp. 594-606.

[LEC] A. Lempel, S. Even, and I. Cederbaum, *An Algorithm for Planarity Testing of Graphs*, Theory of Graphs, Internat. Symposium, Rome, Italy, 1966, P. Rosenstiehl ed., Gordon and Breach, New York, 1967, pp. 215-232.

[LT] R. J. Lipton and R. E. Tarjan, *Applications of a planar separator theorem*, in Proc. 18th Annual IEEE Symposium on Foundations of Computer Science, 1977, pp. 162-170.

[M] K. Mehlhorn, *Data Structures and Algorithms* 3: *Multi-dimensional Searching and Computational Geometry*, Springer-Verlag, Berlin, New York, 1984.

[O] M. Overmars, *Range searching in a set of line segments*, in Proc. ACM Symposium on Computational Geometry, 1985, pp. 177-185.

[P1] F. P. Preparata, *A new approach to planar point location*, SIAM J. Comput., 10 (1981), pp. 473-483.

[P2] ———, *Planar point location revisited: A guided tour of a decade of research* (invited paper), Lecture Notes in Computer Science, Vol. 338 (Proc. 1988 FST & TCS Symposium, Pune, India), Springer-Verlag, Berlin, New York, 1988, pp. 11-17.

[PS] F. P. Preparata and M. I. Shamos, *Computational Geometry*, Springer-Verlag, Berlin, New York, 1985.

[ST] N. Sarnak and R. E. Tarjan, *Planar point location using persistent search trees*, Comm. ACM, 29 (1986), pp. 669-679.

[TP] R. Tamassia and F. P. Preparata, *Dynamic maintenance of planar digraphs, with applications*, *Algorithmica*, to appear. Tech. Report ACT-92, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL, 1988.

[T] R. E. Tarjan, *Data Structures and Network Algorithms*, CBMS-NSF Regional Conference Series in Applied Mathematics, 44, Society for Industrial Applied Mathematics, Philadelphia, PA, 1983.