Finding the Maximal Empty Disk Containing a Query Point*

Haim Kaplan[†] Micha Sharir[‡]

October 16, 2012

Abstract

Let P be a set of n points in the plane. We present an efficient algorithm for preprocessing P, so that, for a given query point q, we can quickly report the largest disk that contains q but its interior is disjoint from P. The storage required by the data structure is $O(n \log n)$, the preprocessing cost is $O(n \log^2 n)$, and a query takes $O(\log^2 n)$ time. We also present an alternative solution with an improved query cost and with slightly worse storage and preprocessing requirements.

1 Introduction

We consider the problem of preprocessing a given set P of n points in the plane into a data structure such that for any given query point q we can efficiently find the largest disk that contains q but its interior does not contain any point of P.

A data structure for finding largest empty disks is needed when one wants to efficiently estimate the size of the "free space" (free of points of P, that is) surrounding a query point and locate the center of this free space. Concrete examples of such a scenario are: (i) We want to transmit a message over the largest possible area (which is a disk if propagation proceeds uniformly in all directions) such that we are sure that q receives the message but none of the eavesdroppers at the points of P hears it. (ii) We want to position a sprinkler to water the largest possible disk that includes our favorite plant at q but does not include any of the locations in P, which are facilities that we do not want to get wet.

Note that it suffices to consider query points q that lie inside the convex hull $CH(P)$ of P because otherwise there are arbitrarily large disks containing q and disjoint from P.

Our results. We present an efficient data structure for largest empty disk queries that requires $O(n \log n)$ storage, $O(n \log^2 n)$ preprocessing time, and can answer a query in

*Work on this paper was partially supported by the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11). Work by Haim Kaplan was also supported by Grant 2006/204 from the U.S.–Israel Binational Science Foundation, and by Grant 822/10 from the Israel Science Fund. Work by Micha Sharir was also supported by NSF Grant CCR-08-30272, by Grant 2006/194 from the U.S.–Israel Binational Science Foundation, by Grant 338/09 from the Israel Science Fund, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

[†]School of Computer Science, Tel Aviv University, Tel Aviv, Israel; email: `haimk@tau.ac.il`

[‡]School of Computer Science, Tel Aviv University, Tel Aviv, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA; email: `michas@tau.ac.il`

$O(\log^2 n)$ time. We also present an alternative data structure that, for any prespecified $\varepsilon > 0$, uses $O(n^{1+\varepsilon})$ storage, requires $O(n^{1+\varepsilon} \log^2 n)$ preprocessing time, and answers a query in $O\left(\frac{1}{\varepsilon} \log n\right)$ time.

Within those latter improved performance bounds, we in fact solve a related problem of independent interest in which the input consists of n arbitrary disks in the plane, and we want to process them into a data structure that allows to efficiently find the largest disk among the input disks containing a query point. A solution to this problem when the disks are the Delaunay disks defined by P turned out to be the bottleneck part in our data structure.

Background. To the best of our knowledge the problem of finding a data structure for largest disks queries was first introduced by Augustine *et al.* [1], who design a data structure that requires $O(n^2)$ space, $O(n^2 \log n)$ preprocessing time, and can answer a query in $O(\log n)$ time. In a more recent version of their work [2], they improve the storage and preprocessing costs of their algorithm, to $O(n^{5/3})$ and $O(n^{5/3} \log n)$, respectively, still retaining the $O(\log n)$ bound for the query cost. They improve the storage and preprocessing costs further, to $O(n^{3/2} \log n)$ and $O(n^{3/2} \log^2 n)$, respectively, at the cost of a slight increase in the query time bound, to $O(\log n \log\log n)$. They also provide in the new version a data structure with near-linear storage and preprocessing, that receives as input a simple polygon P with n edges, and finds, in $O(\log n)$ time, the largest disk containing a query point and lying fully inside P.

A much easier problem, which has been studied much earlier, is that of finding the largest empty disk for a set P of n points in the plane (where, to make sense of the problem, the center of the disk has to lie, say, inside the convex hull of P—recall the remark made earlier concerning points outside the hull). This problem arises in several areas such as data mining, database management, visualization, and VLSI design, and it has a very simple solution. Indeed, the largest empty disk has to be centered at a vertex of the Voronoi diagram of P, or at a crossing of an edge of the diagram with the convex hull boundary, and therefore can be found in $O(n \log n)$ time.

We note that, in their initial study [1], Augustine *et al.* also consider a similar problem, where the goal is to report the largest-area axis-parallel rectangle that contains the query point, is contained in some fixed bounding box B, and its interior is disjoint from P. Their solution for this variant also requires nearly quadratic storage and preprocessing; in a companion paper [5] (see also [4]), we present a more efficient solution, which requires only nearly linear storage and preprocessing, with $O(\log^4 n)$ query time, for that latter problem.

We are not aware of any previous specific work on the problem of preprocessing n disks into a data structure of near-linear size that can find the largest disk containing a query point in logarithmic time. The main structure that we use in this paper for this problem, which is adapted from a similar solution in [1, 2], requires $O(\log^2 n)$ query time. We note that this problem is a special case of a more general range searching setting where ranges have priorities (in our case the size of the disk is its priority) and we want a data structure that can find the range of highest priority containing a query point.

2 Preliminaries

Let P be a set of n points in the plane. We call a disk whose interior is disjoint from P a *P-empty disk*. We wish to preprocess P into a data structure so that, given a query point q, we can efficiently find the largest P-empty disk containing q. The following easy lemma completes the observation made earlier, that it suffices to consider only query points q that lie inside the convex hull $CH(P)$ of P.

Lemma 2.1 *There are arbitrarily large P-empty disks containing q if and only if q does not lie in the interior of the convex hull $CH(P)$ of P.*

Proof. If q does not lie in the interior of $CH(P)$ then there is a line ℓ through q that defines halfplane whose interior is P-empty; clearly there are arbitrarily large P-empty disks in this halfplane tangent to ℓ at q.

For the converse statement, let B be some fixed disk containing $P \cup \{q\}$. Clearly if there are arbitrarily large P-empty disks containing q then there are arbitrarily large P-empty disks containing q on their boundary. For any such disk D, $D \cap B$ contains the intersection of B with a wedge W whose apex is q and whose bounding rays go through the points in $\partial B \cap \partial D$. The wedge W is P-empty ($W \cap B$ is P-empty since it is contained in D and $W \setminus B$ is empty since there are no points of P outside B), and its opening angle tends to π as D grows. In the limit, W becomes a halfplane containing q and openly disjoint from P. Hence q does not lie in the interior of $CH(P)$. $\square$

Let $\text{Vor}(P)$ denote the Voronoi diagram of P.

For a point q in the interior of $CH(P)$, let $D_{\max}(q)$ denote an arbitrary largest P-empty disk containing q, and let $c_{\max}(q)$ denote its center. (In general this disk needs not be unique. For example we may have two congruent Delaunay disks, that is, maximal P-empty disks centered at vertices of $\text{Vor}(P)$, both containing q, and larger than any other P-empty disk containing q. The disk will be unique if we assume that the points of P are in general position.)

The analysis proceeds through the following stages.

Lemma 2.2 *Let q be in the interior of $CH(P)$. Then $c_{\max}(q)$ lies on a Voronoi edge or at a Voronoi vertex.*

Proof. If D is a P-empty disk that contains q and is centered at some point c not on the Voronoi diagram then D touches at most one point of P. If it does not touch any point of P, we can get a larger P-empty disk containing q by expanding D about c until it touches some (unique) point $a \in P$. If D already touches a point a of P then we can get a larger P-empty disk containing q by moving the center of D along the ray from a through c, away from a, keeping it touching a, until it touches another point b of P. This has to occur, for otherwise we get arbitrarily large P-empty disks containing q, contrary to Lemma 2.1. It follows that a disk which is not centered at a Voronoi edge or vertex is not a maximal P-empty disk containing q. $\square$

Suppose that $c_{\max}(q)$ lies in the relative interior of a Voronoi edge e_{ab} defined by two sites $a, b \in P$. There is at least one direction along e_{ab} so that we can increase the size of the disk by moving its center in that direction away from $c_{\max}(q)$ while keeping it P-empty and touching a and b. Since $D_{\max}(q)$ is the largest P-empty disk containing q, we conclude that in this case q must lie on the boundary of $D_{\max}(q)$. Moreover, this boundary is split into two arcs by a and b, and q has to lie on the *smaller* arc; see Figure 1. Note that in this case we have $\angle aqb > \pi/2$. (Note also that ab cannot be a diameter of $D_{\max}(q)$, for otherwise we can expand $D_{\max}(q)$ by moving its center along e_{ab} towards q, keeping it P-empty and containing q.)

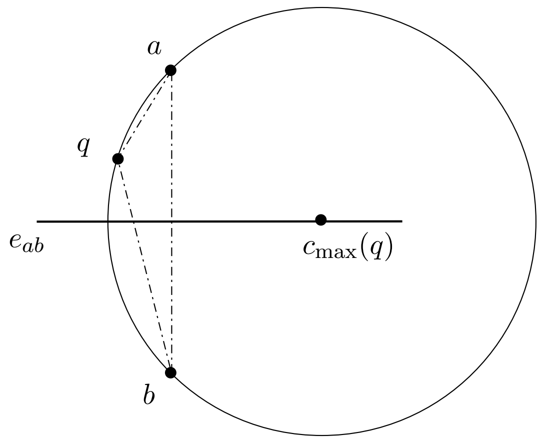

Figure 1: The case where $c_{\max}(q)$ lies in the relative interior of a Voronoi edge.

Continuing the analysis of this scenario, we note that in this case $c_{\max}(q)$ is equally nearest to a, b, and q. That is, $c_{\max}(q)$ is a vertex of the Voronoi cell $V(q)$ of q in the augmented Voronoi diagram $\mathrm{Vor}(P \cup \{q\})$. Then a and b are two consecutive neighbors of q; that is, the Voronoi edges e_{aq}, e_{bq} are consecutive along $\partial V(q)$. The angle between these edges, within $V(q)$, is $\pi - \angle aqb$, and is therefore smaller than $\pi/2$. See Figure 2. Since the sum of the exterior angles of a convex polygon is 2π, it follows that $V(q)$ has at most three vertices that can be the location of $c_{\max}(q)$. We summarize the observations made so far in the following lemma.

Lemma 2.3 *If $c_{\max}(q)$ lies in the relative interior of a Voronoi edge e_{ab} then q lies on the smaller arc connecting a and b along $\partial D_{\max}(q)$. In this case $c_{\max}(q)$ is a vertex of the Voronoi cell $V(q)$ of q in the Voronoi diagram $\mathrm{Vor}(P \cup \{q\})$. Furthermore, $c_{\max}(q)$ is one of the at most three vertices of $V(q)$ with an acute interior angle.*

The preceding analysis therefore suggests the following high-level approach: Identify, without constructing $V(q)$ explicitly (whose complexity can be as high as $\Theta(n)$), the at most three vertices of this cell which might be the location of $c_{\max}(q)$, compute, in $O(1)$ time, the radii of the corresponding largest P-empty disks centered at these vertices, and output the center with the largest radius. This is however only one part of the solution, because we also have to consider Voronoi vertices of $\mathrm{Vor}(P)$ as possible locations for $c_{\max}(q)$.

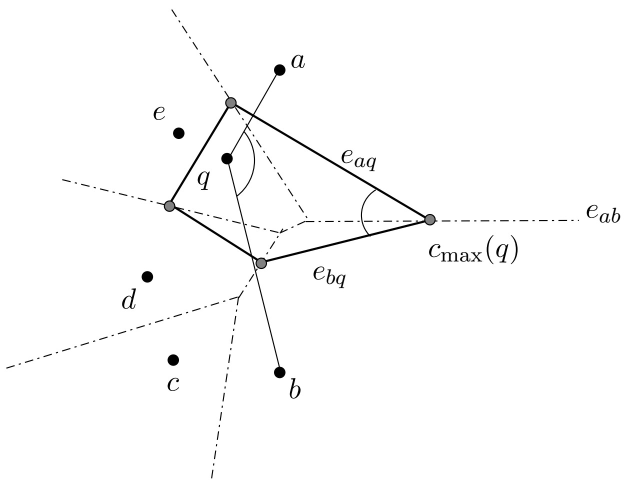

Figure 2: The cell $V(q)$ in $\mathrm{Vor}(P \cup \{q\})$, superimposed on $\mathrm{Vor}(P)$. The angle between e_{aq} and e_{bq} is smaller than $\pi/2$.

2.1 A quick-and-dirty interim solution

Before continuing towards an efficient solution, we can already exploit the analysis given so far, to obtain the following quick-and-dirty solution. Consider a query point q, the cell $V(q)$ of q in $\mathrm{Vor}(P \cup \{q\})$, and its at most three special vertices having acute interior angles, which are the candidate placements for $c_{\max}(q)$ (other than Voronoi vertices of $\mathrm{Vor}(P)$). If we move q, $V(q)$ will vary continuously, and its combinatorial structure (namely, the sequence of its edges and the identity of the three acute vertices) will remain fixed until q reaches some critical placement in which one of these attributes changes. It is easy to verify that the sequence of edges of $V(q)$ can change only when one of its vertices is a vertex of the non-augmented diagram $\mathrm{Vor}(P)$; that is, when q lies on the boundary of the Delaunay disk which is centered at that vertex (and circumscribes its dual Delaunay triangle). The "acuteness" of a vertex of $V(q)$, lying on an original Voronoi edge e_{ab}, can change only when q lies on the boundary of the diametral disk determined by a and b, but only if this disk is P-empty. (Note that the diametral disk is P-empty if and only if ab is an edge of the *Gabriel graph* of P.)

So a quick-and-dirty solution is to draw the $O(n)$ disks of the above two types (Delaunay disks and diametral disks determined by edges of the Gabriel graph), form their arrangement $\mathcal{A}$, and store with each face f of $\mathcal{A}$ the list $L(f)$ of the at most three acute vertices of $V(q)$, for any $q \in f$ (by the preceding argument, the list is the same for all $q \in f$). We also store with each f the largest Delaunay disk, $D(f)$, centered at a vertex of $\mathrm{Vor}(P)$, which contains f. We can compute the lists $L(f)$ and the disks $D(f)$ incrementally, observing that it is straightforward to update $L(f)$ in $O(1)$ time as we cross from one face f to a neighboring face f'. Updating $D(f)$ can be done in $O(\log n)$ time, by dynamically maintaining the list of all Delaunay disks containing the present face f, sorted in the decreasing order of their radii. We then preprocess $\mathcal{A}$ for efficient point location, and obtain the desired data structure. It requires $O(n^2)$ storage and $O(n^2 \log n)$ preprocessing, and a query takes $O(\log n)$ time: Locate the query point q in $\mathcal{A}$, retrieve the list $L(f)$ of the face f containing q, and search each of its (at most three) elements for a possible location of $c_{\max}(q)$. We return the largest among the maximal P-empty disks centered at the vertices defined by $L(f)$ and the disk $D(f)$. Note that the elements of $L(f)$ are only stored symbolically, because the actual vertices of $V(q)$ vary continuously as q moves inside f; still, the combinatorial identification of each of these elements allows us to test it in $O(1)$ time for any concrete query point q in f.

This straightforward approach matches the performance of the algorithm in [1] (but not its improvement in [2]).

3 A more efficient data structure

3.1 Efficient processing of Delaunay disks

It is relatively easy to improve the algorithm that finds the largest Delaunay disk containing q, using the approach given in [1]. That is, we sort the Voronoi vertices v of $\mathrm{Vor}(P)$ by the radius r_v of the Delaunay disk centered at v, and store them, in this inorder, at the leaves of a balanced binary tree T. For each node w of T, let Q_w denote the subset of vertices stored at the leaves of the subtree rooted at w, and let R_w denote the set of the corresponding radii r_v. Construct the union U_w of all the Delaunay disks centered at the points of Q_w, and store it at w. U_w has $O(|Q_w|)$ edges and vertices [6] and can be constructed in $O(|Q_w| \log |Q_w|)$ time (e.g., by lifting the disks to planes in three dimensions and by constructing the upper envelope of these planes), for a total of $O(n \log n)$ storage and $O(n \log^2 n)$ preprocessing time. We process each U_w for fast (logarithmic time) point location.

Now, given a query point q, we search with it through T. We first test, in $O(\log n)$ time, whether $q \in U_{\mathrm{root}}$. If not, we stop and report that q is not contained in any Delaunay disk. Otherwise, at each node w that we visit (starting from the root), we take its right child w_r and check whether $q \in U_{w_r}$. If so, then q is contained in at least one Delaunay disk whose radius belongs to R_{w_r}, and we continue the search at w_r. If $q \notin U_{w_r}$, we continue the search at the left child w_l of w. When the search terminates at some leaf, the disk stored at that leaf is the largest Delaunay disk containing q. The search takes a total of $O(\log^2 n)$ time.

Remark. It is interesting to note that this is the most expensive part of the query procedure, as the other part, described below, takes only $O(\log n)$ time. As a matter of fact, the problem at hand is an instance of the more general problem where we are given n disks in the plane, and wish to preprocess them into a data structure of near-linear size, so that we can report, in *logarithmic* time, the largest disk containing a query point. We provide in Section 4 an algorithm that makes some progress towards this goal, but so far we do not know how to fully accomplish this.

3.2 Disks centered at Voronoi edges

We next consider the other situation, where $c_{\max}(q)$ lies in the relative interior of some Voronoi edge. Let e_{ab} be a Voronoi edge, determined by two points $a, b \in P$ and delimited by the Voronoi vertices v_{abc}, v_{abd}, for two additional points $c, d \in P$, so that v_{abc} (resp., v_{abd}) is equally nearest to a, b, c (resp., to a, b, d). Let D_{ab} denote the diametral disk on ab, and let D_{abc} (resp., D_{abd}) denote the Delaunay disk centered at v_{abc} (resp., v_{abd}), so it circumscribes Δabc (resp., Δabd).

Lemma 3.1 *Any point q for which $c_{\max}(q)$ lies in the relative interior of e_{ab} must lie in*

$$K_{ab} = int\,(D_{ab} \setminus (D_{abc} \cap D_{abd}))\ .$$

Proof. Indeed, we have already argued that such a point q must satisfy $\angle aqb > \pi/2$, which is equivalent to q belonging to the interior of D_{ab}. In addition, the center of the disk circumscribing Δaqb lies in the relative interior of e_{ab} if and only if q lies in the interior of $(D_{abc} \cup D_{abd}) \setminus (D_{abc} \cap D_{abd})$, as is easily verified. The conjunction of these two conditions is $q \in K_{ab}$. $\square$

The case where ab is an edge of the *Gabriel graph* is shown in Figure 3. In this case ab crosses e_{ab} and D_{ab} is P-empty. We refer to such an edge in short as a *Gabriel edge*. In this case K_{ab} is the union of the interiors of the two shaded lunes.

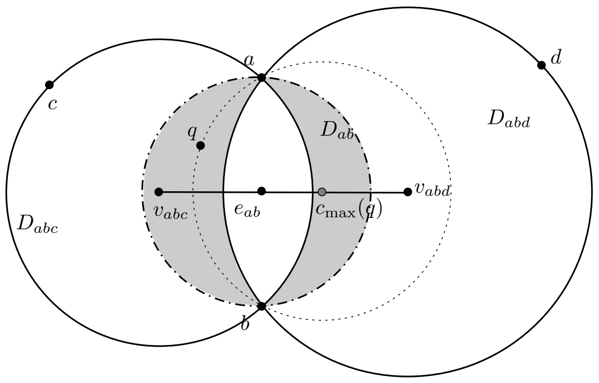

Figure 3: q has to lie in the interior of one of the shaded lunes if $c_{\max}(q)$ lies in the relative interior of the Voronoi edge e_{ab}; the case where ab is an edge of the Gabriel graph. The corresponding disk $D_{\max}(q)$ is shown dotted.

The case where the Delaunay edge ab does not belong to the Gabriel graph (a *non-Gabriel edge*) is shown in Figure 4. Here K_{ab} is the single shaded lune.

Note that for each lune, in both cases of Gabriel edges and non-Gabriel edges, the area enclosed by the "outer" (longer) arc of the lune and the edge ab is P-empty.

We construct the regions K_{ab}, for all Voronoi edges e_{ab} of $\text{Vor}(P)$, and denote the collection of all these $O(n)$ lunes and double lunes as $\mathcal{K}$. A crucial property is that no point q can be contained in more than three regions of $\mathcal{K}$. Indeed, a point q belongs to K_{ab} if and only if the disk circumscribing Δaqb is P-empty, its center lie in the relative interior of the Voronoi edge e_{ab}, and $\angle aqb > \pi/2$. The claim then follows from Lemma 2.3.

Ideally, we could construct the arrangement $\mathcal{A}(\mathcal{K})$, preprocess it for fast point location, and store with each face f of $\mathcal{A}(\mathcal{K})$ the at most three Delaunay edges ab whose regions K_{ab} (fully) contain f. Then, given a query point q, we could locate q in the arrangement, retrieve the at most three corresponding Delaunay edges, and test, in constant time, each of the respective Voronoi edges for possible location of $c_{\max}(q)$. The problem with this approach is that the complexity of $\mathcal{A}(\mathcal{K})$ might be quadratic, as depicted in Figure 5. (Note that the arrangement $\mathcal{A}$ constructed in Section 2.1 is in fact a refinement of $\mathcal{A}(\mathcal{K})$.)

We overcome this problem by first decomposing $\mathcal{K}$ and its lunes into a constant number of subcollections, each consisting of portions of the lunes, so that, within each subproblem

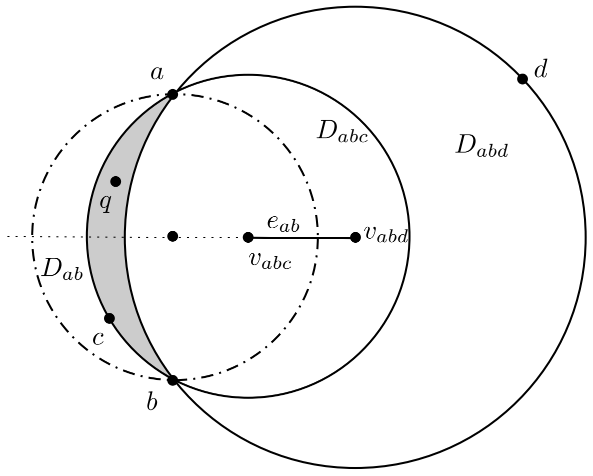

Figure 4: q has to lie in the interior of the shaded lune if $c_{\max}(q)$ lies in the relative interior of the Voronoi edge e_{ab}; the case where ab is not an edge of the Gabriel graph.

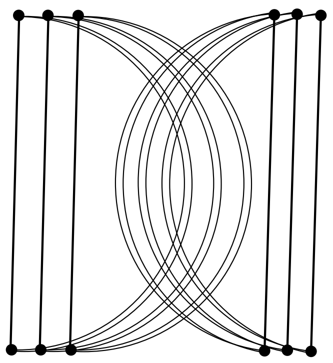

Figure 5: Illustrating a scenario in which the arrangement of the lunes K_{ab} has quadratic complexity.

that we create, the regions are *pairwise openly disjoint*, and so the planar map that they form has linear complexity. Overall, we obtain a structure that uses only linear storage, and the query time, which is dominated by the point location in these maps, is only $O(\log n)$. (The overall query time, though, is $O(\log^2 n)$, because of the search for the largest Delaunnay disk containing q; see Sections 3.1 and 4.)

The intersection pattern of the lunes The following simple lemma provides the main technical tool for decomposing the lunes.

Lemma 3.2 *Let K_{ab} and K_{cd} be two distinct lunes (the four points a, b, c, d need not all be distinct) with a nonempty intersection. Then, for each point $q \in K_{ab} \cap K_{cd}$, there exists a line ℓ through q that (weakly) separates a, b from c, d. That is, one closed halfplane bounded by ℓ contains a and b, and the other closed halfplane contains c and d.*

Proof. Refer to Figure 6. Let D_{abq} (resp., D_{cdq}) denote the disk circumscribing the triangle Δabq (resp., Δcdq). By the properties of the lunes, as analyzed in Lemma 3.1, both disks are P-empty. The circle C_{abq} bounding D_{abq} and the circle C_{cdq} bounding D_{cdq} meet at q and therefore either (i) they also intersect at a second point q', or (ii) they are tangent to each other at q. Let ℓ be the line connecting q and q' in case (i) or the common tangent line at q in case (ii). As is easily checked, ℓ separates the arcs $\alpha = C_{abq} \setminus D_{cdq}$ and $\gamma = C_{cdq} \setminus D_{abq}$. Moreover, since D_{abq} and D_{cdq} are both P-empty, a and b must lie in the closure of α and c and d must lie in the closure of γ, thereby completing the proof of the lemma. $\square$

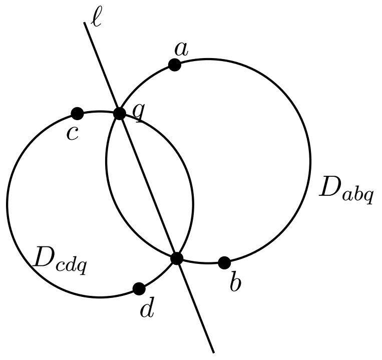

Figure 6: Illustrating the proof of Lemma 3.2.

3.3 Partitioning the lunes

Having these observations at hand, we now proceed to eliminate all intersections between the lunes by an appropriate problem decomposition, as promised above. Rotate the coordinate frame so that no two points have the same x- or y-coordinate. We first partition $\mathcal{K}$ into two subcollections $\mathcal{K}^+$ and $\mathcal{K}^-$, where $\mathcal{K}^+$ (resp., $\mathcal{K}^-$) consists of all the lunes which lie above (resp., below) the lines containing their "bases" (i.e., the corresponding Delaunay edges). We describe the processing of $\mathcal{K}^+$; processing $\mathcal{K}^-$ is done in a fully symmetric manner. We refer to the lunes in $\mathcal{K}^+$ as *upper lunes*.

We next distinguish between lunes in $\mathcal{K}^+$ whose bases have positive slope and those whose bases have negative slope, and treat each subcollection separately. We will only consider the first subcollection, since the other one is handled in a fully symmetric manner, and, for simplicity of notation, we will continue to denote it as $\mathcal{K}^+$.

Let K_{ab} be one of these upper lunes, with, say, a lying to the left of b (and with ab having positive slope). We note that some portion of K_{ab} might extend to the left of the vertical line through a and that K_{ab} lies fully to the left of the vertical line through b (this latter property holds because K_{ab} is always contained in the diametral disk of ab). As is easy to check, K_{ab} always extends to the left if ab is a Gabriel edge, but it does not have to do so if ab is a non-Gabriel edge.

We form from $\mathcal{K}^+$ two new collections. We take each upper lune K_{ab} in $\mathcal{K}^+$, with a to the left of (and below) b, and split it into (at most) two open regions by drawing a vertical line through a. (The union of the regions is K_{ab} without its intersection segment with the vertical line through a.) One region, denoted K_{ab}^T, lies vertically above ab, and the other, denoted K_{ab}^L, lies fully to the left of a. As just noted, K_{ab}^L might be empty. We denote by $\mathcal{K}^T$ (resp., $\mathcal{K}^L$) the collection of all the top portions K_{ab}^T (resp., left portions K_{ab}^L) of the lunes of $\mathcal{K}^+$. See Figure 7 for an illustration.

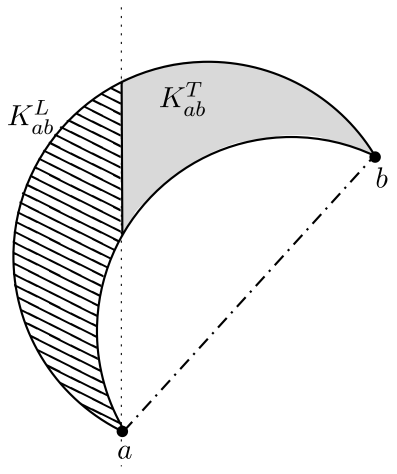

Figure 7: Partitioning an upper lune (with a base ab of positive slope) into a top portion and a left portion.

Lemma 3.3 *The regions of $\mathcal{K}^T$ are pairwise disjoint.*

Proof. Suppose to the contrary that there exist two distinct top regions K_{ab}^T and K_{cd}^T in $\mathcal{K}^T$, with a to the left of b and c to the left of d, such that K_{ab}^T and K_{cd}^T have a nonempty intersection, and let q be a point in both regions. By Lemma 3.2 there exists a line ℓ through q that weakly separates a and b from c and d. This however is impossible because q lies vertically above both segments ab and cd. See Figure 8. $\square$

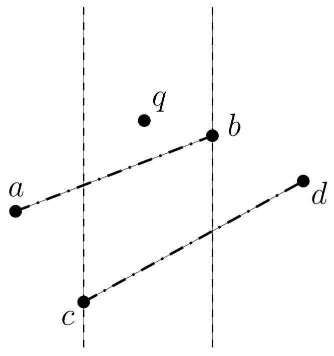

Figure 8: Illustrating the proof of Lemma 3.3.

Lemma 3.4 *The regions of $\mathcal{K}^L$ are pairwise disjoint.*

Proof. Suppose to the contrary that there exist two distinct left regions K_{ab}^L and K_{cd}^L in $\mathcal{K}^L$, with a to the left of b and c to the left of d, such that K_{ab}^L and K_{cd}^L have a nonempty intersection, and let $q \in K^L_{ab} \cap K^L_{cd}$. By Lemma 3.2 there exists a line ℓ through q that weakly separates a and b from c and d. By construction, the two segments ab and cd lie to the right of the vertical line λ through q; one of them, say, ab lies in the upper-right quadrant Q^+ formed by ℓ and λ, and the other, cd, lies in the lower-right quadrant Q^-. See Figure 9. Suppose first that the angle at q of Q^+ is at most $\pi/2$ (Figure 9(left)). Then q (which is different from a) is lower in the y-direction than a. This however is impossible since $q \in K^L_{ab}$, which is contained in the upper portion of the diametral disk of ab, which lies fully above a. Otherwise, the angle at q of Q^- is acute. This too is impossible because K^L_{cd} is fully contained in the interior of the diametral disk of cd. These contradictions establish the assertion of the lemma. □

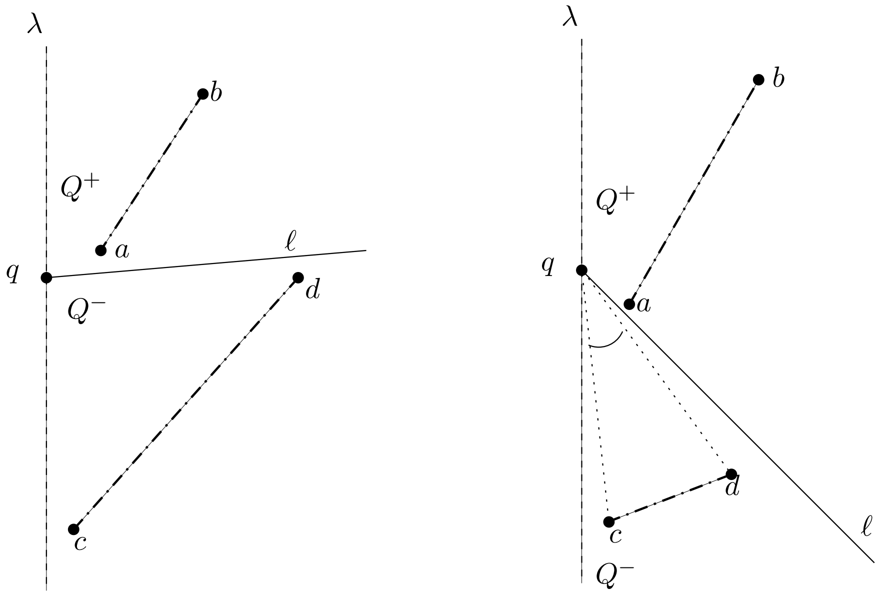

Figure 9: Illustrating the proof of Lemma 3.4. Left: The angle at q of the quadrant Q^+ containing a and b is at most $\pi/2$. Right: The angle at q of the quadrant Q^- containing c and d is acute.

3.4 The data structure

We thus obtain the following algorithm. Partition $\mathcal{K}$ into the four subcollections of upper lunes with positive-slope bases, upper lunes with negative-slope bases, lower lunes with positive-slope bases, and lower lunes with negative-slope bases. Each of these subcollections is processed in a similar manner, and we describe only the processing of the first subcollection, which, as above, we denote as $\mathcal{K}^+$.

We decompose each lune K in $\mathcal{K}^+$ into its top subregion K^T and its left subregion K^L, by the recipe given above, and let $\mathcal{K}^T$ (resp., $\mathcal{K}^L$) denote the collection of the resulting top (resp., left) subregions. By Lemmas 3.3 and 3.4, each collection consists of pairwise disjoint regions, and so forms a planar map of linear complexity (so that each subregion in the collection is a single separate face of the map). We process each of the two resulting maps for fast (logarithmic time) point location queries. Altogether we obtain eight such maps.

Now, given a query point q, we locate it in each of these eight maps, and retrieve the at most three lunes that contain q. Each such lune yields a candidate maximal empty disk containing q and centered at the respective Voronoi edge. (If q happens to lie on a vertical edge of a map, which bounds some region K_{ab}^T or K_{ab}^L, then the Voronoi edge e_{ab} is one of the three candidate edges.) This part of the data structure requires $O(n)$ space, takes $O(n \log n)$ time to construct and querying it takes $O(\log n)$ time. We then find the largest empty Delaunay disk (centered at a Voronoi vertex) containing q, as described in Section 3.1, and output the largest of the at the four candidate disks. The overall complexity is dominated by the costs of handling Delaunay disks.

We thus obtain the main result of this paper.

Theorem 3.5 *One can preprocess a set P of n points in the plane in $O(n \log^2 n)$ time into a data structure of size $O(n \log n)$, so that, given a query point q (in the interior of the convex hull of P), we can compute, in $O(\log^2 n)$ time, the largest P-empty disk containing q.*

The largest P-empty disk containing k query points. In view of the applications discussed in the introduction, a natural interesting generalization of the problem studied in this paper is to handle queries, each consisting of a collection Q of k points, where, say, k is some constant, and the goal is to find the largest disk containing Q (or, rather, its convex hull $CH(Q)$) whose interior is disjoint from P. (Note that when $|Q| > 1$ the problem does not always have a solution.) As far as we know, it is an open problem whether this can be done with near-linear storage, and with query time that depends polylogarithmically on n (and polynomially on k). Some observations concerning this problem are given next.

To answer such a query with a set Q we observe that the largest P-empty disk D containing Q (if it exists) must contain on its boundary either three points of P, in which case it is a Delaunay disk, or two points of P and one point of Q. Indeed, as is easy to see, in all other cases there is a larger P-empty disk containing Q. Finding the largest Delaunay disk containing a set of two or more points is harder than the case of a single query point since we cannot use the binary search procedure in the arrangements of unions of disks described in Section 3.1.

We can determine whether there exists a P-empty disk containing Q in $O(k \log n)$ time with an appropriate preprocessing of P, as follows. We lift the points of P and the points of Q to the paraboloid $z = x^2 + y^2$ in $\mathbb{R}^3$. (For simplicity, we refer to the lifted sets of points also as P and Q, respectively.) In this representation a P-empty disk containing Q corresponds to a hyperplane separating the convex hull $CH(P)$ of P and the convex hull $CH(Q)$ of Q. Such a hyperplane exists if and only if $CH(Q)$ is disjoint from $CH(P)$. It is not difficult to show, using the technique of Dobkin and Kirkpatrick [3] that one can preprocess P in $O(n \log n)$ time, into a linear-size data structure, such that, given a query set Q, we can determine whether $CH(Q)$ and $CH(P)$ are disjoint, in $O(k \log n)$ time.

Briefly and informally, we note that neither of $CH(Q)$ and $CH(P)$ fully contains the other hull, because they are hulls of points on the paraboloid (unless $Q \subseteq P$, in which case no such disk exists). It follows that they intersect if and only if either an edge of $CH(Q)$ intersects a face of $CH(P)$ or (an edge of) $CH(P)$ intersects a face of $CH(Q)$. Intersection of edges of $CH(Q)$ with $CH(P)$ can be detected in $O(k \log n)$ time using the technique of [3]. Once we have determined that no edge of $CH(Q)$ meets $CH(P)$, we test, for each face

f of $CH(Q)$, whether its supporting plane π_f meets $CH(P)$. (This can also be easily done in a total of $O(k \log n)$ time.) If no such intersection is found, f is disjoint from $CH(P)$. Otherwise we obtain a witness point $w \in \pi_f \cap CH(P)$, and we test whether $w \in f$ (and then f intersects $CH(P)$), or not (and then there is no intersection). The latter properties follow because we already know that no edge of Q intersects $CH(P)$.

We also note that finding the largest P-empty disk whose boundary contains two points of P and one of Q is rather easy to do. We simply query the structure presented at Sections 3.2–3.4 with each point of Q separately, collect the at most $3k$ candidate disks, filter out those that do not fully contain Q, and return the largest among the surviving ones. With an appropriate implementation this takes a total of $O(k \log n + k \log k)$ time.

We close this discussion by noting that the problem can be solved with fast query time as specified above, provided one is allowed near-quadratic storage, using standard range searching techniques.

4 Finding the largest (Delaunay) disk containing a query point

As discussed in a remark at the end of Section 3.1, an interesting open problem is to reduce the query time to $O(\log n)$ without increasing significantly the storage and preprocessing costs. As noted, the bottleneck in the query cost is at the stage that finds the largest Delaunay disk that contains q (see Section 3.1).

While we still do not know how to achieve this goal, we provide in this section a partial solution that may be of independent interest. As a matter of fact, we study a more general problem, involving a collection of n arbitrary disks, rather than Delaunay disks of some input point set. We do not know whether the Delaunayhood of the disks makes the problem any easier to solve.

So let $\mathcal{D}$ be a collection of n arbitrary disks; assume for simplicity that all their radii are distinct. Let $M(\mathcal{D})$ be the planar map, each of whose 2-dimensional faces is a maximal connected region f with the property that all its points have the same largest disk $D(f)$ containing them, or none of them is contained in any disk (in which case f is a face of the complement of the union of the disks). We label each face f by the corresponding largest disk $D(f)$ (or by a flag indicating that f lies outside the union). A procedural definition of $M(\mathcal{D})$ goes as follows. Sort the disks of $\mathcal{D}$ in decreasing order of their radii, and enumerate the resulting sequence as $D_1, D_2, \ldots, D_n$. Insert the disks one by one in this order and maintain the map $M(\mathcal{D}_i)$ for the prefix $\mathcal{D}_i$ of the first i inserted disks. When D_i is inserted, we add to $M(\mathcal{D}_{i-1})$ the region $D_i \setminus \bigcup \mathcal{D}_{i-1}$. The "filled" portion of $M(\mathcal{D}_{i-1})$ (that is, the union $\bigcup \mathcal{D}_{i-1}$ of $\mathcal{D}_{i-1}$) is thus not touched, and is augmented by the connected components of the added region.

Once we have $M(\mathcal{D})$ available, we preprocess it for fast point location, and then we are done. Given a query point q, we locate q is $M(\mathcal{D})$ and output the disk $D(f)$ of the face f containing q (or report that q does not lie in any disk). The query time is $O(\log n)$, but the "catch" is that the worst case complexity of $M(\mathcal{D})$ is $\Theta(n^2)$; an example is depicted in Figure 10. The following lemma provides an important special case where the complexity is only linear.

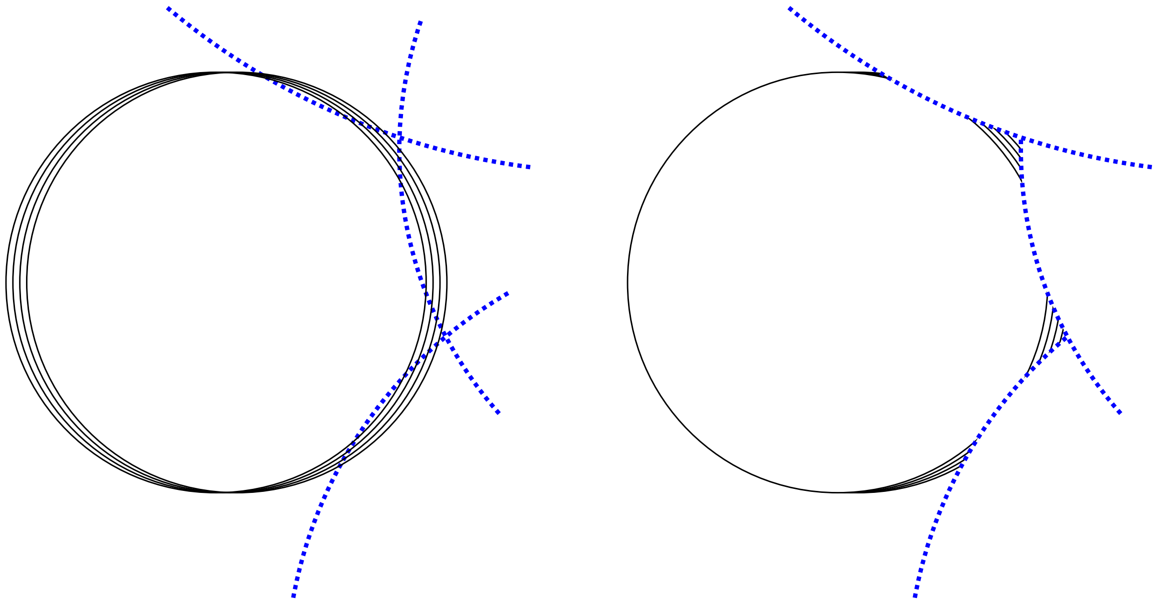

Figure 10: Left: A collection $\mathcal{D}$ of n disks such that $M(\mathcal{D})$ has quadratic complexity. There are $n/2$ solid disks $\{D_1, \ldots, \mathcal{D}_{n/2}\}$ such that D_{i+1} is slightly smaller than D_i and its center is slightly to the right of D_i. There are also $n/2$ dotted disks which are much larger than the solid disks. Each dotted disk D_j intersects each solid disk D_i in a narrow lune D_{ij}. Lunes of different dotted disks are disjoint, that is $D_{i_1 j_1} \cap D_{i_2 j_2} = \emptyset$ for every $j_1 \neq j_2$. Lunes of the same dotted disk are nested, that is $D_{i_1 j} \subset D_{i_2} j$ if $i_2 > i_1$. Right: This is $M(\mathcal{D})$, assuming the sizes of the dotted disks are decreasing from top to bottom.

Lemma 4.1 *Let $\mathcal{D} = \{D_1, \ldots, \mathcal{D}_n\}$ be a collection of disks, sorted in decreasing order by radii, all centered below the x-axis. Then the portion of $M(\mathcal{D})$ above the x-axis has linear complexity.*

Proof. Denote this portion of $M(\mathcal{D})$ by $M^+(\mathcal{D})$, and consider the step that inserts a disk D_i to $M^+(\mathcal{D}_{i-1})$; that is, we add the (portion above the x-axis of the) region $D_i \setminus \bigcup \mathcal{D}_{i-1}$ to the (portion above the x-axis of) $\bigcup \mathcal{D}_{i-1}$. We claim that ∂D_i contributes only (at most) one arc to the new map. Indeed, any such arc γ is a maximal arc of ∂D_i that lies on the boundary of the union $\bigcup \mathcal{D}_i$. Let a be an endpoint of γ, let D_j be the (larger) disk into which ∂D_i enters past a, and let b denote the other intersection point of ∂D_i and ∂D_j. Let c_i and c_j denote the respective centers of D_i and D_j. Note that a and b are symmetric with respect to the line containing $c_i c_j$. Suppose first that the segment $c_i c_j$ intersects ab; see Figure 11(a). Since both c_i and c_j lie below the x-axis and a lies above it, b must lie below it. That is, the portion of ∂D_i on the other side of a reaches the x-axis before it exits the union, so it cannot contribute any arc to the union boundary. Suppose then that $c_i c_j$ does not intersect ab; see Figure 11(b). In this case $\partial D_i \setminus D_j$ is smaller than a semicircle, as is easily verified, and the preceding argument applies in this case too. Repeating the argument to the other endpoint of γ, we conclude that γ is the only arc of ∂D_i appearing in $M^+(\mathcal{D}_i)$, and this is easily seen to complete the proof of the lemma. (The added region $D_i \setminus \bigcup \mathcal{D}_{i-1}$ may also consist of faces that do not touch ∂D_i, but this does not add any new features to $M(\mathcal{D}_{i-1})$; it simply "fills" already existing holes of the preceding union, making them parts of the new union.) □

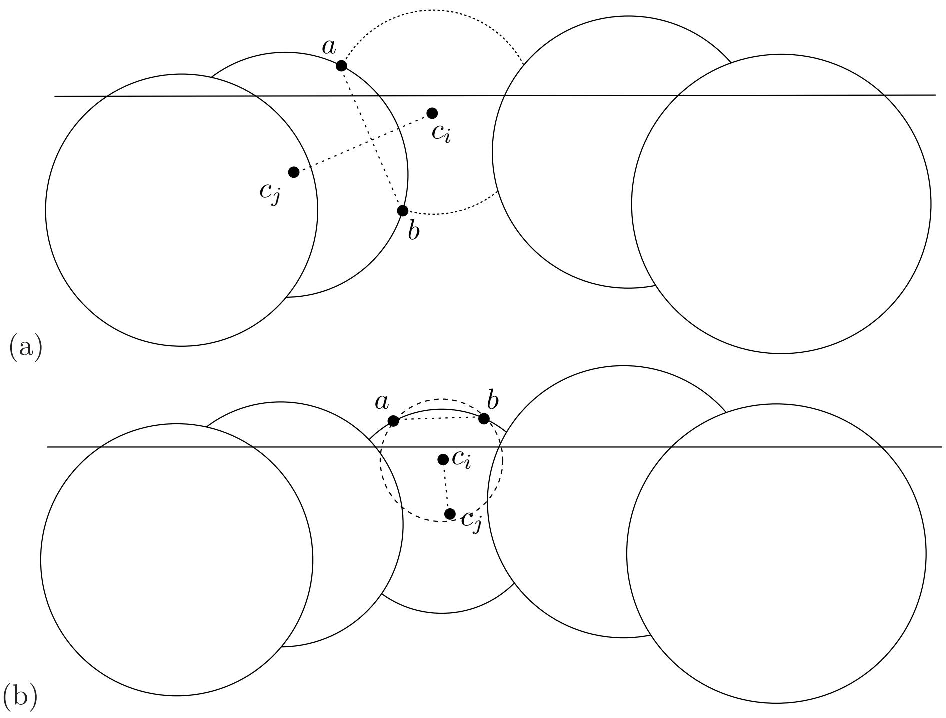

Figure 11: An illustration of the proof of Lemma 4.1.

$M^+(\mathcal{D})$ can be constructed using the following standard divide-and-conquer technique. Partition $\mathcal{D}$ into the subsets $\mathcal{D}_s$ and $\mathcal{D}_b$, where $\mathcal{D}_s$ consists of the $n/2$ smallest disks of $\mathcal{D}$, and $\mathcal{D}_b$ of the remaining larger disks. Construct $M^+(\mathcal{D}_s)$ and $M^+(\mathcal{D}_b)$ recursively, and extract from $M^+(\mathcal{D}_b)$ only the boundary of the union of these disks (within the appropriate halfplane), denoted as U_b. Now run a sweep-based algorithm for merging the maps $M^+(\mathcal{D}_s)$ and U_b. It is easy to verify that any intersection point between arcs of these two maps must be a vertex of $M^+(\mathcal{D})$, so there are only $O(n)$ such points. Once the sweep is over, we prune away the portion of $M^+(\mathcal{D}_s)$ that lies inside U_b, and "glue" the remaining portion of $M^+(\mathcal{D}_s)$ to $M^+(\mathcal{D}_b)$, thereby obtaining $M^+(\mathcal{D})$. It is straightforward to verify that the time it takes to construct $M^+(\mathcal{D})$ using this simple divide-and-conquer algorithm is $O(n \log^2 n)$.

The linear complexity of $M^+(\mathcal{D})$ and its fast construction algorithm described above, suggest the following simple recursive data structure for maintaining an implicit representation of $M(\mathcal{D})$. Sort the disks in increasing order of the y-coordinates of their centers. Fix some parameter r, whose value will be specified later, and partition the resulting sequence of disks into r blocks, each consisting of n/r disks (we ignore rounding issues for the sake of simplicity). Denote the blocks as $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_r$. For each $i = 1, \ldots, r-1$, let λ_i denote some horizontal line that separates the centers of the disks in $\mathcal{B}_i$ from the centers of the disks in $\mathcal{B}_{i+1}$. For each $i = 1, \ldots, r$, let σ_i denote the horizontal strip between λ_{i-1} and λ_i. For $i = 0$ (resp., $i = r$) this is the halfplane below λ_1 (resp., above λ_{r-1}). For each $i = 1, \ldots, r$, put $\mathcal{B}_i^- = \mathcal{B}_1 \cup \cdots \cup \mathcal{B}_{i-1}$ and $\mathcal{B}_i^+ = \mathcal{B}_{i+1} \cup \cdots \cup \mathcal{B}_r$. Let $M^+(\mathcal{B}_i^-)$ denote the portion of $M(\mathcal{B}_i^-)$ that lies above λ_{i-1}, and let $M^-(\mathcal{B}_i^+)$ denote the portion of $M(\mathcal{B}_i^+)$ that lies below λ_i. Lemma 4.1 and its symmetric version (in which we flip the direction of the y-axis) imply that the complexity of each of the maps $M^+(\mathcal{B}_i^-)$, $M^-(\mathcal{B}_i^+)$, for $i = 1, \ldots, r$, is $O(n)$; the overall complexity of all these maps is thus $O(nr)$. We can construct each of these maps, by a divide-and-conquer scheme as described above, in $O(n \log^2 n)$ time, for a total of $O(nr \log^2 n)$ time. This running time subsumes also the preprocessing cost of each of these maps for fast point location.

In addition, we construct a recursive version of the data structure for each of the sets $\mathcal{B}_i$, using the same value of r at each recursive level. The recursion bottoms out when $n \le r$, in which case we simply construct $M(\mathcal{D})$, using $O(r^2)$ storage and $O(r^2 \log r)$ preprocessing. (Here we can afford to construct the full arrangement of the disks, and then extract from it the relevant edges of $M(\mathcal{D})$.)

A query with a point q is performed as follows. We first locate, in $O(\log r)$ time, the horizontal strip σ_i containing q. We then locate q in the maps $M^+(\mathcal{B}_i^-)$ and $M^-(\mathcal{B}_i^+)$, and retrieve the (at most) two corresponding largest disks D^-, D^+ containing q. We then search with q in the recursive data structure for $\mathcal{D}_i$, and output the largest of D^-, D^+, and the disk returned by the recursive call. The cost of a query is $O(k \log n)$, where k is the recursion depth.

Assume that the storage needed for a map $M^+(\mathcal{D})$ or $M^-(\mathcal{D})$ is at most $c|\mathcal{D}|$, for some absolute constant c. Then the maximum storage $S(n)$ of the structure for a set of n disks satisfies the recurrence

$$S(n) \le 2crn + rS(n/r),$$

for $n \ge r$, and $S(n) = O(r^2)$ for $n \le r$. Again, if the recursion has depth k, the overall storage cost is

$$S(n) \le 2crkn + (n/r) \cdot O(r^2) = O(rkn).$$

A similar recursion shows that the preprocessing time is $O(rkn \log^2 n)$. Let $\varepsilon > 0$ be given. We choose $r = n^\varepsilon$ (the same r at all recursive levels, with n being the initial input size), and observe that the recursion then has depth $k = 1/\varepsilon$. With this choice we thus obtain the following result.

Theorem 4.2 *(a) Given a collection $\mathcal{D}$ of n arbitrary disks in the plane, and a parameter $\varepsilon > 0$, we can preprocess $\mathcal{D}$ into a data structure of size $O(n^{1+\varepsilon})$, in $O(n^{1+\varepsilon} \log^2 n)$ time, so that, for any query point q, we can find the largest disk of $\mathcal{D}$ containing q in $O\left(\frac{1}{\varepsilon} \log n\right)$ time.*
(b) Given a set P of n points in the plane, and a parameter $\varepsilon > 0$, we can preprocess P into a data structure of size $O(n^{1+\varepsilon})$, in $O(n^{1+\varepsilon} \log^2 n)$ time, so that, for any query point q, we can find the largest P-empty disk containing q in $O\left(\frac{1}{\varepsilon} \log n\right)$ time.

The main open problem raised by the study in this section is to develop further improved algorithms for the largest disk problem.

Acknowledgments. The authors would like to thank Günter Rote and Joe Mitchell for helpful discussions concerning the problem studied in this paper. In particular, the analysis in Section 4 was triggered by constructive comments that they have made.

References

[1] J. Augustine, S. Das, A. Maheshwari, S. C. Nandy, S. Roy, and S. Sarvattomananda, Recognizing the largest empty circle and axis-parallel rectangle in a desired location, in `arXiv.org:1004.0558`, 2010.

[2] J. Augustine, S. Das, A. Maheshwari, S. C. Nandy, S. Roy, and S. Sarvattomananda, Localized geometric query problems, in `arXiv.org:1111.2918v2`, 2012.

[3] D. P. Dobkin and D. G. Kirkpatrick, Fast detection of polyhedral intersection, *Theoret. Comput. Sci.* 27 (1983), 241–253.

[4] H. Kaplan, S. Mozes, Y. Nussbaum and M. Sharir, Submatrix maximum queries in Monge matrices and Monge partial matrices and their applications, *Proc.* 23^{rd} *ACM-SIAM Annu. Sympos. Discrete Algorithms*, 2012, 338–355.

[5] H. Kaplan and M. Sharir, Finding the maximal empty rectangle containing a query point, in `arXiv.org:1106.3628`, 2011.

[6] K. Kedem, R. Livne, J. Pach and M. Sharir, On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles, *Discrete Comput. Geom.* 1 (1986), 59–71.