A Simple Algorithm for Maintaining the Center of a Planar Point-Set

Reuven Bar-Yehuda [*] Alon Efrat [†] Alon Itai [‡]

June 29, 1993

Abstract

We give a simple deterministic algorithm for maintaining the center of a planar point set, in any convex distance function, when points are added to the set. Our algorithm requires $O(\log^3 n)$ time and $O(1)$ space per insertion. In the case of L_2 norm, our algorithm improves the worst case ($O(\log^4 n)$) of Eppstein's algorithm [3].

Our algorithm can be implemented for other purposes. Using out algorithm with the paradigm suggested by Eppstein [3], we can solve the *two center problem* with respect to any convex distance function (to be defined below) in time $O(n^2 \log^3 n)$. We can also find the farthest point Voronoi Diagram with respect to any convex distance function in time $O(n \log n)$.

1 Introduction

The problem of finding the smallest enclosing disk for a planar set of points has been well treated in literature. It goes back to Sylvester in 1857. See [6] for a historical survey. Nimrod Megiddo was the first to give a linear time algorithm for this problem [6]. Since this problem can be stated as a mathematical programming problem with linear constraints and convex object function in $\mathbb{R}^3$, (see for example [11]) most of the linear programming algorithms can handle this task, (c.f [5] for an expected linear time algorithm).

Let $\mathbf{F}$ be a closed convex shape, containing the origin of the XY coordinate system. We will assume that $\mathbf{F}$ is a simple shape, in the sense that its boundary can be represented as a collection of a constant number of algebraic curves, each of constant degree. The norms L_k, for any k, is a special case of this definition, when $\mathbf{F}$ is the unit ball in this norm.

[*]Department of Computer Science, The Technion

[†]A.E. was with the Department of Computer Science, The Technion, and is now with the School of Mathematical Sciences, Tel-Aviv University. alon@cs.technion.ac.il

[‡]Department of Computer Science, The Technion

Notation 1.1 *Given a convex set* $\mathbf{F}$ *we denote by* $\mathbf{F}_q(r) = r\mathbf{F} + q$, *that is,* $\mathbf{F}$ *scaled by* r *and then translated to* q. $\mathbf{F}_q(r)$ *is a* homothet *of* $\mathbf{F}$ *of* radius r *and* center q. $\bar{\mathbf{F}}$ *denotes* $\mathbf{F}$, *reflected on both axes, and* $\bar{\mathbf{F}}_q(r) = r\bar{\mathbf{F}} + q$.

Definition 1.1 *The* convex distance function *between* q_1 *and* q_2 *induced by* $\mathbf{F}$ *is*

$$d(q_1, q_2)_{\mathbf{F}} = \min\{\alpha \in \mathbb{R} : q_1 \in \mathbf{F}_{q_2}(\alpha)\}.$$

Notation 1.2 *Given a planar set of points* S, *let* $SEH(S)$ *denote the smallest homothet of* $\mathbf{F}$ *containing* S, *let* $\rho(S)$ *denote its radius, and the* center *of* S *is the center of* $SEH(S)$.

Here we present a simple algorithm for maintaining the center of a planar point-set, under any convex distance function, when points are given in a semi-dynamic fashion. That is, points are added to the set, and after each such insertion, we wish to find the center of the new set. Eppstein [3] gave an algorithm for dynamically solving three dimensional linear-programming problem, and one of his applications was to maintain the center in the L_2 norm of a planar point set, in expected time $O(\log^2 n \log\log n)$, and worst-case $O(\log^4 n)$ time.

In this paper we improve this worst case, for any convex distance function. We present an algorithm that performs an insertion in $O(\log^3 n)$ time using $O(n)$ space.

Our algorithm can easily be modified to require $O(\log^3 n/\log\log n)$ time and $O(n \log n)$ space. Using a data structure suggested by Mehlhorn et al. [9] the space can remain linear without sacrificing the $O(\log^3 n/\log\log n)$ time bound.

Results similar to ours can be achieved using the recent result of Kirkpatrick and Snoeyink [4]. A related problem is the **two-center problem**. In this problem, one tries to cover a planar set of points S by two disks such that the radius of the larger among the two disks in minimized, or the sum of radii is minimized. Agarwal and Sharir [1] have solved the first problem in $O(n^2 \log^3 n)$, using the parametric search technique of

Megiddo [7]. Eppstein [3] improved their solution to $O(n^2 \log^2 n \log\log n)$ expected time. In the worst case, his algorithm requires $O(n^2 \log^4 n)$. He constructed the arrangement of lines dual to the points in S, traversed this arrangement and calculated for each cell the radii of the corresponding two circles determined by this cell. With the same idea but using our algorithm for maintaining the center of a point-set, we can deduce an $O(n^2 \log^3 n)$ time deterministic algorithm for solving this problem. Our algorithm is significantly simpler than the one proposed by Agarwal and Sharir, and can also be implemented as well for any convex distance function.

2 Statically Finding *SEH*

Let $S = \{p_1, \cdots, p_n\}$, be a set of points in the plane, and let $\mathbf{Q}_S(r) = \bigcap_{i=1}^{n} \bar{\mathbf{F}}_{p_i}(r)$.

Claim 2.1 $r \geq \rho(S)$ *iff* $\mathbf{Q}_S(r) \neq \emptyset$.

In order to find $\rho(S)$ we first compute which edges appear on $\partial\mathbf{Q}_S(r)$ when $r = r^\infty$ is very large, ($r^\infty = \infty$ for all practical purposes). Then we reduce r while evaluating $\mathbf{Q}_S(r)$. We stop the process when $\mathbf{Q}_S(r)$ is reduced to a single point. $\mathbf{Q}_S(r)$ (if not empty) can be thought of as a convex polygon whose boundary edges are parts of the boundary of $\bar{\mathbf{F}}$, (see fig. 1), where for any $A \subseteq \mathbb{R}^2$, ∂A denotes its boundary. Every connected component of $\mathbf{Q}_S(r) \bigcap \partial\bar{\mathbf{F}}_{p_i}(r)$ is called an **F**-*edge*. Henceforth, the term *homothet* will refers to a homothet of **F**.

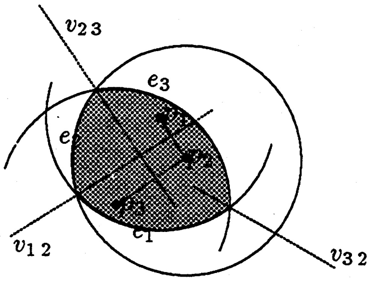

Figure 1: $\mathbf{Q}_S(r)$ *(the dotted area) is the intersection of homothets of* **F** *when* **F** *is a disk. Note that the trajectories are straight lines.*

2.1 Some facts about homothets and F-polygons

The proofs of the following facts appear in the full paper.

Fact 2.2 *The boundaries of any two homothets intersect at most twice.*

Definition 2.1 *For any two points $p_i, p_j \in S$ define the* trajectory

$$v_{i,j} = \bigcup_{\alpha>0} \partial\bar{\mathbf{F}}_{p_i}(\alpha) \cap \partial\bar{\mathbf{F}}_{p_j}(\alpha),$$

[1] *which is a simple path in $\mathbb{R}^2$.*

Fact 2.3 *Any two trajectories (determined by two pairs of points, not necessarily all four distinct) intersect in at most one point.*

Fact 2.4 *When r is reduced, no* **F**-*edge appears between two consecutive* **F**-*edges. Hence the number of* **F**-*edges is a nondecreasing function of r.*

Note that when we know the identity of the three remaining **F**-edges, we can readily find *SEH* .

Definition 2.2 *Let r be fixed. Let $e(r)$ be an* **F**-*edge of $\mathbf{Q}_S(r)$ and let $v^1(r)$ and $v^2(r)$ be its endpoints. r' is a* tentative critical radius *of $e(r)$ if the trajectories of its two endpoints meet at r', i.e., $v^1(r') = v^2(r')$. A radius r' is a* critical radius *of $e(r)$ if $e(r)$ disappears from $\partial\mathbf{Q}_S(\cdot)$ at r'.*

Definition 2.3 *Let $p_i \in S$.* $\bar{\mathbf{F}}_{p_i}(r)$ contributes *to $\mathbf{Q}_S(r)$ if $\partial\bar{\mathbf{F}}_{p_i}(r) \bigcap \partial\mathbf{Q}_S(r)$ is not empty.*

Fact 2.5 *For any r, $\partial\mathbf{Q}_S(r)$ contains at most n* **F**-*edges, since every homothet, $\bar{\mathbf{F}}_p(r)$, $p \in S$ contributes at most one* **F**-*edge.*

Fact 2.6 $\bar{\mathbf{F}}_{p_i}(r^\infty)$, *$p_i \in S$ contributes to $\partial\mathbf{Q}_S(r^\infty)$ (i.e. $\partial\bar{\mathbf{F}}_p(r^\infty) \cap \partial\mathbf{Q}_S(r^\infty) \neq \emptyset$) if and only if p_i satisfies*

- *p_i is a vertex of* CH(S), *the convex hull of S,*
- *there exists a translation $x \in \mathbb{R}^2$ such that $\bar{\mathbf{F}}_x(r^\infty)$ contains all the points of S except p_i in its interior.*

Let V' be the set of vertices of $CH(S)$ that correspond to contributing homothets.

[1] we assume that the boundary (∂) operation has higher priority than union and intersection

Fact 2.7 *The order of edges on $\mathbf{Q}_S(r^\infty)$ is the same as the order by which points of V' appear on $CH(S)$.*

To find $\mathbf{Q}_S(r^\infty)$ we first find $CH(S)$. In order to verify the second condition of Fact 2.6, it suffices to check whether there exists an x such that $\bar{\mathbf{F}}_x(r^\infty)$ contains p_1 and p_2, but does not contain p', where p_1, p_2 are the neighboring vertices of p' on $CH(S)$. Note that we don't have to find r^∞ explicitly.

Let the **F**-edges of $\mathbf{Q}_S(r^\infty)$ be denoted (ordered clockwise) $e_0(r^\infty), \dots e_{m-1}(r^\infty)$ [2]. Let e_k be the first **F**-edge to disappear. When this occurs, the trajectories $v_{k-1,k}$ and $v_{k,k+1}$ meet.[3]. To discover which **F**-edge disappears first, we find the intersections of all neighboring trajectories $v_{j,j+1}$, $(j = 0, \dots m-1)$, and their corresponding *tentative critical radii.* Thus this point is the intersection of the trajectories $v_{k,k-1}$ and $v_{k,k+1}$.

The two edges e_{k-1}, e_k meet at the trajectory $v_{k-1,k}$. When e_k is deleted, e_{k-1} and e_{k+1} become neighbors. Thus, the trajectories $v_{k-1,k}$ and $v_{k,k+1}$ are of no further interest and we compute $v_{k-1,k+1}$. The tentative critical radii of e_{k-1} and e_{k+1} are recomputed.

```
Procedure Static-Find-SEH(S)

1. Find the vertices of $CH(S)$.  For each
   vertex determine if it is in $V'$.
   Reindex the vertices in $V'$, so
   $V' = \{p_1, \dots p_m\}$.
2. Compute the tentative critical values of
   $e_0, \dots e_{m-1}$ and put them in a heap.
3. While the heap contains more than three
   tentative critical radii do
   (a) Find $r_i$, the largest tentative
       critical radius in the heap.
   (b) Remove $r_i$ from the heap.  (Hence $r_i$
       becomes a critical radius).
   (c) Let $e_i$ be the edge corresponding to
       $r_i$.  Recompute the tentative critical
       radii of $e_i$'s neighbor edges, and
       update them in the heap.
4. The three last remaining edges in the
   heap determine $SEH(S)$.
```

2.1.1 Analysis

Finding the convex hull of S can be done in $O(n \log n)$ time [11]. Finding V' takes $O(n)$ time, since for each vertex of $CH(S)$ it takes $O(1)$ time to check if it is in V'.

We maintain the tentative critical radii in a heap. In this data structure finding the minimum requires $O(1)$ time, and the time of insertion or deletion is $O(\log n)$. Each deletion of an **F**-edge requires deleting its tentative critical radius and updating (deleting and inserting) its two neighbors and their tentative radii. Thus, there are at most $3n$ deletions and $2n$ insertions in the course of the algorithm. Since building the initial heap requires $O(n)$, we get:

Theorem 1 *Procedure* **Static-Find-*SEH*** *finds the smallest homothet of* **F** *containing a planar set of n points, in $O(n \log n)$ time and $O(n)$ space.*

An algorithm with similar spirit was proposed by Skyum [13] for finding the static center of a set of points in the L_2 norm. Since the vertices of $\mathbf{Q}_S(\cdot)$ always travel on edges of the farthest point Voronoi Diagram, our algorithm, as well as Skyum's, can be used to compute this diagram.

3 A Dynamic Approach to the *SEH* Problem

As explained in Section 2, the polygon $\mathbf{Q}_S(r)$ shrinks and loses edges as we decrease r. We call this history the *Dynamic Polygon, DP*. The *DP* of S can be discovered while executing **Static-Find-*SEH*** on the points of S. In Section 5.2 we describe a data-structure for storing the *DP*. Assume we are given t disjoint sets of points $S_1, \dots S_t \subseteq \mathbb{R}^2$, and we wish to compute $SEH(\bigcup_{i=1}^t S_i)$. It is easy to see that $r \geq \rho(\bigcup_{i=1}^t S_i)$ **iff** $\bigcap_{i=1}^t \mathbf{Q}_{S_i}(r) \neq \emptyset$.

This relation leads us to the following idea: Partition the (current) set of points S into a collection of subsets

$$S = S_1 \cup S_2 \cup \dots \cup S_t ,$$

and for each such S_i, execute **Static-Find-*SEH***. Store the *DP* $\mathbf{Q}_{S_i}(\cdot)$. When a new point p is given, first compute the new *SEH*, using the technique presented in Section 4. Afterwards update the collection of *DP*s. We shall maintain at most $t = O(\frac{\log n}{\log \log n})$ *DP*s. The question of when to prepare a new *DP* for a new subset of points, as well as when to get rid of an old *DP*, is addressed in Section 5.2

4 Finding the new *SEH*

4.1 Overview

In this subsection we describe how to compute the new *SEH* when a new point is given. (The *DP*s are

[2] arithmetic on indices is done modulo m

[3] note that due to Fact 2.5, we can reindex the points of S such that $e_i \subseteq \partial\bar{\mathbf{F}}_{p_i}$. The indices of the points of S which do not contribute any edge are given arbitrary.

not modified.)

Assume t sets of points $S_1, \dots S_t$ are given, each with its DP, $\mathbf{Q}_{S_i}(\cdot)$. Let $S = \bigcup_{i=1}^{t} S_i$, and let c be the center of $SEH(S)$. Now a new point p is given, and we seek $SEH(\{p\} \cup S)$. If $p \in SEH(S)$ then, of course, there is nothing to calculate. Assume, therefore, $p \notin SEH(S)$. Let $r^* = \rho(\{p\} \cup \bigcup_{i=1}^{t} S_i)$, and let $\mathcal{F}(r) = \bar{\mathbf{F}}_p(r) \cap \bigcap_{i=1}^{t} \mathbf{Q}_{S_i}(r)$.

Claim 4.1 *If $p \notin SEH(S)$ then the center of $SEH(\{p\} \cup S)$ is on $\partial\bar{\mathbf{F}}_p(r^*)$.*

As the claim stated, we can limit our attention to events that happen on $\partial\bar{\mathbf{F}}_p(\cdot)$. Define $J_{S_i}(r) = \mathbf{Q}_{S_i}(r) \cap \partial\bar{\mathbf{F}}_p(r)$. Claim 4.1 tells us that it suffices to look for the smallest radius r for which $\bigcap_{i=1}^{t} J_{S_i}(r) \neq \emptyset$ [4]. We shall now describe the outline of the algorithm for finding r^*.

Procedure Find-New-*SEH*

```
1. Find two radii $\delta_1, \delta_2$, such that $\mathcal{F}(\delta_1) = \emptyset$,
   $\mathcal{F}(\delta_2) \neq \emptyset$, and there is no critical
   radius of any of the DPs in $(\delta_1, \delta_2)$.
2. Find two radii $\lambda_1, \lambda_2$, such
   that $\mathcal{F}(\lambda_1) = \emptyset$, $\mathcal{F}(\lambda_2) \neq \emptyset$, and for every
   $r \in (\lambda_1, \lambda_2)$ no vertex of any of the DPs is
   on $\mathbf{F}_p(r)$.
3. For each pair $(S_i, S_j)$ find $\rho_{ij}$, the
   minimal radius at which $J_{S_i}(r) \cap J_{S_j}(r) \neq \emptyset$.
   (Can be done using elementary geometry in
   $O(1)$ time for each pair.)
4. $r^* = \max_{1 \le i,j \le t} \rho_{ij}$.
```

4.1.1 Analysis

Steps 1 and 2 will be shown to take $O(\log^3 n)$ time, Steps 3 and 4 spend $O(1)$ time for each pair of *DPs* . We shall show that the number of *DPs* $t = O(\frac{\log n}{\log\log n})$, and hence steps 3 and 4 require in total $O(t^2) = O(\log^2 n)$ time. Hence the total running time of **Find-New-*SEH*** is $O(\log^3 n)$.

4.2 Constructing the Oracle $A(\cdot)$

Given a radius r, the oracle determines whether $r > r^*$. This is equivalent to checking whether

$$\bar{\mathbf{F}}_p(r) \cap \bigcap_{p_i \in S} \bar{\mathbf{F}}_{p_i}(r) \neq \emptyset.$$

[4]This idea is reminiscent of the parametric search technique of Megiddo [7].

However, since $\bigcap_{p_i \in S} \bar{\mathbf{F}}_{p_i}(r)$, and $J_{S_i}(r) = \mathbf{Q}_{S_i}(r) \cap \bar{\mathbf{F}}_{p_i}(r)$, we just check whether $\bigcap_{i=1}^{t} J_{S_i}(r) \neq \emptyset$. The sets $J_{S_i}(r)$ are subpaths of $\partial\bar{\mathbf{F}}_p(r)$, so once the $J_{S_i}(r)$ are determined, we have a one-dimensional search problem.

Our first goal is to determine the intersection points of $J_{S_i}(r)$ with $\partial\bar{\mathbf{F}}_p(r)$. A pair of points $q_1, q_2 \in \partial\mathbf{Q}_{S_i}(r)$ is *antipodal* if one belongs to $\bar{\mathbf{F}}_p(r)$ and the other doesn't. Consider any two edges f_i and f_j, and two points $q_1 \in f_i$ and $q_2 \in f_j$. If they are not antipodal, then by geometrical considerations (details will appear in the full paper) we can determine in $O(1)$ time in which of the two path of $\partial\mathbf{Q}_{S_i}(r)$ determined by points there is NO antipodal point to q_1. If f_1 and f_m are both outside $\bar{\mathbf{F}}_p(r)$, or both entirely contained in $\bar{\mathbf{F}}_p(r)$, (which is easily checked in $O(1)$ time) then if there is an antipode to some point on f_1 then this point must be on the path $f_2, \dots f_{m-1}$. The edges of $\mathbf{Q}_{S_i}(r)$ are stored as a search tree. Let f_k be the edge corresponding to its root, and let q_k a point in f_k. If q_k is not antipodal, we again check whether an antipodal point can be either on $f_2, \dots f_{k-1}$ or on $f_{k+1}, \dots f_{m-1}$. Assuming the latter, we found a point $q_l \in f_l$, where f_l is the root of the left subtree of f_k , and check again for an antipodal point, either on $f_k, \dots f_{l-1}$ or on $f_{l+1}, \dots f_{m-1}$. So in $O(\log n)$ such checks we either find an antipodal pair, and then using binary search we find $J_{S_i}(r)$, or determine $\mathbf{Q}_{S_i}(r) \bigcap \bar{\mathbf{F}}_p(r) = \emptyset$.

Recall that the furthest point q' from c among $\partial\bar{\mathbf{F}}_p(r)$ is not contained in any $J_{S_i}(r)$. Travelling along $\partial\mathbf{Q}_{S_i}(r)$ clockwise from q', we meet start-points and end-points of the $J_{S_i}(r)$. $\bigcap_{i=1}^{t} J_{S_i}(r) \neq \emptyset$ if and only if we meet the last end-point after we have met all the start-points. This is easily checked in $O(t)$ time.

4.3 Finding δ_1, δ_2

We have to determine, for every critical radius of every DP , what is its size relative to r^*. Let $I_{S_i}^{(k)}$ be the set of the critical radii of $\mathbf{Q}_{S_i}$, for which their relation to r^* is not known at the beginning of the kth iteration. Note that due to transitivity, if r_l, $r_j \in I_{S_i}^{(k)}$ then any other critical value r_k, between r_l and r_k, is also in I_{S_i}. A set $I_{S_i}^{(k)}$ is *alive* if it contains more than a single critical radius.

Procedure **Find** δ_1, δ_2

```
1. Let $k \leftarrow 1$. For $i = 1, \dots t$, $I_{S_i}^{(1)}$ consists
   of all the critical values of the live
   sets.
2. While there are live sets repeat
```

```
(a) Let $m_{S_i}^{(k+1)}$ be the medians of the live
    $I_{S_i}^{(k)}$.
(b) Let $m^{(k)}$ be the medians of $m_{S_i}^{(k)}$, for
    live $I_{S_i}^{(k)}$.  Call the oracle $\mathbf{A}(m^{(k)})$.
(c) Assuming
    $m^{(k)} \geq r^*$.  Half of the medians, say
    $m_{S_1}^{(k)} \ldots m_{S_{\frac{t}{2}}}^{(k)}$ are larger than $r^*$, and
    at least half of the radii in
    $I_{S_1}^{(k)} \ldots I_{S_{t/2}}^{(k)}$ are larger than $r^*$.
(d) Let $I_{S_i}^{(k+1)}$ be all the radii in $I_{S_i}^{(k)}$
    whose relation to $r^*$ has not been
    determined, $i = 1, \ldots \frac{t}{2}$.
    Let $I_{S_i}^{(k+1)} = I_{S_i}^{(k)}$, $i = \frac{t}{2}+1, \ldots t$
    A symmetric case occurs if $m^{(k)} < r^*$.
(e) $k \leftarrow k+1$
```

4.3.1 Analysis

Claim 4.2 $O(\log n \log t)$ *calls to the oracle suffice.*

Proof: Define a *phase* as the sequence of iterations in which the number of live sequences drops by 1/2. Thus the number of phases is $\log_2 t$. We now show that the time needed for a phase is $O(\log n)$. Let $\pi = \sum_{i=1}^{t} \log_2 |S_i|$. At the beginning of a phase, $\pi \leq t \log n$. Each call to the oracle reduces the length of half of the live sequences by (at least) half. Hence each call to the oracle reduces π by $\frac{t}{4}$. Let k be the number of such calls in one phase, then

$$k \leq \frac{t \log n}{t/4} = 4 \log n. \quad \blacksquare$$

Our data structure does not allow us to find the exact median of each $I_{S_i}^{(k)}$. Instead, we find in each iteration an $m_{S_i}^{k}$ which is *α-median*, that is

$$\alpha \leq \frac{|I_{S_i}^{(k)} \cap (-\infty, m_{S_i}^{k}]|}{|I_{S_i}^{(k)}|} \leq 1-\alpha$$

for some fixed $\alpha \in [0, 1/2)$. However this does not affect the asymptotic complexity of the running time of the algorithm.

The running time of the oracle is $O(t \log n)$, and therefore the total time needed is $O(t \log n \times \log n \log t)$. We shall see in Section 5.2 how to maintain $t = O(\frac{\log n}{\log \log n})$ *DP*s so this running time is $O(\log^3 n)$.

4.4 Finding λ_1, λ_2

We seek for two values λ_1, λ_2 such that

$$F(\lambda_1) = \emptyset \qquad F(\lambda_2) \neq \emptyset$$

and in the process of shrinking r from λ_2 to λ_1, $\partial \mathbf{F}_p(r)$ does not contain any vertex. Note that in the course of executing the oracle for finding δ_1, δ_2, we have found for each $\mathbf{Q}_{S_i}(r)$ two paths $P_{2i}(r)$, $P_{2i+1}(r) \subseteq \partial \mathbf{Q}_{S_i}(r)$, such that for any $r \in (\delta_1, \delta_2)$, $\partial \mathbf{F}_p(r)$ intersects $P_{2i}(r)$ and $P_{2i+1}(r)$.

Let $v(r)$ be a vertex of $\mathbf{Q}_{S_i}(r)$. Let $R(v)$ denote the (unique) radius r at which $v(r)$ is in $\partial \mathbf{F}_p(r)$. We find, for each path P_k two consecutive vertices $v_j(r)$, $v_{j+1}(r)$ such that $R(v_j) < r^* < R(v_{j+1})$. We can perform a binary search among its vertices to find v_j, v_{j+1}. Again we use our oracle $\mathbf{A}(\cdot)$ and use the same mechanism developed in the previous section, with the same time bounds.

5 Data Structures

This section is divided into two subsections. In the first, we show how to store the history of the shrinking of a *DP*. In the second, we show how to update our collection when a new points is added.

5.1 Implementation of a single dynamic polygon $\mathbf{Q}_{S_i}(\cdot)$

We use a persistent data structure (c.f. Sarnak and Tarjan [12]), for representing planar subdivision. This structure enables us for each r to access the **F**-edges of $\mathbf{Q}_{S_i}(r)$.

As explained in Section 2, for $r = r^{\infty}$ we have m **F**-edges and for each critical radius the number of **F**-edges decreases until, when $r = \rho(S_i)$, we have two or three **F**-edges. Let us consider r as the time axis. Thus as r increases the number of **F**-edges also increases. For each value of r we need to search the **F**-edges that existed for *that* r. Consider the XY plane, where r is the X axis, and the **F**-edge number is the Y axis. For each **F**-edge e_j we draw a horizontal half-line from (r_j, j) to X=∞, where r_j is the critical radius of e_j.

Any vertical line l passes through $x = r'$ intersects the half lines corresponding to **F**-edges which are on $\partial \mathbf{Q}_S(r')$, and it meets them in the same order the corresponding **F**-edges appear on $\partial \mathbf{Q}_S(r')$.

We use the following modification of the persistent data structure described by Sarnak and Tarjan. Instead of using a red-black tree, we use BB[α] trees, (see Mehlhorn [8]) which are characterized by the following property **consider some internal node in the tree, the number of nodes in each of its subtrees is at most α times the total number of descendants of that node.** Performing n insertions into an empty $BB[\alpha]$ tree implies only n updates of the tree and hence it can be made persistent with $O(n)$ space.

It is easy to see that now that in the course of the execution of Procedure **find** δ_1, δ_2, finding the α-median means just taking the root of the sub-tree containing $I^k_{S_i}$. When using this mechanism for finding λ_1, λ_2, we redefine $I^{(1)}_{S_i}$ such that it contains all the nodes in the smallest subtree containing the critical vertices of $\partial \mathbf{Q}_{S_i}$ on which we perform our search.

5.2 Maintaining the Data Structure after each Insertion

In this subsection we sketch how to keep the number of *DP*s low, while avoiding spending too much time on constructing new *DP*s from old ones. Our method is based on the schemes of Saxe and Bentley [2] as improved by Overmars and van Leeuwen [10] to convert static data structure to dynamic ones.

Let $b = \sqrt{\log n}$, $t = 1 + \lfloor \log_b n \rfloor = O(\log n / \log\log n)$, and $n = \sum_{i=1}^{t} a_i b^{i-1}$ be the expansion of n to base b. We keep the *DP*s so that *DP*, $\mathbf{Q}_{S_i}(\cdot)$ $(i = 1, ..., t)$, corresponds to $a_i b^{i-1}$ points.

A new point is added to the *DP* of size $a_1 < b$. When its size reaches b it is merged with the *DP* of size $a_2 b$. This procedure is cascaded, so that at any time there is only a single *DP* of *rank i* (size $b^i..b^{i+1} - 1$).

Constructing a *DP* of size $a_i b^i$ requires $a_i b^i \log a_i b^i \leq b^{i+1}(i+1)\log b$ time. Since every point can participate in at most $b-1$ *DP*s of rank i, and each such *DP* contains at least b^i points, in the course of the algorithm we construct at most $(b-1)n/b^i < n/b^{i-1}$ *DP*s of rank i. The total time to construct all the *DP*s of rank i is therefore $(n/b^{i-1})b^{i+1}(i+1)\log b$.

Adding this time for all ranks and amortizing the construction cost over all the insertions yields $O(\log^3 n / \log\log n)$ amortized time per insertion.

The technique of Overmars and van Leeuwen enables us to achieve the same time bounds while increasing the number of *DP*s only by a constant. Hence, an insertion takes $O(\log^3 n / \log\log n)$ in the worst case. More details will appear in the full paper.

Acknowledgments

Thanks to Klara Kedem, Seffi Naor, Avigail Orni and Alon Ziv.

References

[1] P. K. Agarwal and M. Sharir. Planar geometric location problems and maintaining the width of a planar set. In *Proc. 2nd ACM-SIAM Sympos. Discrete Algorithms*, pages 449–458, 1991.

[2] J. L. Bentley and J. B. Saxe. Decomposable searching problems I: static-to-dynamic transformation. *J. Algorithms*, 1:301–358, 1980.

[3] D. Eppstein. Dynamic three-dimensional linear programming. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 488–494, 1991.

[4] D. Kirkpatrick and J. Snoeyink. Tentative prune-and-search for computing voronoi vertices. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, 1993.

[5] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 1–8, 1992.

[6] N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. In *Proc. 23rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 329–338, 1982.

[7] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30:852–865, 1983.

[8] K. Mehlhorn. *Sorting and Searching*, volume 1 of *Data Structures and Algorithms*. Springer-Verlag, Heidelberg, West Germany, 1984.

[9] K. Mehlhorn, S. Näher, and V. Priebe. Private correspondence. 1993.

[10] M. H. Overmars and J. van Leeuwen. Dynamization of decomposable searching problems yielding good worst-case bounds. In *Proc. 5th GI Conf. Theoret. Comput. Sci.*, volume 104 of *Lecture Notes in Computer Science*, pages 224–233. Springer-Verlag, 1981.

[11] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction*. Springer-Verlag, New York, NY, 1985.

[12] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29:669–679, 1986.

[13] S. Skyum. A simple algorithm for computing the smallest enclosing circle. *Inform. Process. Lett.*, 37:121–125, 1991.