Circle Separability Queries in Logarithmic Time

Greg Aloupis* † Luis Barba* Stefan Langerman*‡

Abstract

In this paper we preprocess a set P of n points so that we can answer queries of the following form: Given a convex m-gon Q, report the minimum circle containing P and excluding Q. Our data structure can be constructed in $O(n \log n)$ time using $O(n)$ space, and answers queries in $O(\log n + \log m)$ time.

1 Introduction

The planar separability problem consists of constructing, if possible, a boundary that separates the plane into two components such that two given sets of geometric objects become isolated. Typically this boundary is a single curve such as a line, circle or simple polygon, meaning that each component of the plane is connected. Probably the most classic instance of this problem is to separate two given point sets with a circle (or a line, which is equivalent to an infinitely large circle). A separating line can be found, if it exists, using linear programing. This takes linear time by Megiddo's algorithm [9]. For circle separability (in fact spherical separability in any fixed dimension), O'Rourke, Kosaraju and Megiddo [10] gave a linear-time algorithm for the decision problem improving earlier bounds [3, 8]. They also gave an $O(n \log n)$ time algorithm for finding the largest separating circle and a linear-time algorithm for finding the minimum separating circle between any two finite point sets. With these ideas, Boissonat et al. [4] gave a linear-time algorithm to report the smallest separating circle for two simple polygons, if any exists.

Augustine et al. [1] showed how to preprocess a point set (or a simple polygon) P, so that the largest circle isolating P from a query point can be found in logarithmic time. For the line separability problem, Edelsbrunner showed that a point set P can be preprocessed in $O(n \log n)$ time, so that a separating line between P and a query convex m-gon Q can be computed in $O(\log n + \log m)$ time [7]. In 3D, Dobkin and Kirkpatrick showed that two convex polyhedra of size n and m can be preprocessed in linear time, so that a separating plane, if any exists, can be computed in $O(\log n \cdot \log m)$ time [6]. In this paper we show that a set P on n points can be preprocessed in $O(n \log n)$ time, using $O(n)$ space, so that for any given convex m-gon Q we can find the smallest circle enclosing P and excluding Q in $O(\log n + \log m)$ time. This improves the $O(\log n \cdot \log m)$ bound presented in [2], which is described in this paper as well.

2 Preliminaries

Let P be a set of n points in the plane and let Q be a convex m-gon. A *P-circle* is a circle containing P and a *separating circle* is a P-circle whose interior does not intersect Q. A *separating line* is a straight line leaving the interiors of P and Q in different halfplanes.

Let $\mathbf{C}^*$ denote the minimum separating circle and let $\mathbf{c}^*$ be its center. Note that $\mathbf{C}^*$ passes through at least two points of P, hence $\mathbf{c}^*$ lies on an edge of the farthest-point Voronoi diagram $\mathcal{V}(P)$, which is a tree with leaves at infinity [5]. For each point p of P, let $R(p)$ be the farthest-point Voronoi region of p.

Let C_P be the minimum enclosing circle of P. If C_P is constrained by three points of P then its center, c_P, is at a vertex of $\mathcal{V}(P)$. Otherwise C_P is constrained by two points of P (forming its diameter). In this case, c_P is on the interior of an edge of $\mathcal{V}(P)$ and we insert c_P into $\mathcal{V}(P)$ by splitting the edge where it belongs. Thus, we can think of $\mathcal{V}(P)$ as a rooted tree on c_P. For any given point x on $\mathcal{V}(P)$ there is a unique path along $\mathcal{V}(P)$ joining c_P with x. Throughout this paper we will denote this path by π_x.

Given any point y in the plane, let $C(y)$ be the minimum P-circle with center on y and let $\rho(y)$ be the radius of $C(y)$. We say that y is a *separating point* if $C(y)$ is a separating circle.

3 Properties of the minimum separating circle

In this section we describe some properties of $\mathbf{C}^*$, and the relationship between $\mathbf{c}^*$ and $\mathcal{V}(P)$. Several results in this section have been proved in [2].

Let $\mathrm{CH}(P)$ denote the convex hull of P. We assume that the interiors of Q and $\mathrm{CH}(P)$ are disjoint, otherwise there is no separating circle. Also, if Q and C_P have disjoint interiors, then C_P is trivially the minimum separating circle.

*Département d'Informatique, Université Libre de Bruxelles, Brussels, Belgium, aloupis.greg@gmail.com, {lbarbafl,slanger}@ulb.ac.be

†Chargé de recherches du F.R.S.-FNRS

‡Maître de recherches du F.R.S.-FNRS

Observation 1 *Every P-circle contained in a separating circle is also a separating circle.*

Lemma 2 [2] *Let x be a point on $\mathcal{V}(P)$. The function ρ is monotonically increasing along every edge of the path π_x starting at c_P.*

We remark that Lemma 2 has also been shown to hold on vertices of π_x (not edge interiors), in [12].

Theorem 3 [2] *Let s be a point on $\mathcal{V}(P)$. If s is a separating point, then $\mathbf{c}^*$ belongs to π_s.*

Given a separating point s, we claim that if we move a point y continuously from s towards c_P on π_s, then $C(y)$ will shrink and approach Q, becoming tangent to it for the first time when y reaches $\mathbf{c}^*$. To prove this claim in Lemma 6, we introduce the following notation.

Let x be a point lying on an edge e of $\mathcal{V}(P)$ such that e lies on the bisector of $p, p' \in P$. Let $C^-(x)$ and $C^+(x)$ be the two closed convex regions obtained by splitting the disk $C(x)$ with the segment $[p, p']$. Assume that x is contained in $C^-(x)$; see Figure 1.

Observation 4 *Let x, y be two points lying on an edge e of $\mathcal{V}(P)$. If $\rho(x) > \rho(y)$, then $C^+(x) \subset C^+(y)$ and $C^-(y) \subset C^-(x)$.*

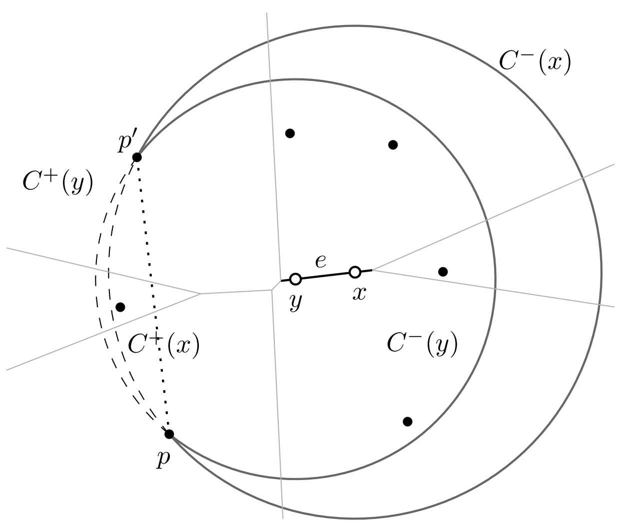

Figure 1: Observation 4 when $\rho(x) > \rho(y)$.

Lemma 5 *Let s be a point on $\mathcal{V}(P)$ and let x and y be two points on π_s. If $\rho(x) > \rho(y)$, then $C^+(x) \subset C^+(y)$ and $C^-(y) \subset C^-(x)$.*

Proof. Note that if x and y lie on the same edge, then the result holds by Observation 4. If they are on different edges, we consider the path $\Phi = (x, v_0, \ldots, v_k, y)$ contained in π_s joining x and y, such that v_i is a vertex of $\mathcal{V}(P)$, $i \in \{0, \ldots, k\}$. Thus, Observation 4 and Lemma 2 imply that $C^+(x) \subset C^+(v_0) \subset \ldots \subset C^+(v_k) \subset C^+(y)$ and that $C^-(y) \subset C^-(v_k) \subset \ldots \subset C^-(v_0) \subset C^-(x)$. $\square$

Note that $\mathbf{C}^* = C(\mathbf{c}^*)$ must be tangent to the boundary of Q. Otherwise, $\mathbf{c}^*$ could be pushed closer to the root on $\mathcal{V}(P)$, while keeping it as a separating point until it reaches Q. From now on we refer to ϕ' as the tangency point between $\mathbf{C}^*$ and Q. We claim that ϕ' lies on the boundary of $C^+(\mathbf{c}^*)$. Assume to the contrary that ϕ' lies on $C^-(\mathbf{c}^*)$. Let $\varepsilon > 0$ and let c_ε be the point obtained by moving $\mathbf{c}^*$ a distance of ε towards c_P on $\mathcal{V}(P)$. Note that by Lemma 2, $\rho(c_\varepsilon) < \rho(\mathbf{c}^*)$. In addition, Lemma 5 implies that $C^-(c_\varepsilon) \subset C^-(\mathbf{c}^*)$. Since we assumed that ϕ' lies on the boundary of $C^-(\mathbf{c}^*)$, we conclude that ϕ' does not belong to $C(c_\varepsilon)$. This implies that, for ε sufficiently small, $C(c_\varepsilon)$ is a separating circle which is a contradiction to the minimality of $\mathbf{C}^*$. The following result was mentioned in [2] without a proof.

Lemma 6 *Let s be a separating point. If x is a point lying on π_s, then $C(x)$ is a separating circle if and only if $\rho(x) \geq \rho(\mathbf{c}^*)$. Moreover, $\mathbf{C}^*$ is the only separating circle whose boundary intersects Q.*

Proof. We know by Theorem 3 that $\mathbf{c}^*$ belongs to π_s. Let x_1 and x_2 be two points on π_s such that $\rho(x_1) < \rho(\mathbf{c}^*)$ and $\rho(\mathbf{c}^*) < \rho(x_2)$. Lemma 5 implies that $C^+(\mathbf{c}^*) \subset C^+(x_1)$ and since ϕ' belongs to the boundary of $C^+(\mathbf{c}^*)$, we conclude $C(x_1)$ contains ϕ' in its interior. Therefore $C(x_1)$ is not a separating circle.

On the other hand, $C(x_2)$ contains no point of Q. Otherwise, let $q \in Q$ be a point lying in $C(x_2)$. Two cases arise: Either q belongs to $C^-(x_2)$ or q belongs to $C^+(x_2)$. In the former case, since $\rho(s) > \rho(x_2)$, $q \in C^-(x_2) \subset C^-(s)$— a contradiction since $C(s)$ is a separating circle. In the latter case, since $\rho(x_2) > \rho(\mathbf{c}^*)$, Lemma 5 would imply that q belongs to the interior of $\mathbf{C}^*$ which would also be a contradiction. $\square$

The basis of our algorithm is to find a separating point s and then perform a binary search on π_s to find a separating circle tangent to Q with center on this path.

4 Preprocessing

We first compute $\mathcal{V}(P)$ and c_P in $O(n \log n)$ time [13]. Assume that $\mathcal{V}(P)$ is stored as a binary tree with n (unbounded) leaves, so that every edge and every vertex of the tree has a set of pointers to the vertices of P defining it. Every Voronoi region is stored as a convex polygon and every vertex p of P has a pointer to $R(p)$. If c_P is not a vertex of $\mathcal{V}(P)$, we split the edge that it belongs to. We want our data structure to support binary search queries on any possible path π_s of $\mathcal{V}(P)$.

Thus, to guide the binary search we would like to have an oracle that answers queries of the following form: Given a vertex v of π_s, decide if $\mathbf{c}^*$ lies either between c_P and v or between v and s in π_s. By Lemma 6, we only need to decide if $C(v)$ is a separating circle.

We will use an operation on the vertices of $\mathcal{V}(P)$ called PointBetween with the following properties. Given two vertices u, v in π_s, PointBetween(u, v) returns a vertex z that splits the path on π_s joining u and v into two subpaths. Moreover, if we use our oracle to discard the subpath that does not contain $\mathbf{c}^*$ and we proceed recursively on the other, then, after $O(\log n)$ iterations, the search interval becomes only an edge of π_s containing $\mathbf{c}^*$.

A data structure that supports this operation was presented in [12]. This data structure can be constructed in $O(n)$ time and uses linear space.

5 The algorithm

Since Q is a convex m-gon, we can check in $O(\log m)$ time if C_P is a separating circle [7]. Thus, assume that C_P is not the minimum separating circle. To determine the position of $\mathbf{c}^*$ on $\mathcal{V}(P)$, we first find a separating point s and then search for $\mathbf{c}^*$ on π_s using our data structure. To find s, we construct a separating line L between P and Q in $O(\log n + \log m)$ time [7]. Let p_L be the point of P closest to L and assume that no other point in P lies at the same distance; otherwise rotate L slightly. Let $L_\perp$ be the perpendicular to L that contains p_L and let s be the intersection of $L_\perp$ with the boundary of $R(p_L)$; see Figure 2. We know that $L_\perp$ intersects $R(p_L)$ because L can be considered as a P-circle, containing only p_L, with center at infinity on $L_\perp$.

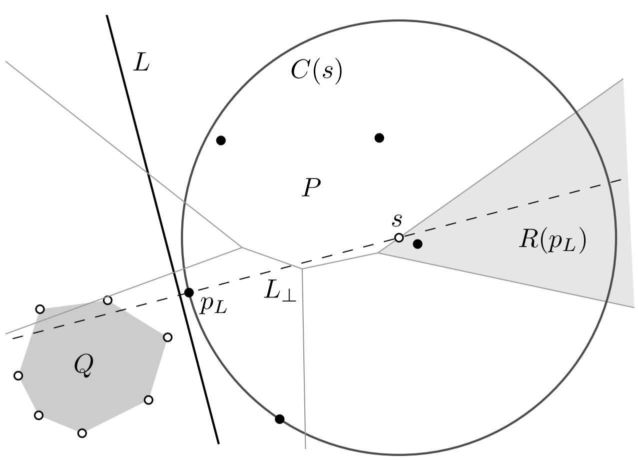

Figure 2: Construction of s. Figure borrowed from [2].

Since s is on the boundary of $R(p_L)$, $C(s)$ passes through p_L. Furthermore $C(s)$ is contained in the same halfplane defined by L that contains P. So $C(s)$ is a separating circle. Assume that s lies on the edge $\overline{xy}$ of $\mathcal{V}(P)$ with $\rho(x) > \rho(y)$ and let $\pi_s = (u_0 = s, u_1 = y, \ldots, u_r = c_P)$ be the path of length $r + 1$ joining s with c_P in $\mathcal{V}(P)$. Theorem 3 implies that $\mathbf{c}^*$ lies on π_s.

It is then possible to use our data structure to perform a binary search on the vertices of π_s, computing, at each vertex v, the radius of $C(v)$ and the distance to Q in $O(\log m)$ time. This way we can determine if $C(v)$ is a separating (or intersecting) circle. This approach finds c_P in $O(\log n \cdot \log m)$ time and was the algorithm given in [2]. However, an improvement can be obtained by using the convexity of Q.

To determine if some point v on π_s is a separating point, it is not always necessary to compute the distance between v and Q. One can first test, in $O(1)$ time, if $C(v)$ intersects a separating line tangent to Q. If not, then $C(v)$ is a separating circle and we can proceed with the binary search. Otherwise, we can try to compute a new separating line, tangent to Q, not intersecting $C(v)$. The advantage of this is that while doing so, we reduce the portion of Q that we need to consider in the future. This is done as follows:

Compute the two internal tangents L and L' between the convex hull of P and Q in $O(\log n + \log m)$ time. The techniques to construct these tangents are shown in Chapter 4 of [11]. Let q and q' be the respective tangency points of L and L' with the boundary of Q. Consider the clockwise polygonal chain $\varphi = [q = q_0, \ldots, q_k = q']$ joining q and q' as in Figure 3. Recall that ϕ' denotes the intersection point between $\mathbf{C}^*$ and the boundary of Q and note that the tangent line to $\mathbf{C}^*$ at ϕ' is a separating line. Therefore, ϕ' must lie on an edge of φ since no separating line passes through any other boundary point of Q.

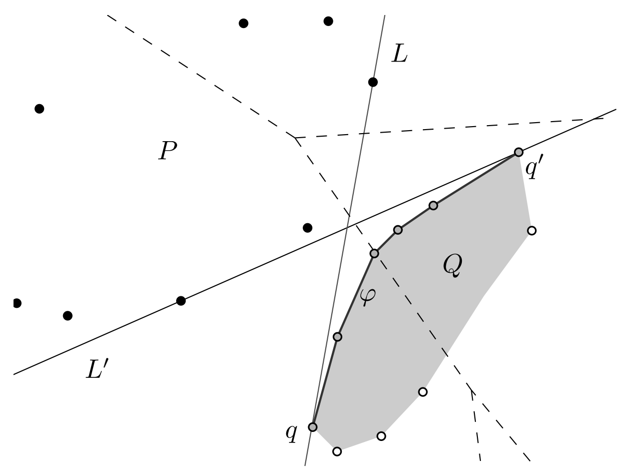

Figure 3: The construction of φ.

If $q = q'$, then $\phi' = q$ and hence we can ignore Q and compute the minimum separating circle between P and

q. As mentioned previously, this takes $O(\log n)$ time. Assume from now on that $q \neq q'$, as shown in Figure 3.

For each edge $e_i = q_i q_{i+1}$ $(0 \leq i \leq k-1)$ of φ, let ℓ_i be the line extending that edge. By construction, we know that each ℓ_i separates P and Q. We say that a point x on ℓ_i but not on e_i lies to the left of e_i if it is closer to q_i, or to the right if it is closer to q_{i+1}.

Our algorithm will essentially perform two parallel binary searches, the first one on π_s and the second one on φ, such that at each step we discard either a section of π_s or a linear fraction of φ. As we search on π_s, every time we find a separating circle, we move towards c_P. When we confirm that a P-circle intersects Q, we move away from c_P. To confirm if a vertex v is a separating point, we compare $C(v)$ to some separating line ℓ_i for intersection in constant time. If $C(v)$ is a separating circle, we discard the section of the path lying below v on $\mathcal{V}(P)$. If $C(v)$ does intersect ℓ_i, we make a quick attempt to check if $C(v)$ intersects Q by comparing $C(v)$ and the edge e_i for intersection. If they intersect, v is not a separating point and we can proceed with the binary search on π_s. Otherwise, the intersection of $C(v)$ with ℓ_i lies either to the left or to the right of e_i. However, in this case we are not able to quickly conclude whether $C(v)$ intersects Q or not. Thus, we suspend the binary search on $\mathcal{V}(P)$ and focus on $C(v)$, using its intersection with ℓ_i to eliminate half of φ. Specifically, the fact that $C(v)$ intersects ℓ_i to one side of e_i (right or left) tells us that no future P-circle on our search will intersect ℓ_i on the other side of e_i. This implicitly discards half of φ from future consideration, and is discussed in more detail in Theorem 7. Thus, in constant time, we manage to remove a section of the path π_s, or half of φ. The entire process is detailed in Algorithm 1.

Theorem 7 *Algorithm 1 finds the edge of π_s containing $\mathbf{c}^*$ in $O(\log n + \log m)$ time.*

Proof. Our algorithm maintains two invariants. The first is that $C(u)$ is never a separating circle and $C(v)$ is always a separating circle. To begin with, $C(u) = C(s)$ is a separating circle while $C(v) = C_P$ is not. If either of these assumptions does not hold, the problem is solved trivially, without resorting to this algorithm. Changes to u and v occur in steps 14 or 17, and in both the invariant is preserved. Thus, $\mathbf{c}^*$ always lies on the path joining u with v. As a second invariant, ϕ' always lies on the clockwise path joining q_a with q_b along φ. We already explained that the invariant holds when $a = 0$ and $b = k$, corresponding to the inner tangents supporting P and Q. Thus, we only need to look at steps 20 and 22 where a and b are redefined. We analyze Step 20, however Step 22 is analogous.

In Step 20 we know that $C(z)$ intersects ℓ_j to the left of e_j and that e_j does not intersect $C(z)$. We claim that for every point w lying on an edge of π_s, if $C(w)$ is a

Algorithm 1 Given $\varphi = [q = q_0, \ldots, q_k = q']$ and $\pi_s = (u_0 = s, u_1 = y, \ldots, u_r = c_P)$, find the edge of π_s that contains $\mathbf{c}^*$.

1: Define the initial subpath of π_s that contains $\mathbf{c}^*$, $u \leftarrow s, v \leftarrow c_P$
2: Define the initial search interval on the chain φ, $a \leftarrow 0, b \leftarrow k$
3: **if** u and v are neighbors in $\mathcal{V}(P)$ and $b = a+1$ **then**
4:     Finish and report the segment $S = [u, v]$ and the segment $H = [q_a, q_b]$
5: **end if**
6: Let $z \leftarrow$ FindPointBetween(u, v), $j \leftarrow \lfloor \frac{a+b}{2} \rfloor$
7: Let $e_j \leftarrow \overline{q_j q_{j+1}}$ and let ℓ_j be the line extending e_j
8: **if** $b > a+1$ **then**
9:     Compute $\rho(z)$ and let $\delta \leftarrow d(z, \ell_j)$, $\Delta \leftarrow d(z, e_j)$
10: **else**
11:     Compute $\rho(z)$ and let $\delta \leftarrow d(z, e_j)$, $\Delta \leftarrow d(z, e_j)$
12: **end if**
13: **if** $\rho(z) \leq \delta$, that is $C(z)$ is a separating circle **then**
14:     Move forward on π_s, $u \leftarrow z$ and return to Step 3
15: **else**
16:     **if** $\rho(z) > \Delta$, that is if $C(z)$ is not a separating circle **then**
17:         Move backward on π_s, $v \leftarrow z$ and return to Step 3
18:     **else**
19:         **if** $C(z)$ intersects ℓ_j to the left of e_j **then**
20:             Discard the polygonal chain to the right of e_j, $b \leftarrow \max\{j, a+1\}$
21:         **else**
22:             Discard the polygonal chain to the left of e_j, $a \leftarrow j$
23:         **end if**
24:         Return to Step 3
25:     **end if**
26: **end if**

separating circle that intersects ℓ_j, then it intersects it to the left of e_j. Note that if our claim is true, we can ignore the polygonal chain lying to the right of e_j since no separating circle will intersect it. To prove our claim, suppose that there is a point w on π_s, such that $C(w)$ is a separating circle and $C(w)$ intersects ℓ_j to the right of e_j. Let x and x' be two points on the intersection of ℓ_j with $C(w)$ and $C(z)$, respectively. First suppose that $\rho(w) < \rho(z)$ and recall that by Lemma 5, since x' lies on $C^+(z) \subset C^+(w)$, x' lies in $C(w)$. Thus, both x and x' belong to $C(w)$ which by convexity implies that e_j is contained in $C(w)$. Therefore $C(w)$ is not a separating circle which is a contradiction. Analogously, if $\rho(w) > \rho(z)$, then e_j is contained in $C(z)$ which is directly a contradiction since we assumed the opposite. Thus, our claim holds.

Note that in each iteration of the algorithm, a, b, u or v are redefined so that either a linear fraction of φ is discarded, or a part of π_s is discarded and a new call to PointBetween is performed. Recall that our data structure guarantees that $O(\log n)$ calls to PointBetween are sufficient to reduce the search interval in π_s to an edge [12]. Thus, the algorithm finishes in $O(\log n + \log m)$ iterations.

One additional detail needs to be considered when $b = a + 1$. In this case only one edge $e = [q_a, q_{a+1}]$ remains from φ, and ϕ' lies on e. Thus, if the line ℓ extending e intersects $C(z)$ but e does not, then either Step 20 or 22 is executed. However, nothing will change in these steps and the algorithm will loop. In order to avoid that, we check in Step 8 if only one edge e of φ remains. If this is the case, we know by our invariant that ϕ' belongs to e and therefore we continue the search computing the distance to e instead of computing the distance to the line extending it. This way, the search on φ stops but it continues on π_s until the edge of $\mathcal{V}(P)$ containing $\mathbf{c}^*$ is found.

Since we ensured that every edge in $\mathcal{V}(P)$ has pointers to the points in P that defined it, every step in the algorithm can be executed in $O(1)$ time. Thus, we conclude that Algorithm 1 finishes in $O(\log n + \log m)$ time. Since both invariants are preserved during the execution, Lemma 6 implies that the algorithm returns segments $[u, v]$ from π_s containing $\mathbf{c}^*$, and $[q_a, q_b]$ from φ containing ϕ'. □

From the output of Algorithm 1 it is trivial to obtain $\mathbf{c}^*$ in constant time.

Corollary 8 *After preprocessing a set P of n points in $O(n \log n)$ time, the minimum separating circle between P and any query convex m-gon can be found in $O(\log n + \log m)$ time.*

References

[1] J. Augustine, S. Das, A. Maheshwari, S. C. Nandy, S. Roy, and S. Sarvattomananda. Localized geometric query problems. *CoRR*, abs/1111.2918, 2011.

[2] L. Barba and J. Urrutia. Dynamic circle separability between convex polygons. In *Proceedings of the Spanish Meetings in Computational Geometry*, pages 43–46, 2011.

[3] B. K. Bhattacharya. Circular separability of planar point sets. *Computational Morphology*, pages 25–39, 1988.

[4] J.-D. Boissonnat, J. Czyzowicz, O. Devillers, and M. Yvinec. Circular separability of polygons. *Algorithmica*, 30:67–82, 2001.

[5] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, third edition, 2008.

[6] D. P. Dobkin and D. G. Kirkpatrick. Fast detection of polyhedral intersection. *Theoretical Computer Science*, 27(3):241 – 253, 1983.

[7] H. Edelsbrunner. Computing the extreme distances between two convex polygons. *Journal of Algorithms*, 6(2):213 – 224, 1985.

[8] C. E. Kim and T. A. Anderson. Digital disks and a digital compactness measure. In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 117–124, New York, NY, USA, 1984. ACM.

[9] N. Megiddo. Linear programming in linear time when the dimension is fixed. *J. ACM*, 31(1):114–127, Jan. 1984.

[10] J. O'Rourke, S. Rao Kosaraju, and N. Megiddo. Computing circular separability. *Discrete and Computational Geometry*, 1:105–113, 1986.

[11] F. Preparata and M. Shamos. *Computational geometry: an introduction.* Springer-Verlag, 1985.

[12] S. Roy, A. Karmakar, S. Das, and S. C. Nandy. Constrained minimum enclosing circle with center on a query line segment. *Computational Geometry*, 42(6-7):632 – 638, 2009.

[13] M. I. Shamos and D. Hoey. Closest-point problems. In *Foundations of Computer Science, 1975., 16th Annual Symposium on*, pages 151 –162, oct. 1975.