# On the Planar Two-Center Problem and Circular Hulls

**Haitao Wang**
Department of Computer Science, Utah State University, Logan, UT 84322, USA
haitao.wang@usu.edu

## Abstract

Given a set $S$ of $n$ points in the Euclidean plane, the two-center problem is to find two congruent disks of smallest radius whose union covers all points of $S$. Previously, Eppstein [SODA'97] gave a randomized algorithm of $O(n \log^2 n)$ expected time and Chan [CGTA'99] presented a deterministic algorithm of $O(n \log^2 n \log^2 \log n)$ time. In this paper, we propose an $O(n \log^2 n)$ time deterministic algorithm, which improves Chan's deterministic algorithm and matches the randomized bound of Eppstein. If $S$ is in convex position, we solve the problem in $O(n \log n \log \log n)$ deterministic time. Our results rely on new techniques for dynamically maintaining circular hulls under point insertions and deletions, which are of independent interest.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms; Theory of computation → Computational geometry

**Keywords and phrases** two-center, disk coverage, circular hulls, dynamic data structures

**Digital Object Identifier** 10.4230/LIPIcs.SoCG.2020.68

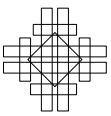
**Related Version** A full version of this paper is available at `https://arxiv.org/abs/2002.07945`.

## 1 Introduction

Given a set $S$ of $n$ points in the Euclidean plane, we consider the planar 2-center problem that is to find two congruent disks of smallest radius whose union covers all points of $S$.

The classical 1-center problem for a set of points is to find the smallest disk covering all points, and the problem can be solved in linear time in any fixed dimensional space [9, 13, 24]. As a natural generalization, the 2-center problem has attracted much attention. Hershberger and Suri [19] first solved the decision version of the problem in $O(n^2 \log n)$ time, which was later improved to $O(n^2)$ time [18]. Using this result and parametric search [23], Agarwal and Sharir [2] gave an $O(n^2 \log^3 n)$ time algorithm for the 2-center problem. Katz and Sharir [21] achieved the same running time by using expanders instead of parametric search. Eppstein [15] presented a randomized algorithm of $O(n^2 \log^2 n \log \log n)$ expected time. Later, Jaromczyk and Kowaluk [20] proposed an $O(n^2)$ time algorithm. A breakthrough was achieved by Sharir [26], who proposed the first subquadratic algorithm for the problem, and the running time is $O(n \log^9 n)$. Afterwards, following Sharir's algorithmic scheme, Eppstein [16] derived a randomized algorithm of $O(n \log^2 n)$ expected time, and then Chan [6] developed an $O(n \log^2 n \log^2 \log n)$ time deterministic algorithm and a randomized algorithm of $O(n \log^2 n)$ time with high probability. Recently, Tan and Jiang [27] proposed a simple algorithm of $O(n \log^2 n)$ time based on binary search, but unfortunately, the algorithm is not correct (see the full paper for details). The problem has an $\Omega(n \log n)$ time lower bound in the algebraic decision tree model [16], by a reduction from the max-gap problem.

In this paper, we present a new deterministic algorithm of $O(n \log^2 n)$ time, which improves the $O(n \log^2 n \log^2 \log n)$ time deterministic algorithm by Chan [6] and matches the randomized bound of $O(n \log^2 n)$ [6, 16]. This is the first progress on the problem since Chan's work [6] was published twenty years ago. Further, if $S$ is in convex position (i.e., the points


of $S$ are all on the convex hull of $S$), our technique solves the problem in $O(n \log n \log\log n)$ time. Previously, Kim and Shin [22] announced an $O(n \log^2 n)$ time algorithm for this convex position case, but Tan and Jiang [27] found errors in their time analysis.

Some variations of the 2-center problem have also been considered in the literature. Agarwal et al. [3] studied the discrete 2-center problem where the centers of the two disks must be in $S$, and they solved the problem in $O(n^{4/3} \log^5 n)$ time. Agarwal and Phillips [1] considered an outlier version of the problem where $k$ points of $S$ are allowed to be outside the two disks, and they presented a randomized algorithm of $O(nk^7 \log^3 n)$ expected time. In addition to the set $S$, the problem of Halperin et al. [17] also involves a set of pairwise disjoint simple polygons, and the centers of the two disks are required to lie outside all polygons. Both exact and approximation algorithms are given in [17]. Arkin et al. [4] studied a bichromatic 2-center problem for a set of $n$ pairs of points in the plane, and the goal is to assign a red color to a point and a blue color to the other point for every pair, such that $\max\{r_1, r_2\}$ is minimized, where $r_1$ (resp., $r_2$) is the radius of the smallest disk covering all red (resp., blue) points. Arkin et al. [4] gave an $O(n^3 \log^2 n)$ time algorithm, which was recently improved to $O(n^2 \log^2 n)$ time by Wang and Xue [28].

## 1.1 Our techniques

Let $D_1^*$ and $D_2^*$ be two congruent disks in an optimal solution such that the distance of their centers is minimized. Let $r^*$ be their radius and $\delta^*$ the distance of their centers. If $\delta^* \geq r^*$, we call it the *distant case*; otherwise, it is the *nearby case*.

Eppstein [16] already solved the distant case in $O(n \log^2 n)$ deterministic time. Solving the nearby case turns out to be the bottleneck in all previous three sub-quadratic time algorithms [6, 16, 26]. Specifically, Sharir [26] first solved it in $O(n \log^9 n)$ deterministic time. Eppstein [16] gave a randomized algorithm of $O(n \log n \log\log n)$ expected time. Chan [16] proposed a randomized algorithm of $O(n \log n)$ time with high probability and a deterministic algorithm of $O(n \log^2 n \log^2 \log n)$ time. Our contribution is an $O(n \log n \log\log n)$ time deterministic algorithm for the nearby case, which improves Chan's algorithm by a factor of $\log n \log\log n$. Combining with the $O(n \log^2 n)$ time deterministic algorithm of Eppstein [16] for the distant case, the 2-center problem can now be solved in $O(n \log^2 n)$ deterministic time. Interestingly, solving the distant case now becomes the bottleneck of the problem.

Our algorithm (for the nearby case) is based on Chan's framework [6]. Our improvement is twofold. First, Chan [6] derived an $O(n \log n)$ time algorithm for the *decision problem*, i.e., given $r$, decide whether $r^* \leq r$. We improve the algorithm to $O(n)$ time, after $O(n \log n)$ time preprocessing. Second, Chan [6] solved the optimization problem (i.e., the original 2-center problem) by parametric search. To this end, Chan developed a parallel algorithm for the decision problem and the algorithm runs in $O(\log n \log^2 \log n)$ parallel steps using $O(n \log n)$ processors. By applying Cole's parametric search [10] and using his $O(n \log n)$ time decision algorithm, Chan solved the optimization problem in $O(n \log^2 n \log^2 \log n)$ time. We first notice that simply replacing Chan's $O(n \log n)$ time decision algorithm with our new $O(n)$ time algorithm does not lead to any improvement. Indeed, in Chan's parallel algorithm, the number of processors times the number of parallel steps is $O(n \log^2 n \log^2 \log n)$. We further design another parallel algorithm for the decision problem, which runs in $O(\log n \log\log n)$ parallel steps using $O(n)$ processors. Consequently, by applying Cole's parametric search with our $O(n)$ time decision algorithm, we solve the optimization problem in $O(n \log n \log\log n)$ time. Note that although Cole's parametric search is used, our algorithm mainly involves independent binary searches and no sorting networks are needed.

In addition, we show that our algorithm can be easily applied to solving the convex position case in $O(n \log n \log\log n)$ time.

**Circular hulls.** To obtain our algorithm for the decision problem, we develop new techniques for *circular hulls* [19] (also known as $\alpha$-*hulls* with $\alpha = 1$ [14]). A circular hull of radius $r$ for a set $Q$ of points is the common intersection of all disks of radius $r$ containing $Q$ (to see how circular hulls are related to the two-center problem, notice that there exists a disk of radius $r$ covering all points of $Q$ if and only if the circular hull of $Q$ of radius $r$ exists). Although circular hulls have been studied before, our result needs more efficient algorithms for certain operations. For example, two algorithms [14, 19] were known for constructing the circular hull for a set of $n$ points; both algorithms run in $O(n \log n)$ time, even if the points are given sorted. We instead present a linear-time algorithm once the points are sorted. Also, Hershberger and Suri [19] gave a linear-time algorithm to find the common tangents of two circular hulls separated by a line, and we design a new algorithm of $O(\log n)$ time. We also need to maintain a dynamic circular hull for a set of points under point insertions and deletions. Hershberger and Suri [19] gave a semi-dynamic data structure that can support deletions in $O(\log n)$ amortized time each. In our problem, we need to handle both insertions and deletions but with the following special properties: the point in each insertion must be to the right of all points of $Q$ and the point in each deletion must be the leftmost point of $Q$. Our data structure can handle each update in $O(1)$ amortized time (which leads to the linear time decision algorithm for the 2-center problem[1]). We believe that these results on circular hulls are interesting in their own right.

**Outline.** We introduce notation and review some previous work in Section 2. In Section 3, we present our decision algorithm, and the algorithm needs a data structure to maintain circular hulls dynamically, which is given in the full paper. Section 4 solves the optimization problem. The convex position case is discussed in Section 5.

## 2 Preliminaries

We begin with some notation. It suffices to solve the nearby case. Thus, we assume $\delta^* < r^*$ in the rest of the paper. In the nearby case, it is possible to find in $O(n)$ time a constant number of points such that at least one of them, denoted by $o$, is in $D_1^* \cap D_2^*$ [16]. We assume that $o$ is the origin of the plane. We make a general position assumption: no two points of $S$ are collinear with $o$ and no two points of $S$ have the same $x$-coordinate. This assumption does not affect the running time of the algorithm, but simplifies the presentation.

For any set $P$ of points in the plane, let $\tau(P)$ denote the radius of the smallest enclosing disk of $P$. For a connected region $B$ in the plane, let $\partial B$ denote the boundary of $B$.

The boundaries of the two disks $D_1^*$ and $D_2^*$ have exactly two intersections, and let $\rho_1$ and $\rho_2$ be the two rays through these intersections emanating from $o$ (e.g., see Fig. 1). As argued in [6], one of the two coordinate axes must separate $\rho_1$ and $\rho_2$ since the angle between the two rays lies in $[\pi/2, 3\pi/2]$, and without loss of generality, we assume it is the $x$-axis.

Let $S^+$ denote the subset of points of $S$ above the $x$-axis, and $S^- = S \setminus S^+$. For notational simplicity, let $|S^+| = |S^-| = n$. Let $p_1, p_2, \ldots, p_n$ be the sorted list of $S^+$ counterclockwise around $o$, and $q_1, q_2, \ldots, q_n$ the sorted list of $S^-$ also counterclockwise around $o$ (e.g., see Fig. 2). For each $i = 0, 1, \ldots, n$ and $j = 0, 1, \ldots, n$, define $L_{ij} = \{p_{i+1} \ldots, p_n, q_1, \ldots, q_j\}$ and $R_{ij} = \{q_{j+1}, \ldots, q_n, p_1, \ldots, p_i\}$. Note that if $i = n$, then $L_{ij} = \{q_1, \ldots, q_j\}$, and if $j = n$, then $R_{ij} = \{p_1, \ldots, p_i\}$. In words, if we consider a ray emanating from $o$ and between $p_i$ and $p_{i+1}$, and another ray emanating from $o$ and between $q_j$ and $q_{j+1}$, then $L_{ij}$ (resp., $R_{ij}$) consist of all points to the left (resp., right) of the two rays (e.g., see Fig. 2).

[1] As will be clear later, the points involved in our dynamic circular hull problem are actually sorted radially by a point; we can extend the result for the left-right sorted case to the radically sorted case.

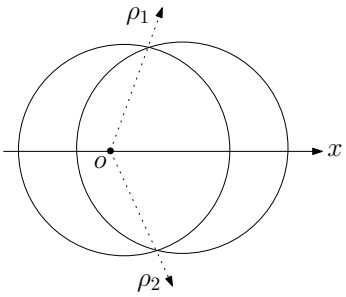



**Figure 1** Illustrating the nearby case.

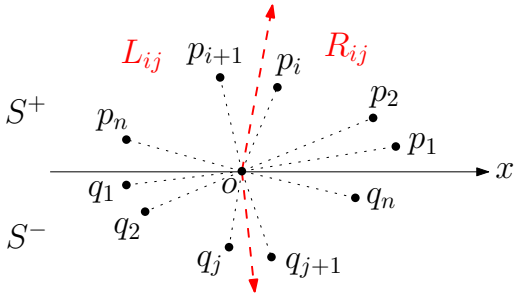



**Figure 2** Illustrating the points of $S^+$ and $S^-$.

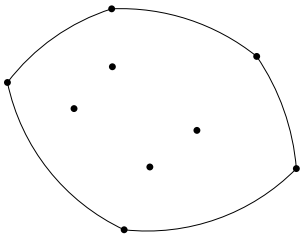


**Figure 3** The circular hull of a set of points.

Note that the partition of $S$ by the two rays $\rho_1 \cup \rho_2$ is $\{L_{ij}, R_{ij}\}$ for some $i$ and $j$, and thus $r^* = \max\{\tau(L_{ij}), \tau(R_{ij})\}$. Define $A[i,j] = \tau(L_{ij})$ and $B[i,j] = \tau(R_{ij})$, for all $0 \leq i, j \leq n$. Then, $r^* = \min_{0 \leq i,j \leq n} \max\{A[i,j], B[i,j]\}$. If we consider $A$ and $B$ as $(n+1) \times (n+1)$ matrices, then each row of $A$ (resp., $B$) is monotonically increasing (resp., decreasing) and each column of $A$ (resp., $B$) is monotonically decreasing (resp., increasing). For each $i \in [0, n]$, define $r_i^* = \min_{0 \leq j \leq n} \max\{A[i,j], B[i,j]\}$. Thus, $r^* = \min_{0 \leq i \leq n} r_i^*$.

## 2.1 Circular hulls

For any point $c$ in the plane and a value $r$, we use $D_r(c)$ to denote the disk centered at $c$ with radius $r$. For a set $Q$ of points in the plane, define $\mathcal{I}_r(Q) = \bigcap_{c \in Q} D_r(c)$, i.e., the common intersection of the disks $D_r(c)$ for all points $c \in Q$. Note that $\mathcal{I}_r(Q)$ is convex. A dual concept of $\mathcal{I}_r(Q)$ is the *circular hull* [19] (also known as $\alpha$-*hull* with $\alpha = 1$ [14]; e.g., see Fig 3), denoted by $\alpha_r(Q)$, which is the common intersection of all disks of radius $r$ containing $Q$. $\alpha_r(Q)$ is convex and unique. The vertices of $\alpha_r(Q)$ is a subset of $Q$ and the edges are arcs of circles of radius $r$. $\mathcal{I}_r(Q)$ and $\alpha_r(Q)$ are dual to each other: Every arc of $\alpha_r(Q)$ is on the circle of radius $r$ centered at a vertex of $\mathcal{I}_r(Q)$ and every arc of $\mathcal{I}_r(Q)$ is on the circle of radius $r$ centered at a vertex of $\alpha_r(Q)$. Note that $\alpha_r(Q)$ exists if and only if $\mathcal{I}_r(Q) \neq \emptyset$, which is true if and only $\tau(Q) \leq r$. For brevity, we often drop the subscript $r$ from $\mathcal{I}_r(Q)$ and $\alpha_r(Q)$ if it is clear from the context.

Circular hulls are critical to our algorithm. As discussed in [19], circular hulls have many properties similar to convex hulls. However, circular hulls also have special properties that convex hulls do not possess. For example, the circular hull for a set of points may not exist. Also, the leftmost point of a set $Q$ of points must be a vertex of the convex hull of $Q$, but this may not be the case for the circular hull. Due to these special properties, extending algorithms on convex hulls to circular hulls sometimes is not trivial, as will be seen later. In the following, we introduce some concepts on circular hulls that will be needed later.

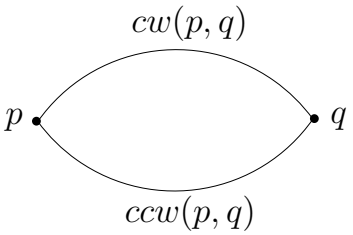



**Figure 4** Illustrating two minor arcs of $p$ and $q$.

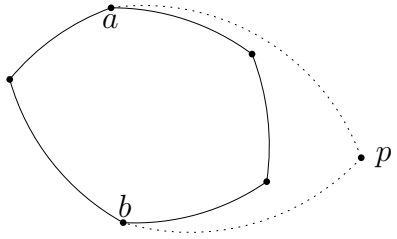



**Figure 5** Illustrating the two tangents from $p$ to $\alpha(Q)$: $cw(a, p)$ and $ccw(b, p)$.

We assume that $r = 1$ and thus a disk of radius $r$ is a *unit disk* (whose boundary is a *unit circle*). We use $\alpha(Q)$ to refer to $\alpha_r(Q)$. We assume that $\alpha(Q)$ exists.

For any arc of a circle, the circle is called the *supporting circle* of the arc, and the disk enclosed in the circle is called the *supporting disk* of the arc. If a disk $D$ contains all points of a set $P$, we say $D$ *covers* $P$. We say that a set $P$ of points in the plane is *unit disk coverable* if there is a unit disk containing all points of $P$, which is true if and only if $\alpha(P)$ exists.

Consider two points $p$ and $q$ that are unit disk coverable. There must be a unit circle with $p$ and $q$ on it, and we call the arc of the circle subtending an angle of at most 180° a *minor arc* [19]. Note that there are two minor arcs connecting $p$ and $q$, we use $cw(p, q)$ to refer to the one clockwise around the center of the supporting circle of the arc from $p$ to $q$, and use $ccw(p, q)$ to refer to the other one (e.g., see Fig. 4). Note that $cw(p, q) = ccw(q, p)$ and $ccw(p, q) = cw(q, p)$. For any minor arc $w$, we use $D(w)$ to denote the supporting disk of $w$, i.e., the disk whose boundary contains $w$. Note that all arcs of $\alpha(Q)$ are minor arcs. We make a general position assumption that no point of $Q$ is on a minor arc of two other points of $Q$. The following observation has already been discovered previously [14, 19].

▶ **Observation 1** ([14, 19]).
1. *A point $p$ of $Q$ is a vertex of $\alpha(Q)$ iff there is a unit disk covering $Q$ and with $p$ on the boundary.*
2. *A minor arc connecting two points of $Q$ is an arc of $\alpha(Q)$ iff its supporting disk covers $Q$.*
3. *$\alpha(Q)$ is the common intersection of the supporting disks of all arcs of $\alpha(Q)$.*
4. *A unit disk covers $Q$ iff it contains $\alpha(Q)$.*
5. *For any subset $Q'$ of $Q$, $\alpha(Q') \subseteq \alpha(Q)$.*

For any vertex $v$ of $\alpha(Q)$, we refer to the clockwise neighboring vertex of $v$ on $\alpha(Q)$ the *clockwise neighbor* of $v$, and the *counterclockwise* neighbor is defined analogously. We use $cw(v)$ and $ccw(v)$ to denote $v$'s clockwise and counterclockwise neighbors, respectively.

**Tangents.** Consider a vertex $v$ in the circular hull $\alpha(Q)$. Consider the arc $cw(ccw(v), v)$ of $\alpha(Q)$. Let $D$ be the disk $D(cw(ccw(v), v))$. By Observation 1(2) and (4), $D$ contains $\alpha(Q)$. Observe that if we rotate $D$ around $v$ clockwise until $\partial D$ contains the arc $cw(v, cw(v))$, $D$ always contains $\alpha(Q)$, and in fact, this continuum of disks $D$ are the only unit disks that contain $\alpha(Q)$ and have $v$ on the boundaries. For each of such disk $D$, we say that $D$ (and any part of $\partial D$ containing $v$) is *tangent* to $\alpha(Q)$ at $v$. We have the following observation.

▶ **Observation 2.** *A unit disk $D$ that contains a vertex $v$ of $\alpha(Q)$ on its boundary is tangent to $\alpha(Q)$ at $v$ if and only if $D$ contains both $cw(v)$ and $ccw(v)$.*

Let $p$ be a point outside $\alpha(Q)$. If there is a vertex $a$ on $\alpha(Q)$ such that $D(cw(a, p))$ is tangent to $\alpha(Q)$ at $a$, then the arc $cw(a, p)$ is an *upper tangent* from $p$ to $\alpha(Q)$; e.g., see Fig 5. If there is a vertex $b$ on $\alpha(Q)$ such that $D(ccw(b, p))$ is tangent to $\alpha(Q)$ at $b$, then

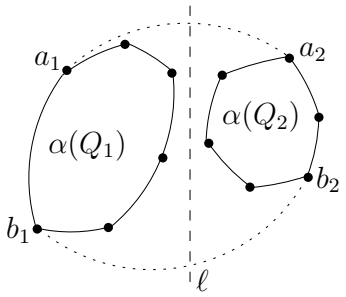



**Figure 6** Illustrating the upper common tangent $cw(a_1, a_2)$ and the lower common tangent $ccw(b_1, b_2)$ of $\alpha(Q_1)$ and $\alpha(Q_2)$.

the arc $ccw(b, p)$ is a *lower tangent* from $p$ to $\alpha(Q)$. By replacing the arcs of $\alpha(Q)$ clockwise from $a$ to $b$ with the two tangents from $p$, we obtain $\alpha(Q \cup \{p\})$. This also shows that $p$ has tangents to $\alpha(Q)$ if and only if $Q \cup \{p\}$ is unit disk coverable and $p$ is outside $\alpha(Q)$. Note that $a = b$ is possible, in which case $\alpha(Q \cup \{p\}) = \alpha(\{a, p\})$.

**Common tangents of two circular hulls.** Let $Q_1$ and $Q_2$ be two sets of points in the plane such that all points of $Q_1$ (resp., $Q_2$) are to the left (resp., right) of a vertical line $\ell$. Let $Q = Q_1 \cup Q_2$. A unit disk $D$ that is tangent to $\alpha(Q_1)$, say at a vertex $a$, and is also tangent to $\alpha(Q_2)$, say at a vertex $b$, is said to be *commonly tangent* to $\alpha(Q_1)$ and $\alpha(Q_2)$. The minor arc of $D$ connecting $a$ and $b$ is called a *common tangent* of the two circular hulls. It is an *upper* (resp, *lower*) tangent if it is clockwise (resp., counterclockwise) from $a$ to $b$ along the minor arc (e.g., see Fig. 6). The common tangents of $\alpha(Q_1)$ and $\alpha(Q_2)$ may not exist. Indeed, if $\alpha(Q)$ does not exist, then the common tangents do not exist. Otherwise the common tangents do not exist either if all points of $Q_2$ are contained in $\alpha(Q_1)$, which happens only if $Q_2$ is covered by $D(w)$ for the rightmost arc $w$ of $\alpha(Q_1)$ and we call it the $Q_1$-*dominating case*, or if all points of $Q_1$ are contained in $\alpha(Q_2)$, which happens only if $Q_1$ is covered by $D(w')$ for the leftmost arc $w'$ of $\alpha(Q_2)$ and we call it the $Q_2$-*dominating case*. If none of the above cases happens, then there are exactly two common tangents between the two hulls. Each tangent intersects the vertical line $\ell$, which separates $Q_1$ and $Q_2$, and the upper tangent intersects $\ell$ higher than the lower tangent does.

## 3 The decision problem

This section is concerned with the decision problem: Given a value $r$, decide whether $r^* \leq r$. Previously, Chan [6] solved the problem in $O(n \log n)$ time (Chan actually considered a slightly different problem: decide whether $r^* < r$, but the idea is similar). We present an $O(n)$ time algorithm, after $O(n \log n)$ time preprocessing to sort all points of $S^+$ and $S^-$ to obtain the sorted lists $p_1, \ldots, p_n$ and $q_1, \ldots, q_n$.

Given $r$, we use the following algorithmic framework in Algorithm 1 from [6] (see Theorem 3.3), which can decide whether $r^* \leq r$, and if yes, report all indices $i$ with $r_i^* \leq r$.

The algorithm is simple, but the technical crux is in how to decide whether $A[i, j+1] \leq r$ and whether $B[i, j] \leq r$. Chan [6] built a data structure in $O(n \log n)$ time so that each of these two steps can be done in $O(\log n)$ time, which leads to an overall $O(n \log n)$ time for his decision algorithm. Our innovation is a new data structure that can perform each of the two steps in $O(1)$ amortized time, resulting in an $O(n)$ time algorithm. Our idea is motivated by the following observation.

**Algorithm 1** The decision algorithm of Chan [6].

```
1 j ← −1;
2 for i ← 0 to n do
3     while A[i, j + 1] ≤ r do j + + ;
4     if B[i, j] ≤ r then report i ;
5 end
```

▶ **Observation 3.** *All such elements $A[i, j + 1]$ that are checked in the algorithms (i.e., Line 3) are in a path of the matrix $A$ from $A[0, 0]$ to an element in the bottom row and the path only goes rightwards or downwards. The same holds for the elements of $B$ that are checked in the algorithms (i.e., Line 4).*

We call such a path in $A$ as specified in the observation a *monotone path*, which has at most $2(n + 1)$ elements of $A$. We show that we can determine in $O(n)$ time whether $A[i, j] \leq r$ for all elements $A[i, j]$ in a monotone path of $A$. The same algorithm works for $B$ as well.

Let $\pi$ be a monotone path of $A$, starting from $A[0, 0]$. Consider any element $A[i, j]$ on $\pi$. Recall that $A[i, j] = \tau(L_{ij})$. The next value of $\pi$ after $A[i, j]$ is either $A[i, j + 1]$ or $A[i + 1, j]$, i.e., either $\tau(L_{i,j+1})$ or $\tau(L_{i+1,j})$. Note that $L_{i,j+1}$ can be obtained from $L_{ij}$ by inserting $q_{j+1}$ and $L_{i+1,j}$ can be obtained from $L_{ij}$ by deleting $p_{i+1}$. Because the points $p_1, p_2, \ldots, p_n, q_1, q_2 \ldots, q_n$ are ordered around $o$ counterclockwise, our problem becomes the following. Maintain a sublist $Q$ of the above sorted list of $S$, with $Q = S^+$ initially, to determine whether $\tau(Q) \leq r$ (or equivalently whether $\alpha_r(Q)$ exists) under deletions and insertions, such that a deletion operation deletes the first point of $Q$ and an insertion operation inserts the point of $S$ following the last point of $Q$. Further, deletions only happen to points of $S^+$ (i.e., once $p_n$ is deleted from $Q$, no deletions will happen). We refer to the problem as the *dynamic circular hull problem*. In the full paper we solve the problem in $O(n)$ time, i.e., each update takes $O(1)$ amortized time. This leads to the following result.

▶ **Theorem 4.** *Assume that points of $S$ are sorted cyclically around $o$. Given any $r$, whether $r^* \leq r$ can be decided in $O(n)$ time.*

▶ Remark. For the nearby case, Chan proposed (in Theorem 3.4 [16]) a randomized algorithm of $O(n \log n)$ time with high probability (i.e., $1 - 2^{-\Omega(n/\log^{12} n)}$) by using his $O(n \log n)$ time decision algorithm. Applying our decision algorithm and following Chan's algorithm (specifically, setting $m$ to $\lfloor n/\log^7 n \rfloor$ instead of $\lfloor n/\log^6 n \rfloor$ in the algorithm of Theorem 3.4 in [16]), we can obtain the following result: After $O(n \log n)$ deterministic time preprocessing, we can compute $r^*$ for the nearby case in $O(n)$ time with high probability (i.e., $1 - 2^{-\Omega(n/\log^{14} n)}$).

## 4 The optimization problem

With Theorem 4, we solve the optimization problem by parametric search [10, 23]. As Chan's algorithm [6], because our decision algorithm is inherently sequential, we need to design a parallel decision algorithm. Chan [6] gave a parallel decision algorithm that runs in $O(\log n \log^2 \log n)$ parallel steps using $O(n \log n)$ processors. Consequently, by using his $O(n \log n)$ time decision algorithm and applying Cole's parametric search [10], Chan [6] solved the optimization problem in $O(n \log^2 n \log^2 \log n)$ time. By following Chan's algorithmic scheme, we develop a new parallel decision algorithm that runs in $O(\log n \log \log n)$ parallel steps using $O(n)$ processors. Then, with the serial decision algorithm in Theorem 4 and applying Cole's parametric search [10] on our new parallel decision algorithm, we solve the optimization problem in $O(n \log n \log \log n)$ time.

Our algorithm relies on Lemma 5, whose proof is quite independent of the remainder of this section and is given in the full paper. Note that Hershberger and Suri [19] gave a linear-time algorithm to achieve the same result as Lemma 5, which suffices for their purpose.

▶ **Lemma 5.** *Given the circular hull (with respect to a radius $r$) of a set $L$ of points and the circular hull of another set $R$ of points such that the points of $L$ and $R$ are separated by a line, one can do the following in $O(\log(|L|+|R|))$ time (assuming that the vertices of each circular hull are stored in a data structure that supports binary search): determine whether the circular hull of $L \cup R$ (with respect to $r$) exists; if yes, either determine which dominating case happens (i.e., all points of a set are contained in the circular hull of the other set) or compute the two common tangents between the circular hulls of $L$ and $R$.*

For any $0 \le i \le j \le n$, let $S^+[i,j] = \{p_i, p_{i+1}, \ldots, p_j\}$ and $S^-[i,j] = \{q_i, q_{i+1}, \ldots, q_j\}$. By using Lemma 5, we have the following lemma.

▶ **Lemma 6.** *We can preprocess $S$ and compute an interval $(r_1, r_2]$ containing $r^*$ in $O(n \log n)$ time so that given any $r \in (r_1, r_2)$ and any pair $(i,j)$ with $1 \le i \le j \le n$, we can determine whether $\alpha_r(S^+[i,j])$ (resp., $\alpha_r(S^-[i,j])$) exists, and if yes, return the root of a balanced binary search tree representing the circular hull, in $O(\log k \log\log k)$ parallel steps using $O(\log k)$ processors, or in $O(\log^2 k)$ time using one processor, where $k = j - i + 1$.*

**Proof.** As in [6, 16], we use the following geometric transformation. For any point $p = (a,b)$, let $h(p)$ denote the halfspace $\{(x,y,z) : z \ge a^2 + b^2 - 2ax - aby\}$. Then, for any set $P$ of points in the plane, $(\tau(P))^2$ is the minimum of $x^2 + y^2 + z$ over all points $(x,y,z)$ in the polyhedron $\mathcal{H}(P) = \bigcap_{p \in P} h(p)$.

*Preprocessing.* We build a complete binary search tree $T^+$ on the set $S^+ = \{p_1, p_2, \ldots, p_n\}$ such that the leaves of $T^+$ from left to right storing the points of $S^+$ in their index order. Each internal node $v$ of $T^+$ stores a hierarchical representation [11] of the polyhedron $\mathcal{H}(P)$, where $P$ is the set of points stored in the leaves of the subtree rooted at $v$ ($P$ is called a *canonical subset*). Computing the polyhedrons of all internal nodes of $T^+$ can be done in $O(n \log n)$ time in a bottom-up manner using linear time polyhedra intersection algorithms [7, 8]. Similarly, we build a tree $T^-$ on the set $S^- = \{q_1, q_2, \ldots, q_n\}$.

Consider a vertex $v = (x,y,z)$ of $\mathcal{H}(P)$ for a canonical subset $P$ of $T^+$. Define $r(v) = \sqrt{x^2 + y^2 + z}$. Let $C$ be the set of the values $r(v)$ of all vertices $v$ of $\mathcal{H}(P)$ for all canonical subsets $P$ of $T^+$. Note that $|C| = O(n \log n)$. We find the smallest value $r(v) \in C$ such that $r^* \le r(v)$, and let $r_2$ denote such $r(v)$. The value $r_2$ can be found in $O(n \log n)$ using our linear time decision algorithm and doing binary search on $C$ using the linear time selection algorithm [5]. Next, we find the largest value in $C$ that is smaller than $r_2$, and let $r_1$ denote that value. By definition, $r^* \in (r_1, r_2]$ and $(r_1, r_2)$ does not contain any element of $C$.

Consider a canonical subset $P$ of $T^+$ and any $r \in (r_1, r_2)$. We construct $\mathcal{I}_r(P)$ for each canonical subset $P$ of $T^+$ by intersecting the facets of $\mathcal{H}(P)$ with the paraboloid $W(r) = \{(x,y,z) : x^2 + y^2 + z = r^2\}$ and projecting them vertically to the $xy$-plane. By the definitions of $r_1$ and $r_2$, the paraboloid $W(r)$ intersects the same set of edges of $\mathcal{H}(P)$ for all $r \in (r_1, r_2)$; this implies that $\mathcal{I}_r(P)$ is combinatorially the same for all $r \in (r_1, r_2)$. Hence, we can consider $\alpha_r(P)$, which is the dual of $\mathcal{I}_r(P)$, as a parameterized circular hull of $P$. We store the (parameterized) vertices of $\alpha_r(P)$ in a balanced binary search tree. Since $\mathcal{H}(P)$ is convex, we can obtain $\mathcal{I}_r(P)$ and thus the balanced binary search tree for $\alpha_r(P)$ in $O(|P|)$ time; we associate the tree at the node of $T^+$ for $P$. Because the total size of $\mathcal{H}(P)$ for all canonical subsets $P$ in $T^+$ is $O(n \log n)$, we can obtain the balanced binary search trees for $\alpha_r(P)$ of all canonical subsets $P$ in $T^+$ in $O(n \log n)$ time.

We do the same for $T^-$ as above, which will obtain two values $r_1'$ and $r_2'$ correspondingly as above $r_1$ and $r_2$. We update $r_1 = \max\{r_1, r_1'\}$ and $r_2 = \min\{r_2, r_2'\}$; so $r^* \in (r_1, r_2]$ still holds. This finishes our processing on $S$, which takes $O(n \log n)$ time and is independent of $r$.

*Queries.* Given any $r \in (r_1, r_2)$ and any pair $(i, j)$ with $i < j$, we determine whether $\alpha_r(S^+[i, j])$ exists, and if yes, return the root of a balanced binary search tree representing it, as follows (the case for $S^-[i, j]$ is similar). Let $k = j - i + 1$ and let $P = S^+[i, j]$.

By the standard method, we first find $O(\log k)$ canonical subsets of $T^+$ whose union is exactly $S^+[i, j]$. Our following computation procedure can be described as a complete binary tree $T$ where the leaves corresponding to the above $O(\log k)$ canonical subsets. So $T$ has $O(\log k)$ leaves, and its height is $O(\log \log k)$. For each leave of $T$, its circular hull is already available due to the preprocessing. For each internal node $v$ that is the parent of two leaves, we compute the circular hull of the union of the two subsets $P_1$ and $P_2$ of the two leaves. As the points of $S^+$ are ordered radially by $o$, the two subsets are separated by a line through $o$. Hence, we can find the common tangents (if exist) using Lemma 5 in $O(\log k)$ time because the size of each subset is no more than $k$. Recall that the circular hull of each canonical subset is represented by a balanced binary search tree. After having the common tangents, we split and merge the two balanced binary search trees to obtain a balanced binary search tree for $\alpha_r(P_1 \cup P_2)$. In addition, we keep unaltered the two original trees for $\alpha_r(P_1)$ and $\alpha_r(P_2)$ respectively, and this can be done by using persistent data structures (e.g., using the copy-path technique [12, 25]) in $O(\log k)$ time. In this way, the original trees for $\alpha_r(P_1)$ and $\alpha_r(P_2)$ can be used in parallel for other computations. If the algorithm detects that $\alpha_r(P_1 \cup P_2)$ does not exist, then we halt the algorithm and report that $\alpha_r(S^+[i, j])$ does not exist. Also, if the algorithm finds that a dominating case happens, e.g., the $P_1$-dominating case, then $\alpha_r(P_1 \cup P_2) = \alpha_r(P_1)$ and thus we simply return the root of the tree for $\alpha_r(P_1)$.

We do this for all internal nodes in the second level of $T$ (i.e., the level above the leaves) in parallel by assigning a processor for each node. As $T$ has $O(\log k)$ leaves, we can compute the circular hulls for the second level in $O(\log k)$ parallel steps using $O(\log k)$ processors. Then, we proceed on the third level similarly. At the root of $T$, we will have the root of a balanced binary search tree for $\alpha_r(P)$. Using $O(\log k)$ processors, this takes $O(\log k \log \log k)$ parallel steps because each level needs $O(\log k)$ parallel steps and the height of $T$ is $O(\log \log k)$.

Alternatively, if we only use one processor, then since $T$ has $O(\log k)$ nodes and we spend $O(\log k)$ time on each node, the total time is $O(\log^2 k)$. ◀

Armed with Lemma 6, to determine whether $r^* \le r$, we use the algorithm framework in Theorem 4.2 of Chan [6], but we provide a more efficient implementation, as follows.

Recall the definitions of the matrices $A$ and $B$ in Section 2, and in particular, each row of $A$ (resp., $B$) is monotonically increasing while each column of $A$ (resp., $B$) is monotonically decreasing. For convenience, let $A[i, -1] = 0$ and $A[i, n+1] = B[i, -1] = \infty$ for all $0 \le i \le n$. Let $m = \lfloor n/\log^6 n \rfloor$. Let $j_t = t \cdot \lfloor n/m \rfloor$ for $t = 1, 2, \ldots, m-1$. Set $j_0 = -1$ and $j_m = n$. For each $t \in [0, m]$, find the largest $i_t \in [0, n]$ with $A[i_t, j_t] \ge B[i_t, j_t]$ (set $i_t = -1$ if no such index exists; note that $i_0 = -1$). Observe that $i_0 \le i_1 \le \cdots \le i_m$. Each $i_t$ can be found in $O(\log^7 n)$ time by binary search using Lemma 7. Hence, computing all $i_t$'s takes $O(n \log n)$ time. This is part of our preprocessing, independent of $r$.

▶ **Lemma 7** ([6, 16]). *After $O(n \log n)$ time preprocessing, $A[i, j]$ and $B[i, j]$ can be computed in $O(\log^6 n)$ time for any given pair $(i, j)$.*

Given $r > 0$, our goal is to decide whether $r^* \leq r$. Let $(r_1, r_2]$ be the interval obtained by Lemma 6. Since $r^* \in (r_1, r_2]$, if $r \leq r_1$, then $r^* > r$; if $r \geq r_2$, then $r^* \leq r$. It remains to resolve the case $r \in (r_1, r_2)$, as follows. In this case the result of Lemma 6 applies.

We will decide whether $r_i^* \leq r$ for all $i = 0, 1, \ldots, n$ (recall that $r^* \leq r$ iff some $r_i^* \leq r$). Let $t \in [0, m-1]$ such that $i_t < i \leq i_{t+1}$. If $A[i, j_t] > r$, then return $r_i^* > r$. Otherwise, find (by binary search) the largest $j \in [j_t, j_{t+1}]$ with $A[i, j] \leq r$, and return $r_i^* \leq r$ if and only if $B[i, j] \leq r$. See the pseudocode below. See Theorem 4.2 of [6] for the algorithm correctness.

**Algorithm 2** The decision algorithm of Theorem 4.2 by Chan [6].

1 Let $t \in [0, m-1]$ such that $i_t < i \leq i_{t+1}$;
2 **if** $A[i, j_t] > r$ **then** return $r_i^* > r$ ;
3 find the largest $j \in [j_t, j_{t+1}]$ with $A[i, j] \leq r$;
4 return $r_i^* \leq r$ iff $B[i, j] \leq r$;

Chan [6] implemented the algorithm in $O(\log n \log^2 \log n)$ parallel steps using $O(n \log n)$ processors. With Lemma 6, we provide a faster implementation of $O(\log n \log \log n)$ parallel steps using $O(n)$ processors. Line 1 can be done in $O(n)$ time as part of the preprocessing, independent of $r$. We first discuss how to implement Line 3 for all indices $i$, and we will show later that Lines 2 and 4 can be implemented in a similar (and faster) way.

For each $t = 0, 1, \ldots, m-1$, if $i_{t+1} - i_t \leq \log^6 n$, then we form a group of at most $\log^6 n$ indices: $i_t + 1, i_t + 2, \ldots, i_{t+1}$. Otherwise, starting from $i_t + 1$, we form a group for every consecutive $\log^6 n$ indices until $i_{t+1}$, so every group has exactly $\log^6 n$ indices except that the last group may have less than $\log^6 n$ indices. In this way, we have at most $2m$ groups, each of which consists of at most $\log^6 n$ consecutive indices in $(i_t, i_{t+1}]$ for some $t \in [0, m-1]$.

Consider a group $G = \{a, a+1, \ldots, a+b\}$ of indices in $(i_t, i_{t+1}]$. Note that $b < \log^6 n$. For each $i \in [a, a+b]$ such that $A[i, j_t] \leq r$, we need to perform binary search on $[j_t, j_{t+1}]$ to find the largest index $j$ with $A[i, j] \leq r$. To this end, we do the following. We compute the two circular hulls $\alpha(S^+[a+b, n])$ and $\alpha(S^-[1, j_t])$, in $O(\log n \log \log n)$ parallel steps using $O(\log n)$ processors by Lemma 6. Note that by "compute the two circular hulls", we mean that the two circular hulls are computed implicitly in the sense that each of them is represented by a balanced binary search tree and we have the access of its root. If $\alpha(S^+[a+b, n])$ (resp., $\alpha(S^-[1, j_t])$) does not exist, we set it to *null*. We do this for all $2m$ groups in parallel, which takes $O(\log n \log \log n)$ parallel steps using $O(m \log n) \in O(n)$ processors.

Consider the group $G$ defined above again. For each $i \in [a, a+b]$, we need to do binary search on $[j_t, j_{t+1}]$ for $O(\log(j_{t+1} - j_t)) = O(\log \log n)$ iterations. In each iteration, the goal is to determine whether $A[i, j] \leq r$ for an index $j \in [j_t, j_{t+1}]$. To this end, it suffices to determine whether $\alpha(U_{ij})$ exists. Notice that $U_{ij} = S^+[i+1, a+b-1] \cup S^+[a+b, n] \cup S^-[1, j_t] \cup S^-[j_t+1, j]$. $\alpha(S^+[a+b, n])$ and $\alpha(S^-[1, j_t])$ are already computed above. If one of them does not exist, then $\alpha(U_{ij})$ does not exist and thus $A[i, j] > r$. Otherwise, we compute the circular hull $\alpha(S^+[i+1, a+b-1])$, which can be done in $O(\log^2 \log n)$ time using one processor by Lemma 6 because $a + b - 1 - i \leq b - 1 \leq \log^6 n$. We also compute $\alpha(S^-[j_t + 1, j])$ in $O(\log^2 \log n)$ time using one processor. Then, we compute the common tangents of $\alpha(S^+[i+1, a+b-1])$ and $\alpha(S^+[a+b, n])$ by Lemma 5 (note that $S^+[i+1, a+b-1]$ and $S^+[a+b, n]$ are separated by a line through $o$), in $O(\log n)$ time using one processor. Then, we merge the two hulls with the two common tangents to obtain a balanced binary search tree for $\alpha(S^+[i+1, n])$. Because we want to keep the tree for $\alpha(S^+[a+b, n])$ unaltered so that it can participate in other computations in parallel, we use a persistent tree to represent it. Similarly, we obtain a tree for $\alpha(S^-[1, j])$, in $O(\log n)$ time using one processor. If one of $\alpha(S^+[i+1, n])$ and $\alpha(S^-[1, j])$

does not exist, then we return $A[i,j] > r$. Note that $S^+[a+b,n]$ and $S^-[1,j]$ are separated by $\ell$ and $U_{ij} = S^+[a+b,n] \cup S^-[1,j]$. By Lemma 5, we can determine whether $\alpha(U_{ij})$ exists in $O(\log n)$ time using one processor and consequently determine whether $A[i,j] \leq r$. Hence, the above algorithm determines whether $A[i,j] \leq r$ in $O(\log n)$ time using one processor.

If we do the above for all $i$'s in parallel, then we can determine whether $A[i,j] \leq r$ in $O(\log n)$ time using $n+1$ processors, for each iteration of the binary search. As there are $O(\log\log n)$ iterations, the binary search procedure (i.e., Line 3) for all $i = 0, 1, \ldots, n$ runs in $O(\log n \log\log n)$ parallel steps using $n+1$ processors.

For implementing Line 2, we can use the same approach as above by grouping the indices $i$ into $2m$ groups. The difference is that now each $i$ has a specific index $j$, i.e., $j = j_t$, for deciding whether $A[i,j] \leq r$, and thus we do not have to do binary search. Hence, using $n+1$ processors, we can implement Line 2 for all $i = 0, 1, \ldots, n$ in $O(\log n)$ parallel steps. We can do the same for Line 4. As a summary, we have the following theorem.

▶ **Theorem 8.** *After $O(n \log n)$ time preprocessing on $S$, given any $r$, we can decide whether $r^* \leq r$ in $O(\log n \log\log n)$ parallel steps using $O(n)$ processors.*

With the serial decision algorithm in Theorem 4 and applying Cole's parametric search [10] on the parallel decision algorithm in Theorem 8, the following result follows.

▶ **Theorem 9.** *The value $r^*$ can be computed in $O(n \log n \log\log n)$ time.*

**Proof.** Suppose there is a serial decision algorithm of time $T_S$ and another parallel decision algorithm that runs in $T_p$ parallel steps using $P$ processors. Then, Megiddo's parametric search [23] can compute $r^*$ in $O(PT_p + T_sT_p \log P)$ time by simulating the parallel decision algorithm on $r^*$ and using the serial decision algorithm to resolve comparisons with $r^*$. If the parallel decision algorithm has a "bounded fan-in or bounded fan-out" property, then Cole's technique [10] can reduce the time complexity to $O(PT_p + T_s(T_p + \log P))$. Like Chan's algorithm [6], our algorithm has this property because it mainly consists of $O(\log\log n)$ rounds of independent binary search (i.e., the algorithm of Lemma 5). In our case, $T_s = O(n)$, $T_p = O(\log n \log\log n)$, and $P = O(n)$. Thus, applying Cole's technique, $r^*$ can be computed in $O(n \log n \log\log n)$ time. ◀

▶ **Corollary 10.** *The planar two-center problem can be solved in $O(n \log^2 n)$ time.*

**Proof.** This follows by combining Theorem 9, which is for the nearby case, with the $O(n \log^2 n)$ time algorithm for the distant case [16]. ◀

## 5 The convex position case

In this section, we consider the case where $S$ is in convex position (i.e., every point of $S$ is a vertex of the convex hull of $S$). We show that our above $O(n \log n \log\log n)$ time algorithm can be applied to solving this case in the same time asymptotically.

We first compute the convex hull $CH(S)$ of $S$ and order all vertices clockwise as $p_1, p_2, \ldots, p_n$. A key observation [22] is that there is an optimal solution with two congruent disks $D_1^*$ and $D_2^*$ of radius $r^*$ such that $D_1^*$ covers the points of $S$ in a chain of $\partial CH(S)$ and $D_2^*$ covers the rest of the points. In other words, the cyclic list of $p_1, p_2, \ldots, p_n$ can be cut into two lists such that one list is covered by $D_1^*$ and the other list is covered by $D_2^*$.

Let $o$ be any point in the interior of $CH(S)$. By the above observation, there exists a pair of rays $\rho_1$ and $\rho_2$ emanating from $o$ such that $D_1^*$ covers all points of $S$ on one side of the two rays and $D_2^*$ covers the points of $S$ in the other side. To apply our previous algorithm, we need to find a line $\ell$ that separates the two rays. For this, we propose the following approach.

For any $i, j \in [1, n]$, let $S_{cw}[i, j]$ denote the subset of vertices on $CH(S)$ clockwise from $p_i$ to $p_j$, and $S_{cw}[i, j] = \{p_i\}$ if $i = j$. Due to the above observation, $r^* = \min_{i,j\in[1,n]} \max\{\tau(S_{cw}[i, j]), \tau(S_{cw}[j + 1, i - 1])\}$, with indices modulo $n$. For each $i \in [1, n]$, define $r(i) = \min_{h\in[i,i+n-1]} \max\{\tau(S_{cw}[i, h]), \tau(S_{cw}[h + 1, i - 1])\}$. Notice that as $h$ increases in $[1, n-1]$, $\tau(S_{cw}[1, h])$ is monotonically increasing while $\tau(S_{cw}[h+1, n])$ is monotonically decreasing. Define $k$ to be the largest index in $[1, n-1]$ such that $\tau(S_{cw}[1, k]) \leq \tau(S_{cw}[k+1, n])$.

▶ **Lemma 11.** *$r^*$ is equal to the minimum of the following values: $r(1)$, $r(k+1)$, $r(k+2)$, and $\max\{\tau(S_{cw}[i, j]), \tau(S_{cw}[j + 1, i - 1])$ for all indices $i$ and $j$ with $i \in [1, k]$ and $j \in [k + 2, n]$.*

**Proof.** Observe that $r^* = \min_{i,j\in[1,n]} \max\{\tau(S_{cw}[i, j]), \tau(S_{cw}[j+1, i-1])\} = \min_{1\leq h\leq n} r(h)$. Hence, $r^*$ is no larger than any of the values specified in the lemma statement.

Let $i$ and $j$ be two indices such that $r^* = \max\{\tau(S_{cw}[i, j]), \tau(S_{cw}[j + 1, i - 1])\}$ with $1 \leq i \leq j \leq n$. We claim that $r^* = r(i)$. Indeed, since $r^* = \min_{1\leq h\leq n} r(h)$, we have $r^* \leq r(i)$. On the other hand, as $r(i) \leq \max\{\tau(S_{cw}[i, j]), \tau(S_{cw}[j + 1, i - 1])\} = r^*$, we obtain $r(i) = r^*$. By a similar argument, $r^* = r(j + 1)$ also holds.

Without loss of generality, we assume that $r^* = \tau(S_{cw}[i, j]) \geq \tau(S_{cw}[j + 1, i - 1])$.

If $i \in [1, k]$ and $j \in [k+2, n]$, then the lemma follows. Otherwise, one of the following four cases must hold: $i = k + 1$, $j = k + 1$, $[i, j] \subseteq [1, k]$, and $[i, j] \subseteq [k + 2, n]$. If $i = k + 1$, then $r^* = r(k + 1)$. If $j = k + 1$, then $r^* = r(k + 2)$. Below we show that $r^* = r(1)$ if $[i, j] \subseteq [1, k]$ and we also show that the case $[i, j] \subseteq [k + 2, n]$ cannot happen, which will prove the lemma.

If $[i, j] \subseteq [1, k]$, then $\tau(S_{cw}[j+1, i-1]) \geq \tau(S_{cw}[k+1, n])$, for $S_{cw}[k+1, n] \subseteq S_{cw}[j+1, i-1]$. By the definition of $k$, we have $\tau(S_{cw}[k + 1, n]) \geq \tau(S_{cw}[1, k])$. Because $[i, j] \subseteq [1, k]$, $\tau(S_{cw}[1, k]) \geq \tau(S_{cw}[i, j])$. Combining the above three inequalities leads to the following: $\tau(S_{cw}[j+1, i-1]) \geq \tau(S_{cw}[k+1, n]) \geq \tau(S_{cw}[1, k]) \geq \tau(S_{cw}[i, j])$. Because $r^* = \tau(S_{cw}[i, j]) \geq \tau(S_{cw}[j + 1, i - 1])$, we obtain $r^* = \tau(S_{cw}[j + 1, i - 1]) = \tau(S_{cw}[k + 1, n]) = \tau(S_{cw}[1, k]) = \tau(S_{cw}[i, j])$. Notice that $r(1) \leq \max\{\tau(S_{cw}[1, k]), \tau(S_{cw}[k+1, n])\}$. Thus, we derive $r(1) \leq r^*$. Since $r^* \leq r(1)$, we finally have $r^* = r(1)$.

If $[i, j] \subseteq [k+2, n]$, then $\tau(S_{cw}[j+1, i-1]) \geq \tau(S_{cw}[1, k+1])$. By the definition of $k$, we have $\tau(S_{cw}[1, k+1]) > \tau(S_{cw}[k+2, n])$. Also, since $[i, j] \subseteq [k+2, n]$, $\tau(S_{cw}[k+2, n]) \geq \tau(S_{cw}[i, j])$ holds. Therefore, we obtain $\tau(S_{cw}[j + 1, i - 1]) \geq \tau(S_{cw}[1, k + 1]) > \tau(S_{cw}[k + 2, n]) \geq \tau(S_{cw}[i, j])$, which incurs contradiction since $r^* = \tau(S_{cw}[i, j]) \geq \tau(S_{cw}[j + 1, i - 1])$. Thus, the case $[i, j] \subseteq [k + 2, n]$ cannot happen. ◀

Based on the above lemma, our algorithm works as follows.

We first compute $r(1)$ and the index $k$. This can be easily done in $O(n \log n)$ time. Indeed, as $h$ increases in $[1, n - 1]$, $\tau(S_{cw}[1, h])$ is monotonically increasing while $\tau(S_{cw}[h + 1, n])$ is monotonically decreasing. Thus, $r_1^*$ and $k$ can be found by binary search on $[1, n - 1]$. As both $\tau(S_{cw}[1, h])$ and $\tau(S_{cw}[h+1, n])$ can be computed in $O(n)$ time, the binary search takes $O(n \log n)$ time. Similarly, we can compute $r(k + 1)$ and $r(k + 2)$ in $O(n \log n)$ time.

If $r^* \notin \{r(1), r(k + 1), r(k + 2)\}$, then $r^* = \max\{\tau(S_{cw}[i, j]), \tau(S_{cw}[j + 1, i - 1])$ for two indices $i$ and $j$ with $i \in [1, k]$ and $j \in [k + 2, n]$. We can compute it as follows. Let $\ell$ be a line through $v_{k+1}$ and intersecting the interior of $\overline{p_n p_1}$. Let $o$ be any point on $\ell$ in the interior of $CH(S)$. Lemma 11 implies $\ell$ and $o$ satisfy the property discussed above, i.e., $\ell$ separates the two rays $\rho_1$ and $\rho_2$. Consequently, we can apply our algorithm for Theorem 9 to compute $r^*$ in $O(n \log n \log \log n)$ time.

▶ **Theorem 12.** *The planar two-center problem for a set of $n$ points in convex position can be solved in $O(n \log n \log \log n)$ time.*

▶ Remark. The randomized result remarked after Theorem 4 also applies here: $r^*$ can be computed in $O(n)$ time with high probability after $O(n \log n)$ deterministic time preprocessing.

## References


1. P.K. Agarwal and J.M. Phillips. An efficient algorithm for 2D Euclidean 2-center with outliers. In *Proceedings of the 16th Annual European Symposium on Algorithms (ESA)*, pages 64–75, 2008.
2. P.K. Agarwal and M. Sharir. Planar geometric location problems. *Algorithmica*, 11:185–195, 1994.
3. P.K. Agarwal, M. Sharir, and E. Welzl. The discrete 2-center problem. *Discrete and Computational Geometry*, 20:287–305, 1998.
4. E.M. Arkin, J.M. Díaz-Báñez, F. Hurtado, P. Kumar, J.S.B. Mitchell, B. Palop, P. Pérez-Lantero, M. Saumell, and R.I. Silveira. Bichromatic 2-center of pairs of points. *Computational Geometry: Theory and Applications*, 48:94–107, 2015.
5. M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest, and R.E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7:448–461, 1973.
6. T.M. Chan. More planar two-center algorithms. *Computational Geometry: Theory and Applications*, 13:189–198, 1999.
7. T.M. Chan. A simpler linear-time algorithm for intersecting two convex polyhedra in three dimensions. *Discrete and Computational Geometry*, 56:860–865, 2016.
8. B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM Journal on Computing*, 21(4):671–696, 1992.
9. B. Chazelle and J. Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *Journal of Algorithms*, 21:579–597, 1996.
10. R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *Journal of the ACM*, 34(1):200–208, 1987.
11. D.P. Dobkin and D.G. Kirkpatrick. Determining the separation of preprocessed polyhedra – A unified approach. In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 400–413, 1990.
12. J. Driscoll, N. Sarnak, D. Sleator, and R.E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38(1):86–124, 1989.
13. M.E. Dyer. On a multidimensional search technique and its application to the Euclidean one centre problem. *SIAM Journal on Computing*, 15(3):725–738, 1986.
14. H. Edelsbrunner, D. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29:551–559, 1983.
15. D. Eppstein. Dynamic three-dimensional linear programming. *ORSA Journal on Computing*, 4:360–368, 1992.
16. D. Eppstein. Faster construction of planar two-centers. In *Proc. of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 131–138, 1997.
17. D. Halperin, M. Sharir, and K. Goldberg. The 2-center problem with obstacles. *Journal of Algorithms*, 42:109–134, 2002.
18. J. Hershberger. A faster algorithm for the two-center decision problem. *Information Processing Letters*, 1:23–29, 1993.
19. J. Hershberger and S. Suri. Finding tailored partitions. *Journal of Algorithms*, 3:431–463, 1991.
20. J. Jaromczyk and M. Kowaluk. An efficient algorithm for the Euclidean two-center problem. In *Proceedings of the 10th Annual Symposium on Computational Geometry (SoCG)*, pages 303–311, 1994.
21. M. Katz and M. Sharir. An expander-based approach to geometric optimization. *SIAM Journal on Computing*, 26(5):1384–1408, 1997.
22. S.K. Kim and C.-S. Shin. Efficient algorithms for two-center problems for a convex polygon. In *Proceedings of the 6th International Computing and Combinatorics Conference (COCOON)*, pages 299–309, 2000.
23. N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30(4):852–865, 1983.

**24** N. Megiddo. Linear-time algorithms for linear programming in $R^3$ and related problems. *SIAM Journal on Computing*, 12(4):759–776, 1983.

**25** N. Sarnak and R.E. Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29:669–679, 1986.

**26** M. Sharir. A near-linear algorithm for the planar 2-center problem. *Discrete and Computational Geometry*, 18:125–134, 1997.

**27** X. Tan and B. Jiang. Simple $O(n \log^2 n)$ algorithms for the planar 2-center problem. In *Proceedings of the 23rd International Computing and Combinatorics Conference (COCOON)*, pages 481–491, 2017.

**28** H. Wang and J. Xue. Improved algorithms for the bichromatic two-center problem for pairs of points. In *Proceedings of the 16th Algorithms and Data Structures Symposium (WADS)*, pages 578–591, 2019.