Finding Tailored Partitions

JOHN HERSHEYBERGER

DEC Systems Research Center, 130 Lytton Avenue, Palo Alto, California 94301

AND

SUBHASH SURI

Bell Communications Research, 445 South Street, Morristown, New Jersey 07960

Received May 4, 1989; revised June 28, 1990

We consider the following problem: given a planar set of points S, a measure μ acting on S, and a pair of values μ1 and μ2, does there exist a bipartition S = S1 ∪ S2 satisfying μ(Si) ≤ μi for i = 1, 2? We present algorithms for several natural measures, including the diameter (set measure), the area, perimeter, or diagonal of the smallest enclosing axes-parallel rectangle (rectangular measure), the side length of the smallest enclosing axes-parallel square (square measure), and the radius of the smallest enclosing circle (circular measure). The algorithms run in time \(O(n \log n)\) for the set, rectangle, and square measures, and in time \(O(n^2 \log n)\) for the circular measure. The problem of partitioning S into an arbitrary number \(k\) of subsets is known to be NP-complete for many of these measures.

1. Introduction

Cluster analysis deals with the problem of partitioning the elements of a set into subsets according to some optimization criterion. The goal of the analysis is to partition the set into homogeneous and well-separated classes. The terms cluster and separation are suggestive of the geometric nature of this problem: the elements of the set are generally represented as points in some feature space and the separation is measured by some distance function. Formally, we are given a set of points S in a \(d\)-dimensional space, with a measure μ acting on it. A large number of clustering problems have the following form: partition S into \(k\) subsets \(S_1, S_2, \ldots, S_k\), such that \(f(\mu(S_1), \mu(S_2), \ldots, \mu(S_k)) \leq \mu^*\), where \(\mu^*\) and \(k\) are part of the input, and \(f\) is some function of the measures of the subsets.
The choice of \( \mu \) and \( f \), which determine the optimization criterion, depends on the actual application. Some of the most common choices for the measure \( \mu \) are the diameter and the radius of the cluster. Despite its apparent simplicity, finding an optimal clustering is an intractably hard problem under either of these measures, even if such simple optimization criteria as the sum or the max are used, and the space is restricted to the two-dimensional plane. When we use the maximum radius as our criterion, we arrive at a well-known clustering problem in operations research, namely, the \( k \)-centers problem: given a set \( S \) of \( n \) points in the plane, an integer \( k \), and a real number \( r^* \), partition \( S \) into \( k \) subsets each of which can be enclosed by a disk of radius \( r^* \). The decision counterpart of this problem is known to be NP-complete for an arbitrary \( k \) (Megiddo and Supowit [11]). The related problem with the diameter measure also is intractable; the \( \text{diameter} \) is the largest distance between two points of the set. That is, given a planar set of points \( S \), an integer \( k \), and a real number \( d^* \), it is NP-complete to decide if \( S \) can be partitioned into \( k \) subsets \( S_1, S_2, \ldots, S_k \), such that \( \text{diameter}(S_i) \leq d^* \) for \( i = 1, 2, \ldots, k \) (Johnson [10]). The intractability under the \( \text{sum} \) measure was established by Brucker (see Garey and Johnson [8, p. 281]).

Consequently, many researchers have concentrated on finding partitions into a small, fixed number \( k \) of clusters. We also take this approach and consider the simplest case \( k = 2 \). Besides being of interest in its own right, an efficient algorithm for the bipartition clustering can also be used as a heuristic to obtain an arbitrary partitioning by recursive invocation. The 2-clustering problem for minimizing the maximum diameter was first considered by Avis [7], who proposed an \( O(n^2 \log n) \) time algorithm. The time complexity was later improved to optimal \( O(n \log n) \) by Asano, Bhattacharya, Keil, and Yao [1]. The problem of minimizing the sum of the two diameters was studied in Monma and Suri [12], they gave an \( O(n^2) \) time algorithm for the problem.

In this paper, we consider the 2-clustering problem where individual constraints are specified for each of the subsets, rather than a single constraint on their maximum or sum. Such a formulation can be used to perform a finer cluster analysis, since an analyst may wish to control the sizes of individual clusters. Consider, for instance, the following application, which is cited by Avis [2]. Given a collection of communication posts in the plane, we wish to partition the posts into two clusters so that the transmitters attached to the posts in each cluster can all communicate with each other (i.e., they form a clique). If the transmitters come in two varieties of transmission range, then we would like to find clusters that respect individual range constraints.

We consider the following abstract formulation of the problem. Given a set of points \( S \), a measure \( \mu \), and a pair of values \( \mu_1 \) and \( \mu_2 \), decide
if there exists a bipartition $S = S_1 \cup S_2$ that simultaneously satisfies $\mu(S_1) \leq \mu_1$ and $\mu(S_2) \leq \mu_2$. The measures we consider include the diameter of the set; the area, perimeter, and diagonal of the smallest enclosing rectangle with sides parallel to the coordinate axes; the side length of the smallest enclosing square with sides parallel to the coordinate axes; and the radius of the smallest enclosing circle. Our algorithms for the diameter, rectangle, and square measures take $O(n \log n)$ time, while the algorithm for the circle measure takes $O(n^2 \log n)$ time. All of our algorithms follow the same general scheme; their details vary according to the specific measure involved.

The following problem, originally considered by Avis [2], also falls within our framework: given a planar set of points $S$ and two pairs $(a, b)$ and $(c, d)$ of points from $S$, does there exist a partition $S = S_1 \cup S_2$ such that the Euclidean diameter of $S_1$ (resp. $S_2$) equals the distance between $a$ and $b$ (resp. $c$ and $d$)? We solve this problem in optimal $O(n \log n)$ time, improving Avis's $O(n^2)$ time solution.

In the course of developing our algorithms, we make use of circular hulls (also known as $\alpha$-hulls [6, 7]). These are similar to ordinary convex hulls, except their edges are arcs of circles. An attractive aspect of circular hulls is that they intersect with other circles in a highly restricted manner. We establish several properties of circular hulls that may have other applications as well (see Section 4).

We give an $O(n \log n)$ amortized time algorithm for maintaining a hull (either circular or ordinary convex hull) with online deletions. Chazelle[3] also achieves this bound in the context of the convex-layers problem; however, our method (1) is simpler, (2) applies to both ordinary convex hulls and circular hulls, and (3) allows vertices to be deleted in any arbitrary sequence, and not just in the sequence determined by the convex layers.

Sections 2 through 4 consider the diameter measure: Section 2 describes the general algorithm for the diameter measure, Section 3 presents the details of implementing this algorithm for the $L_1$ and $L_\infty$ metrics, and Section 4 describes the implementation for the Euclidean diameter. Rectangular and square measures are considered in Section 5; circular measure is considered in Section 6. Section 7 presents conclusions, open problems, and directions for future research.

2. **Diameter Measure: The General Algorithm**

Let $\mu(S)$ denote the $L_p$ diameter of the set $S$, where $p \in \{1, 2, \infty\}$. Without loss of generality, we assume that $\mu_1 = 1$, $\mu_2 = d$, and $d \geq 1$. We use $D(x, r)$ to denote the closed disk centered at point $x$ with radius $r$. 


under the $L_p$ metric; the metric will be clear from the context. We use $\overline{D}(x,r)$ to denote the complement of the disk. The following general algorithm works for all three metrics $L_1$, $L_2$, and $L_\infty$.

**Algorithm Diameter-Partition ($S$).**

1. Compute $\mu(S)$.
2. If $\mu(S) \leq d$ then
   3. Output $S_1 = \emptyset$, $S_2 = S$.
   Else
     4. Let $a$ and $b$ be two points of $S$ such that $\mu(S) = \mu((a,b))$.
       Do lines 4.1–4.4 twice, once with $s_1 := a$, $s_2 := b$ and once with $s_1 := b$, $s_2 := a$.
       4.1. $S_1 := S \cap \overline{D}(s_2,d)$ and $S_2 := S \cap D(s_2,d) \cap \overline{D}(s_1,1)$.
       4.2. $U := S \setminus (S_1 \cup S_2)$.
       4.3. Call Phase-1 $(S_1, S_2, U)$.
       4.4. If $\mu(S_i) \leq 1$ and $\mu(S_2) \leq d$ then
         call Phase-2 $(S_1, S_2, U)$ and output $S_1, S_2$.

End Algorithm.

If the diameter of the set is less than or equal to $d$, then the algorithm correctly finds the trivial bipartition in line 3. Otherwise, we use the inference that $a$ and $b$ must lie in different subsets. The remaining steps of the algorithm are therefore executed once for each of the two distinct ways of assigning $a$ and $b$ to $S_1$ and $S_2$. The motivating idea of line 4.1 is to initialize $S_1 := \{s_1\}$ and $S_2 := \{s_2\}$; the more complicated statement at line 4.1 makes additional assignments that are forced by the assignment of $s_1$ and $s_2$. The statement ensures that $U$ is contained in the intersection of $D(s_2,d)$ and $D(s_1,1)$, a property required by the implementation of Phase-1. The asymmetric definitions of $S_1$ and $S_2$ ensure that $S_1 \cap S_2 = \emptyset$.

The procedures Phase-1 and Phase-2, which assign the rest of the points, are specified below by pseudo-code.

**Procedure Phase-1 ($S_1, S_2, U$).**

$Q := S_1 \cup S_2$;
while $Q$ is non-empty do
  begin
    Remove a point $s$ from $Q$;
    if $s \in S_1$ then $r := 1$ else $r := d$;
    $N := U \cap \overline{D}(s,r)$; /* unassigned points outside $D(s,r)$ */
    $U := U \setminus N$;
    if $s \in S_1$ then $S_2 := S_2 \cup N$ else $S_1 := S_1 \cup N$;
    $Q := Q \cup N$;
  end
end Procedure.
FINDING TAILORED PARTITIONS

Finding tailored partitions

Phase-1 in action

**Procedure Phase-2** \((S_1, S_2, U)\).

Partition \(U\) into \(U_1\) and \(U_2\) such that \(\mu(U_1) \leq 1\) and \(\mu(U_2) \leq d\); /*always succeeds*/

\[ S_1 := S_1 \cup U_1 \quad \text{and} \quad S_2 := S_2 \cup U_2; \]

**end Procedure.**

Phase-1 propagates the constraints imposed by the initial assignment of \(a\) and \(b\). In the code, \(U\) is the set of points as yet unassigned. We place points in \(Q\) after they have been assigned to \(S_1\) or \(S_2\) but before we know the consequences of that assignment. When we remove points from \(Q\), we determine the consequences of their assignment. Figure 1 shows a generic step of Phase-1.

In Phase-2, we operate on a set of unassigned points, \(U\), that has the following property: the \(L_{\infty}\) distance from any point in \(U\) to any point of \(S_1\) (resp. \(S_2\)) is at most 1 (resp. \(d\)). The task of Phase-2 is to find a partition \(U = U_1 \cup U_2\) such that the individual sets respect the given constraints. The following two lemmas establish the correctness of these procedures.

**Lemma 2.1.** If \(\mu(S) > d\) and there is a bipartition \(S = S_1 \cup S_2\) satisfying \(\mu(S_1) < 1\) and \(\mu(S_2) \leq d\), then the assignment generated by one of the two calls to Phase-1 is consistent with it.

**Proof.** Phase-1 is executed only if the distance between the diametral pair \(a\) and \(b\) exceeds \(d\). Thus, \(a\) and \(b\) cannot both go in the same subset of the partition. Observe that, except for the initial assignment of \(a\) and \(b\), all assignments during Phase-1 are forced. Since we try both possible assignments of \(a\) and \(b\), the lemma follows. \(\Box\)
If the sets \( S_1 \) and \( S_2 \) generated by Phase-1 violate the diameter constraints for each of the two initial assignments of \( a \) and \( b \), then Diameter-Partition returns without success. By Lemma 2.1, no legal partition of \( S \) is possible and hence the algorithm performs correctly. On the other hand, if the sets \( S_1 \) and \( S_2 \) respect the diameter constraints, it is still possible that some unassigned points remain in \( U \). Phase-2 finds an appropriate partition of \( U \) and then returns \( S_1 \) and \( S_2 \). We shall show that Phase-2 of the algorithm always succeeds. That is, if Phase-1 finds a valid assignment of all points except those remaining in \( U \), then a complete partition is always possible.

**Lemma 2.2.** If the sets \( S_1 \) and \( S_2 \) generated by Phase-1 respect the diameter constraints and \( U = U_1 \cup U_2 \) is the partition found by Phase-2, then \( \mu(S_1 \cup U_1) \leq 1 \) and \( \mu(S_2 \cup U_2) \leq d \).

**Proof.** We consider the case \( S_1 \cup U_1 \); the other case is similar. Any two points of \( S_1 \) are within unit distance of each other, and any point of \( U \) is at most unit distance from any point of \( S_1 \). Thus, if \( \mu(S_1 \cup U_1) > 1 \) were to hold, the distance between two points of \( U_1 \) would have to exceed one. That is impossible, since by construction \( \mu(U_1) \leq 1 \).

Provided that the partition required by Phase-2 can be achieved, the algorithm Diameter-Partition either finds the desired bipartition or determines its impossibility. The actual partition of \( U \) in Phase-2 depends on the specific metric, as does the implementation of the while loop in Phase-1. The next two sections discuss them in detail.

A slight modification of the algorithm Diameter-Partition solves Avis's problem of specified diametral pairs. In that problem, two pairs \( \{a, b\} \) and \( \{c, d\} \) of points from \( S \) are specified to be diametral pairs of \( S_1 \) and \( S_2 \), respectively. Let \( ab \) and \( cd \) be the distances determined by the two pairs. We perform a modified version of lines 4.1–4.4 of Diameter-Partition, without performing lines 1–3. The modified version assigns each pair to its set, assigns the points outside the disks of the four points, and proceeds as in the original algorithm. That is, line 4.1 is changed to set \( S_1 := \{a, b\} \cup (S \cap \bar{D}(c, cd)) \cup (S \cap \bar{D}(d, cd)) \) and set \( S_2 \) similarly, but the other lines are unchanged. This alteration suffices to solve Avis's problem.

### 3. Diameter Measure: \( L_1 \) and \( L_\infty \) Implementation

In this section, we describe how to implement the algorithm Diameter-Partition under the \( L_1 \) or \( L_\infty \) diameter measure. Lines 1–3 of the algorithm are easily implemented to run in time \( O(n) \), since the diameter of \( S \) can be computed in this time. The later steps of the algorithm involve
intersections of disks. In \( L_w \), the disk \( D(x, r) \) is an axes-parallel square of side length \( 2r \) with center at \( x \). In \( L_1 \), this disk is a similar square with side length \( \sqrt{2}r \), rotated 45°. Therefore, we only need to describe the algorithm for the \( L_w \) metric; everything works for the \( L_1 \) metric by rotating the plane 45° and scaling it by a factor of \( \sqrt{2} \).

In Phase-1, the key step is to compute the set \( N \) at each iteration of the \textbf{while} loop. This set \( N \) is the intersection of the set \( U \) with the exterior of a disk. The set \( U \) consists of the points of \( S \) that lie inside an intersection of disks; initially, \( U \) is the set of points inside \( D(s_1, 1) \cap D(s_2, d) \). Observe that under the \( L_w \) metric, an intersection of disks (squares) is an axes-parallel rectangle, and so \( U \) is the portion of \( S \) contained in such a rectangle. For convenience, let this rectangle be denoted by \( R(U) \). At each iteration of the \textbf{while} loop, the set \( N \) consists of points that are inside \( R(U) \) but outside a square \( D(s, r) \) (see the third line of the \textbf{while} loop in Phase-1). If we maintain two lists of the points of \( U \), one sorted by \( x \)-coordinate and one by \( y \)-coordinate, then the set \( N \) can be extracted in time \( O(|N|) \), and the set \( U \) can be updated in the same time bound. Since each point is placed in \( N \) at most once, the complexity of Phase-1 is \( O(n) \) plus the time to sort.

In Phase-2, we have a set of points \( U \) that is left unassigned by Phase-1. The following lemma lets us find a legal partition of \( U \) in linear time.

\textbf{Lemma 3.1.} Let \( l \) be the shorter of the two axes-parallel segments that bisect \( R(U) \). Let \( U_1 \) and \( U_2 \) be the subsets of \( U \) on either side of \( l \). Then \( \mu(U_1) \leq 1 \) and \( \mu(U_2) \leq 1 \).

\textbf{Proof.} Since the \( L_w \) distance between \( s_1 \) and \( s_2 \) is greater than \( d \), \( s_1 \) does not lie in \( D(s_2, d) \), nor does \( s_2 \) lie in \( D(s_1, 1) \). Because \( R(U) \subseteq D(s_1, 1) \cap D(s_2, d) \), the longer side of \( R(U) \) has length at most 2, and the shorter side has length less than 1. Bisecting \( R(U) \) by \( l \) ensures that the diameter of each half is at most 1. This completes the proof. \( \square \)

This proves the following theorem.\(^1\)

\textbf{Theorem 3.2.} Given a set \( S \) of \( n \) points in the plane and two real numbers \( \mu_1 \) and \( \mu_2 \), we can constructively determine in \( O(n \log n) \) time whether there is a partition \( S = S_1 \cup S_2 \) such that the \( L_1 \) (\( L_w \)) diameter of \( S_i \) is at most \( \mu_i \), \( i = 1, 2 \).

In the next section, we consider the case of the Euclidean diameter.

\(^1\)The algorithm of Theorem 3.2 is not optimal. Linear algorithms have been devised by Z. Drezner [5], by Sumanta Guha [9], and by Günter Rote and Gerhard Woeginger (after the conference publication of this paper); see Section 7.
4. Diameter Measure: $L_2$ Implementation

4.1. Overview

In this section, we discuss the implementation of the algorithm Diameter-Partition for the Euclidean diameter. It is well known that the diameter of a planar set can be computed in time $O(n \log n)$ (see, e.g., Preparata and Shamos [16]). Therefore, excluding the time spent in Phase-1 and Phase-2, the algorithm Diameter-Partition runs in $O(n \log n)$ time. We now describe at a high level how to implement these two subroutines to run in time $O(n \log n)$.

The key step of Phase-1 is the iterated computation of the set $N$. This involves determining which points of $U$ lie outside the disk $D(s, r)$. Essentially, we need to maintain $U$ in a data structure that will allow the following operations:

D1. Given a point $s$ and a radius $r$, report all points $N \subseteq U$ that lie outside the disk $D(s, r)$.

D2. Delete the set of points $N$ from the data structure.

We would like a data structure with reporting and update cost $O(|N| \log n)$. The convex hull seems like a possible candidate, since we can use nested convex hulls to answer half-plane reporting queries in $O(\log n + k)$ time, where $k$ is the number of points reported [4]. Unfortunately, convex hulls do not seem to work for circle queries: detecting intersections between a convex polygon and a circle is too costly. Even with preprocessing allowed, we do not know how to find each intersection within a polylog time bound.

We circumvent this difficulty by introducing the notion of a circular hull. This is like a convex hull except its boundary consists of arcs of circles. The key property of a circular hull is that we can find its intersections with a circle quickly. We prove several interesting properties of circular hulls that parallel those of ordinary convex hulls. Our data structure maintains the circular hulls dynamically and performs operations (D1) and (D2) within the desired time bound.

In particular, the operation (D2) requires our data structure to support deletions from the circular hull at a cost of $O(\log n)$ per deletion. For ordinary convex hulls, the algorithm of Overmars and van Leeuwen [13] supports insertions and deletions in time $O(\log^2 n)$ each. In the case of deletions alone, this can be improved to $O(\log n)$ by adapting a technique of Chazelle [3]. We simplify and generalize Chazelle's method to obtain an $O(\log n)$ time deletion procedure for circular hulls.

Phase-2 finds a partition $U = U_1 \cup U_2$ such that the respective diameters of $U_1$ and $U_2$ are at most $1$ and $d$. Our construction method is quite
easy, and it finds this partition in linear time. We prove the correctness of the method by a geometric argument.

Our plan is as follows. In Subsection 4.2, we introduce circular hulls and establish the basic properties that make them useful to us. In Subsection 4.3, we discuss our data structure for maintaining circular hulls so that operations (D1) and (D2) are implemented within the time bounds stated above. Finally, in Subsection 4.4, we describe the partition of \( U \) that meets the requirements of Phase-2.

4.2. Circular Hulls

Before introducing the notion of circular hulls, we establish some preliminary facts concerning the interaction of a circle with the boundary of the intersection of several disks. Some definitions are needed. If \( a \) and \( b \) are two points in the plane, then \( ab \) denotes the Euclidean distance between them and \( 
\overline{ab} \) denotes the line segment joining them. If \( R \) is a closed, bounded region in the plane, then \( \partial R \) denotes the boundary of \( R \). A unit circle is a circle of radius \( r = 1 \). A unit disk is the closed region of the plane bounded by a unit circle. An arc of a unit circle is a unit-radius arc. A minor arc is one that subtends an angle of at most 180° at the center. The following proposition is quite straightforward.

**Lemma 4.1.** For any two points contained in a unit disk, there are precisely two unit-radius, minor arcs joining them. Both of these arcs are contained in the unit disk. If \( R \) is the common intersection of a finite number of unit disks, \( p \) and \( q \) are two points in \( R \), and \( A_1, A_2 \) are the two minor unit-radius arcs joining \( p \) and \( q \), then \( A_1 \cup A_2 \subset R \).

Let \( R \) be the common intersection of a finite number of unit disks: \( R = \bigcap_{i=1}^{k} D_i \). Let \( D \) be a disk of radius \( r \geq 1 \). We are interested in determining the maximum number of points in which \( \partial D \) can intersect \( \partial R \). If \( D = D_i \) for some \( i \), \( 1 \leq i \leq k \), then \( \partial D \cap \partial R \) consists of a unit-radius arc. We consider this the degenerate case, and pretend that \( \partial D \cap \partial R \) consists of the two endpoints of the common arc. It is the nondegenerate case, where \( D \) is distinct from all \( D_i \), that is of primary interest to us. The following lemma shows that \( \partial D \cap \partial R \) contains at most two points.

**Lemma 4.2.** Let \( R = \bigcap_{i=1}^{k} D_i \) be the common intersection of a finite number of unit disks. Let \( D \) be a disk of radius \( r \geq 1 \) that is distinct from all \( D_i \), \( i = 1, 2, \ldots, k \). Then the number of intersections between \( \partial D \) and \( \partial R \) is zero, one (if the intersection is tangential), or two (proper intersections).

**Proof.** Because \( \partial D \) and \( \partial R \) are both planar, closed curves, the number of proper intersections plus twice the number of tangential intersections is even. This quantity would have to be four or more if the lemma were false.
We consider the two ways of violating the lemma: (1) there is a tangential intersection and at least one more intersection, or (2) there are at least four consecutive proper intersections. In both cases we show that one of the arcs of $\partial D$ defined by the intersections with $\partial R$ is a minor arc that lies outside $R$. In case (1), since the curvature of $\partial R$ is at least as large as that of $\partial D$ at the point of tangency, $\partial D$ lies locally outside $\partial R$ on both sides of the tangential intersection, and at least one of the two arcs of $\partial D$ between the tangency and the other point(s) of intersection must be a minor arc. In case (2), at least two of the arcs incident to proper intersections must lie outside $R$, and one of these must be minor. Thus in either case (1) or case (2), we obtain a minor arc of $\partial D$ that lies outside $R$ and whose endpoints $x_1$ and $x_2$ are on $\partial R$. Because the radius of $D$ is at least one, this arc must lie inside the region bounded by the two unit-radius arcs from $x_1$ to $x_2$, and hence, by Lemma 4.1, must lie inside $R$, a contradiction. This completes the proof. □

Let $U$ be a finite set of points contained in a unit disk. The circular hull of $U$, denoted $\mathcal{E}(U)$, is the common intersection of all unit disks containing $U$. (Circular hulls are also known as $\alpha$-hulls [6, 7]. Edelsbrunner, Kirkpatrick, and Seidel considered the properties of $\alpha$-hulls when $\alpha$ (the circle radius) is variable [7]; we focus on the case $\alpha = 1$.) It is easy to see that $\mathcal{E}(U)$ is convex and unique. The vertices of $\mathcal{E}(U)$ are a subset of $U$ and the “edges” of $\mathcal{E}(U)$ are unit-radius arcs. Since $\mathcal{E}(U)$ can always be represented as the common intersection of at most $|U|$ unit disks, Lemma 4.2 applies to circular hulls.

**Corollary 4.3.** Let $U$ be a finite, planar set of points contained in a unit disk, and let $\mathcal{E}(U)$ be its circular hull. If $D$ is a disk of radius $r \geq 1$ that is not coincident with an arc of $\partial \mathcal{E}(U)$, then $|\partial D \cap \partial \mathcal{E}(U)| \leq 2$.

The case where $D$ is one of the unit disks contributing an arc to the boundary of $\mathcal{E}(U)$ is considered degenerate; as before, we pretend that the intersections occur at the two endpoints of the arc.

If $D$ is a unit disk, then the intersection of $\partial \mathcal{E}(U)$ and $\partial D$ has one further property crucial to our use of circular hulls:

**Lemma 4.4.** Let $\mathcal{E}$ be a circular hull, and let $D$ be a unit-radius disk. Either $\mathcal{E}$ is contained in $D$, or at least one vertex of $\partial \mathcal{E}$ lies outside $D$.

**Proof.** We show that if all the vertices of $\partial \mathcal{E}$ are inside $D$, then so is all of $\mathcal{E}$. The edge of $\partial \mathcal{E}$ joining any two consecutive vertices of $\partial \mathcal{E}$ is a unit-radius, minor arc, which by Lemma 4.1 lies inside $D$. Because every edge and every vertex of $\partial \mathcal{E}$ lies inside $D$, so does all of $\mathcal{E}$. □
Next we show that the intersection points \( \partial D \cap \partial \mathcal{E}(U) \) can be determined in logarithmic time. We do this in two steps, which are described in Lemmas 4.5 and 4.6, respectively.

**Lemma 4.5.** Let \( \mathcal{E} \) be a circular hull, and let \( v_0, v_1, \ldots, v_{m-1} \) be the clockwise list of its vertices. Let \( D \) be a disk of radius at least one. Let \( v_i, v_j \in \mathcal{E} \) be two vertices such that \( v_i \in D \) and \( v_j \notin D \). Then the set of points \( \partial D \cap \partial \mathcal{E} \) can be determined in time \( O(\log m) \).

**Proof.** Without loss of generality, assume that \( i \) is less than \( j \). We perform two separate binary searches on the sets of vertices \( A = \{ v_i, v_{i+1}, \ldots, v_j \} \) and \( B = \{ v_j, v_{j+1}, \ldots, v_i \} \) to locate the two intersection points. To perform the search on \( A \), we pick the middle vertex in \( A \), say, \( v_k \). If \( v_k \in D \), we discard \( v_{k-1}, v_{k+1}, \ldots, v_j \) from \( A \), otherwise we discard \( v_{k+1}, v_{k+2}, \ldots, v_j \) from \( A \). \( ^2 \) (Of course, we only maintain the indices delimiting the range of the vertices not yet discarded from \( A \).) After \( \lceil \log(j-i) \rceil \) tests, each of which takes \( O(1) \) time, the search stops with an index \( k' \), \( i \leq k' < j \), such that \( v_{k'} \in D \) and \( v_{k'+1} \notin D \). The exact point of intersection can now be determined in \( O(1) \) time by intersecting \( \partial D \) and one of the unit arcs passing through the point-pair \( v_{k'}, v_{k'+1} \). The search on the set \( B \) is carried out in a similar manner. The time bound follows.

**Lemma 4.6.** Let \( \mathcal{E} \) be a circular hull, and let \( v_0, v_1, \ldots, v_{m-1} \) be the clockwise list of its vertices. Let \( D \) be a disk of radius at least one. Let \( v_i, v_j \in \mathcal{E} \) be two vertices such that either \( v_i, v_j \notin D \) or \( v_i, v_j \in D \), but neither \( v_i \) nor \( v_j \) lies on \( \partial D \). Then at least one of the two circular chains \( \mathcal{E}_1 = (v_i, v_{i+1}, \ldots, v_j) \) and \( \mathcal{E}_2 = (v_j, v_{j+1}, \ldots, v_i) \) is not intersected by \( \partial D \), and we can distinguish this chain in time \( O(1) \).

**Proof.** Assume, without loss of generality, that \( v_i \) and \( v_j \) lie on a vertical line, with \( v_i \) below \( v_j \). Let \( D_i \) (resp. \( D_j \)) be the unit disk that contributes the arc between \( v_i \) and \( v_{i+1} \) (resp. \( v_j \) and \( v_{j+1} \)) to \( \partial \mathcal{E} \). Let \( (l, r) = \partial D_i \cap \partial D_j \) such that the clockwise ordering of the points is \( v_i, l, v_j, r \). Let \( A_i \) and \( A_j \) be the two unit-radius minor arcs between \( v_i \) and \( v_j \) such that \( A_i \) is closer to \( l \). Let us also label the arcs along the boundary of \( D_i \cap D_j \) as \( A_1, A_2, A_3, A_4 \), going clockwise from \( v_i \). See Fig. 2. Now, notice that by definition

\[ \mathcal{E} \subset D_i \cap D_j. \]

Furthermore, the circular chain \( \mathcal{E}_1 = (v_i, v_{i+1}, \ldots, v_j) \) is contained in the region bounded by \( A_1, A_2, \) and \( A_i \). Similarly, \( \mathcal{E}_2 = (v_j, v_{j+1}, \ldots, v_i) \) is contained in the region bounded by \( A_3, A_4, \) and \( A_i \). We will show that if

\(^2\) All operations on the indices are modulo \( m \).
$D$ intersects one of these regions, it cannot intersect the other. Since the radius of $D$ is at least one, $D$ cannot be contained in $D_i \cap D_j$. Hence, if $\partial D \cap \partial C \neq \emptyset$, then $\partial D$ must intersect the boundary of $D_i \cap D_j$, namely, $\bigcup_{i=1}^{n} A_i$. By Lemma 4.2, the number of intersections between $\partial D$ and $\partial(D_i \cap D_j)$ is one or two. But $\partial D$ cannot intersect both $A_1 \cup A_2$ and $A_3 \cup A_4$, since that would contradict the hypothesis that $v_i, v_j$ are either both in or both out of $D$. Similarly, $\partial D$ can intersect at most one of $A_i$ and $A_j$. Now we are done: if $\partial D \cap (A_1 \cup A_2) \neq \emptyset$, then $\partial D$ cannot intersect $C_2 = (v_j, v_{j+1}, \ldots, v_i)$, and if $\partial D \cap (A_3 \cup A_4) \neq \emptyset$, then $\partial D$ cannot intersect $C_1 = (v_i, v_{i+1}, \ldots, v_j)$. Since both the tests can be performed in $O(1)$ time, the lemma follows. 

Using the previous two lemmas, we can now design a procedure for determining the intersection points of $\partial C$ and the disk $D$.

**Lemma 4.7.** Let $\partial C$ be a circular hull with $m$ vertices, given as an ordered list of vertices along the boundary and stored in a structure that supports binary search. Let $D$ be a disk of radius $r \geq 1$. Then the set of points $\partial D \cap \partial C$ can be determined in time $O(\log m)$.

**Proof.** We choose two vertices $v_i, v_j \in \partial C$ such that there are roughly an equal number of vertices on either side of $v_i v_j$. We consider two cases. First, suppose that at least one of the two points, say $v_i$, lies on $\partial D$. The point $v_i$ is obviously an intersection point, and we only need to check if there is another one. Observe that at least one of the two neighbors of $v_i$ lies strictly inside $D$. Without loss of generality, let $v_{i-1}$ be that neighbor. If $v_{i+1}$ lies inside $D$, then $v_i$ is the only intersection between $\partial D$ and $\partial C$ (cf. Lemma 4.2); if $v_{i+1}$ lies on $\partial D$, then the intersection points are $v_i$ and $v_{i+1}$ (cf. Lemma 4.2), and if $v_{i+1}$ lies outside $D$, then we can find the other intersection point by using Lemma 4.5, since we know two points $v_{i-1}$ and $v_{i+1}$, one inside $D$ and the other outside. In the second case, suppose that neither $v_i$ nor $v_j$ lies on $\partial D$. If one of them lies inside and
the other outside $D$, then we can compute the intersection points $\partial D \cap \partial \mathcal{E}$ by Lemma 4.5. Otherwise, after an $O(1)$ time test, we discard half the vertices from further consideration using Lemma 4.6. We then solve the problem recursively for the remaining half. The time bound clearly follows.

In the next subsection, we show that the circular hull of a set of $n$ points can be constructed in $O(n \log n)$ time, and then be maintained dynamically as the points are deleted at an amortized cost of $O(\log n)$ time per deletion.

4.3. Maintaining a Circular Hull with Deletions

This section describes the data structure we use to represent the circular hull of a set of points $U \subseteq S$. The data structure supports two main operations:

1. Given a query circle of radius $r \geq 1$, find its intersections with $\partial \mathcal{E}(U)$ in time $O(\log n)$.
2. Delete a point from $U$ and update the circular hull at an amortized cost of $O(\log n)$.

Using the data structure, we can implement the procedure Phase-1 to run in $O(n \log n)$ time. We describe how to implement Phase-1 now, then give the details of the data structure.

In the preceding section we defined a circular hull $\mathcal{E}(U)$ as the intersection of all unit-radius disks containing $U$. To implement Phase-1, we broaden this definition slightly to allow different radii. For a given radius $r$, we define $\mathcal{E}_r(U)$ to be the intersection of all radius-$r$ disks containing $U$ (an $r$-hull with $\alpha = 1/r$). In our implementation of Phase-1, we maintain the two circular hulls $\mathcal{E}_r(U)$ and $\mathcal{E}_s(U)$.

Line 4.1 of algorithm Diameter-Partition guarantees that $U$ lies inside a unit disk, so both circular hulls are well defined. In the procedure Phase-1, we want to identify all points of $U$ outside a given circle of radius $r = \{1, d\}$, then delete them from $U$. Our implementation combines these operations in a single loop. Each iteration of the loop first uses operation 1 to intersect $\partial \mathcal{E}_r(U)$, the hull composed of circles of radius $r$, with the circle. By Lemma 4.4, if any point of $U$ lies outside the circle, so does a vertex of $\partial \mathcal{E}_r(U)$. The loop deletes the hull vertices outside the circle and updates both $\mathcal{E}_r(U)$ and $\mathcal{E}_s(U)$ using operation 2. The loop is repeated until $\mathcal{E}_r(U)$ lies inside the circle. The amortized cost of executing the loop is $O((k + 1)\log n)$, where $k$ is the number of points deleted from $U$. Because only $n$ points can be deleted altogether, the total time spent in Phase-1 is $O(n \log n)$. 
In the remainder of this section we describe the data structure we use to support operations 1 and 2. Multiple radii are unimportant in this context, so we speak only of $\mathcal{E}(U) \equiv \mathcal{E}_i(U)$.

Our data structure is very similar to those of Overmars and van Leeuwen [13] and Chazelle [3]. It stores the input set of points $U$ in the leaves of a complete binary tree $T(U)$, with the points sorted by their $x$-coordinates. An internal node of $T(U)$ represents the circular hull of the leaves of its subtree. The subtree hulls of $T(U)$ are stored as circularly linked lists. Although the lists at nodes of $T(U)$ change when points are deleted, the underlying tree is completely static, as is the assignment of points to nodes. The tree is determined once and for all when the points are sorted by $x$-coordinate. Thus when a point of $U$ is deleted, some lists in $T(U)$ may become null.

The data structures to support operations 1 and 2 are largely independent. We describe the two structures in order. To support operation 1, we need to perform binary search on $\mathcal{E}(U)$. Therefore the linked list representing $\mathcal{E}(U)$, which is stored at the root of $T(U)$, is also represented as a standard balanced binary search tree, which we denote by $B(U)$. Each insertion or deletion of a vertex in the list at the root of $T(U)$ is accompanied by the same operation on $B(U)$. (A vertex $v$ is inserted into $B(U)$ when the deletion of some other vertex first causes $v$ to appear on $\mathcal{E}(U)$. Vertex $v$ is deleted from $B(U)$ when it is deleted from $U$.) Each insertion or deletion takes $O(\log n)$ time. We show below that there are only $2n$ such operations.

We use the search tree $B(U)$ to perform the binary searches of Lemmas 4.5 through 4.7. The algorithms specified by the lemmas repeatedly find the midpoint of an interval of vertices, then replace one endpoint of the interval by the midpoint. Each step performs comparisons involving the vertices bounding the interval (Lemma 4.5) and involving the arcs immediately clockwise of those vertices (Lemma 4.6). Each subtree of $B(U)$ represents an interval of vertices on $\mathcal{E}(U)$. At the root of each subtree, we store the constant amount of data needed to perform the comparisons for one search step of Lemma 4.5 or 4.6. When $B(U)$ changes due to an insertion or deletion, the data stored on a single root-to-leaf path must be updated; this takes logarithmic time. Given the search tree $B(U)$, finding intersections with $\mathcal{E}(U)$ simply involves a walk from the root to a leaf of $B(U)$. We have established the following lemma.

**Lemma 4.8.** Let $\mathcal{D}$ be a circular hull with $m$ vertices, stored using the search tree structure $B(U)$ described above. Let $D$ be a disk of radius at least one. Then the set of points $\partial D \cap \mathcal{E}$ can be determined in time $O(\log m)$. Furthermore, if $\mathcal{E}$ is changed by adding or deleting vertices, these changes can be reflected in the search tree in $O(\log m)$ time apiece.
We now describe the part of our data structure that supports operation 2, the deletion of points from \( U \). In the tree \( T(U) \), the sets of points associated with two sibling nodes are separable by a vertical line, so we begin by proving properties of circular hulls of separable sets of points. We then describe \( T(U) \), first giving its static structure, then giving the operations needed to create it initially and to maintain it as points are deleted from \( U \). We prove that creating and maintaining \( T(U) \) both take \( O(n \log n) \) total time.

In order to describe our data structure, we first need to establish some properties of circular hulls. Key to our approach is the following straightforward observation.

**Lemma 4.9.** Let \( A \) and \( B \) be disjoint sets of points. If \( v \in A \) is on \( \partial \mathcal{C}(A \cup B) \), then \( v \) is also on \( \partial \mathcal{C}(A) \). If \( v_1, v_2 \in A \) are consecutive vertices of \( \partial \mathcal{C}(A \cup B) \), then the arc between them appears on both \( \partial \mathcal{C}(A \cup B) \) and \( \partial \mathcal{C}(A) \).

We now consider circular hulls of separable sets of points. Let \( F \) be the set of points associated with a node in \( T(U) \), and let \( L \) and \( R \) be the sets associated with its left and right children. Suppose that \( L \) and \( R \) are separated by the vertical line \( x = x\text{split} \). Unlike ordinary convex hulls, \( \mathcal{C}(L) \) and \( \mathcal{C}(R) \) are not necessarily disjoint. In fact, either of \( \mathcal{C}(L) \) and \( \mathcal{C}(R) \) may contain the other.

**Lemma 4.10.** At most one arc of \( \mathcal{C}(L) \) and one arc of \( \mathcal{C}(R) \) cross the vertical line \( x = x\text{split} \). The circular hull \( \mathcal{C}(L) \) contains \( R \) if and only if it contains \( \mathcal{C}(R) \). Symmetrically, \( \mathcal{C}(R) \) contains \( L \) if and only if it contains \( \mathcal{C}(L) \).

**Proof.** The endpoints of arcs are points of \( L \) and \( R \); neither \( L \) nor \( R \) has points on both sides of \( x = x\text{split} \). Because \( \mathcal{C}(L) \) and \( \mathcal{C}(R) \) are convex, at most one arc of each crosses \( x = x\text{split} \).

The reverse implication of the second claim is trivial. We prove the forward implication. Each point of \( \partial \mathcal{C}(R) \) lies on a minor arc between two points of \( R \). By Lemma 4.1, \( \mathcal{C}(L) \) contains both minor arcs between any two points inside it. Therefore \( \mathcal{C}(L) \) contains \( \partial \mathcal{C}(R) \) and a fortiori contains \( \mathcal{C}(R) \).

The following descriptive lemma exposes similarities between circular hulls and ordinary convex hulls.

**Lemma 4.11.** Let \( L \) and \( R \) be two sets of points separable by a vertical line \( x = x\text{split} \). If neither \( \mathcal{C}(L) \) nor \( \mathcal{C}(R) \) contains the other, then \( \partial \mathcal{C}(L \cup R) \) contains two arcs not in \( \partial \mathcal{C}(L) \) or \( \partial \mathcal{C}(R) \), both crossing \( x = x\text{split} \). The other arcs in \( \partial \mathcal{C}(L \cup R) \) form two contiguous chains, one from \( \partial \mathcal{C}(L) \) and one from \( \partial \mathcal{C}(R) \).
Proof. By Lemma 4.10, \( \partial \mathcal{C}(L \cup R) \) must have vertices from both \( L \) and \( R \). It is convex, so it has only two edges crossing \( x = x_{\text{split}} \). Each has one endpoint in \( L \) and one in \( R \). All other arcs on \( \partial \mathcal{C}(L \cup R) \) have both endpoints in \( L \) or in \( R \). Those arcs must be edges of \( \partial \mathcal{C}(L) \) or \( \partial \mathcal{C}(R) \) by Lemma 4.9. Since they form two contiguous chains of \( \partial \mathcal{C}(L \cup R) \), they must also be contiguous on \( \partial \mathcal{C}(L) \) and \( \partial \mathcal{C}(R) \). □

The two arcs described in the previous lemma are the outer common tangents, in the circular sense, of \( L \) and \( R \).

Armed with the preceding lemmas, we can now describe the lists stored at the nodes of \( T(U) \). Each node \( f \) in \( T(U) \) represents the circular hull of the points at the leaves of the subtree rooted at \( f \). We would like node \( f \) to store a doubly linked list of the vertices of this circular hull. However, since each point of \( U \) might be a hull vertex at each level of the tree, this scheme could require \( \Omega(n \log n) \) space. Therefore we use an idea due to Overmars and van Leeuwen [13] and store each point at the highest level in the tree at which it appears on a circular hull. Let \( F \) be the set of points associated with node \( f \), and let \( l \) and \( r \) be associated with \( l \) and \( r \), the children of \( f \). To build the list for \( \partial \mathcal{C}(F) \), once the tangents between \( \mathcal{C}(L) \) and \( \mathcal{C}(R) \) are known, we cut the two lists for \( \partial \mathcal{C}(L) \) and \( \partial \mathcal{C}(R) \) into two chains apiece, one that contributes to \( \partial \mathcal{C}(F) \) and one that does not. The chains that do not contribute to \( \partial \mathcal{C}(F) \) are stored at \( l \) and \( r \), while the chains that do contribute are linked to form the chain for \( \partial \mathcal{C}(F) \) and stored at \( f \). A similar scheme applies when \( \partial \mathcal{C}(F) \) is equal to \( \partial \mathcal{C}(L) \) or \( \partial \mathcal{C}(R) \). Note that this list storage scheme retains very few complete lists; the only list guaranteed to be complete is the one at the root of the tree \( T(U) \). Therefore node \( f \) also stores enough information to recreate the lists for \( \partial \mathcal{C}(L) \) and \( \partial \mathcal{C}(R) \) from the fragments at \( l \) and \( r \) and the list for \( \partial \mathcal{C}(F) \). This capability is used in the algorithm for deleting points of \( U \).

Lemma 4.12. The data structure \( T(U) \) can be built initially in \( O(n \log n) \) time.

Proof. The construction uses a divide-and-conquer algorithm. To build the linked list for the root of \( T(U) \) the algorithm builds the lists for the children of the root recursively, then combines them. The hard part of the combination step is determining whether either the left or right circular hull contains the other, and finding their common tangents if neither does. Once the tangents are known, the required list surgery takes only constant time. Our methods for detecting containment and finding common tangents run in time linear in the sizes of the lists involved. Thus the running time of the construction algorithm is given by the recurrence \( f(n) = 2f(n/2) + O(n) \), which has solution \( f(n) = O(n \log n) \).
As in Lemmas 4.10 and 4.11, let $L$ and $R$ be the point sets associated with the two children of a node in $T(U)$. By Lemma 4.10, $\mathcal{E}(L)$ contains $\mathcal{E}(R)$ if and only if all the vertices of $\partial\mathcal{E}(R)$ are contained in the supporting circle of the rightmost arc of $\partial\mathcal{E}(L)$. We test this condition in linear time. Similarly, we test whether $\mathcal{E}(R)$ contains $\mathcal{E}(L)$. If neither contains the other, then by Lemma 4.11, common tangents exist.

To find the common tangents between $\partial\mathcal{E}(L)$ and $\partial\mathcal{E}(R)$, we use the technique of Preparata and Hong [15] as described by Chazelle [3]. Let $v$ be the rightmost point of $R$. Pick a vertex of $\partial\mathcal{E}(L)$ and find the unit circle containing $L$ and tangent to $e(L)$ at that vertex. Now "roll" the tangent circle around the boundary of $\mathcal{E}(L)$ until (1) it passes through $v$, and (2) the upper arc from $v$ to $\partial\mathcal{E}(L)$ is a minor arc. Now roll the circle around $\partial\mathcal{E}(L)$ counterclockwise while watching the movement of the intersection of the circle with $\partial\mathcal{E}(R)$. When the circle becomes tangent to $\partial\mathcal{E}(R)$, it is the upper tangent between $\partial\mathcal{E}(L)$ and $\partial\mathcal{E}(R)$. See Fig. 3. This algorithm takes $O(|L| + |R|)$ time.

The algorithm for deleting a point $v$ from $U$ and updating $T(U)$ is based on those of Overmars and van Leeuwen [13] and Chazelle [3]. Like the initialization algorithm, it can be described recursively. Let $L$, $R$, and $F$ be the subsets of $U$ corresponding to two siblings and their parent in $T(U)$, and suppose $v = L$. To compute the list for $\partial\mathcal{E}(F \setminus \{v\})$, given the list for $\partial\mathcal{E}(F)$, the algorithm recreates the lists for $\partial\mathcal{E}(L)$ and $\partial\mathcal{E}(R)$ in constant time, recursively computes $\partial\mathcal{E}(L \setminus \{v\})$, and builds the list for $\partial\mathcal{E}(F \setminus \{v\})$ from $\partial\mathcal{E}(L \setminus \{v\})$ and $\partial\mathcal{E}(R)$. Finding the tangents prior to merging may take more than constant time. In Overmars and van Leeuwen’s approach, it takes $O(\log n)$ worst-case time, so each deletion takes $f(n) = f(n/2) + O(\log n) = O(\log^2 n)$ time altogether. We follow Chazelle’s example to achieve a constant-time average merge cost (but with a linear worst-case bound) that gives an $O(\log n)$ amortized cost per deletion.
To specify the tangent-finding algorithm and bound its running time, we must first describe what happens when a point \( u \) is deleted. By Lemma 4.9, every vertex \( u \) on \( \partial \mathcal{E}(F) \) that is not equal to \( v \) stays on \( \partial \mathcal{E}(F) \). If \( v \in \partial \mathcal{E}(F) \), then deleting \( v \) does not change \( \partial \mathcal{E}(F) \). If the vertices of \( \partial \mathcal{E}(F) \) before deletion of point \( u \) are \( \ldots, a, v, b, \ldots \), then the new \( \partial \mathcal{E}(F) \) replaces \( v \) by a possibly empty chain \( v_1, \ldots, v_k \), which consists of the points of \( F \) that were hidden by the arcs from \( a \) to \( v \) and \( b \) to \( v \). In what follows, we use \( F' \) and \( L' \) to denote \( F \) and \( L \) with \( u \) deleted from them.

**Lemma 4.13.** Let \( L, R \), and \( F \) be as above, and let \( u \) be a point of \( L \) that lies on \( \partial \mathcal{E}(F) \). If \( u \) is deleted, new vertices may be introduced into \( \partial \mathcal{E}(L') \) and \( \partial \mathcal{E}(F') \). The vertices that belong to \( \partial \mathcal{E}(F') \) but not to \( \partial \mathcal{E}(F) \) are either

- former neighbors of \( u \) on \( \partial \mathcal{E}(L) \),
- vertices that belong to \( \partial \mathcal{E}(L') \) but not to \( \partial \mathcal{E}(L) \), or
- vertices of \( \partial \mathcal{E}(R) \) adjacent to the chain of \( \partial \mathcal{E}(R) \) on \( \partial \mathcal{E}(F) \).

**Proof.** There are two tangents from \( \mathcal{E}(L) \) to \( \mathcal{E}(R) \), upper and lower. If neither is incident to \( u \), then exactly those vertices introduced into \( \partial \mathcal{E}(L') \) by the deletion of \( u \) are introduced into \( \partial \mathcal{E}(F') \).

If, say, the upper tangent is incident to \( u \), then the new upper tangent may expose new vertices on \( \partial \mathcal{E}(L') \). Suppose the sequence of vertices on \( \partial \mathcal{E}(L) \) before the deletion of \( u \) was \( \ldots, a, v, b, \ldots \). We show that the new upper tangent from \( \mathcal{E}(R) \) to \( \mathcal{E}(L') \) must have one endpoint in the interval \( a, \ldots, b \), inclusive. Let \( w \) be the new tangent endpoint on \( \partial \mathcal{E}(L') \). If \( w \) were outside \( a, \ldots, b \), then \( w \) would have belonged to \( \partial \mathcal{E}(L) \) before \( u \) was deleted. Then the upper tangent from \( w \) to \( \partial \mathcal{E}(R) \) would also have been the upper tangent before \( u \) was deleted, a contradiction.

Let \( \pi \) be the chain of vertices of \( \partial \mathcal{E}(R) \) that belong to \( \partial \mathcal{E}(F) \); by Lemma 4.9, \( \pi \) also belongs to \( \partial \mathcal{E}(F') \). By Lemma 4.11, any vertices of \( \partial \mathcal{E}(R) \) that belong to \( \partial \mathcal{E}(F') \setminus \partial \mathcal{E}(F) \) must be adjacent to \( \pi \). □

The vertices introduced onto \( \partial \mathcal{E}(F) \) by the deletion of \( u \) remain on \( \partial \mathcal{E}(F) \) until they are deleted themselves, as Lemma 4.9 implies. Phrased more generally, each point of \( U \) moves monotonically up \( T(U) \) until it is deleted. For each \( u \in U \), there is some leaf \( l \) in \( T(U) \) and an ancestor \( a \) of the leaf such that \( u \) is a vertex of the circular hulls associated with all the nodes on the path from \( l \) to \( a \), and no others; \( u \) is stored at \( a \). The point \( u \) moves up \( T(U) \) each time a deletion introduces it onto a new hull. Because \( T(U) \) has \( O(\log n) \) levels, there are \( O(n \log n) \) movements of points between levels of \( T(U) \). We use this fact to bound the total time we
spend on deletions: deleting a point takes \( O(\log n) \) time plus time proportional to the number of movements of points up \( T(U) \).

We now describe how to find the new tangents from \( \partial \mathscr{C}(L') \) to \( \partial \mathscr{C}(R) \), given that the deleted point \( v \) was an endpoint of one or both of the old tangent edges. We assume that \( v \) was not the only point of \( L \), since in that case the new \( \partial \mathscr{C}(F) \) is just equal to \( \partial \mathscr{C}(R) \).

Any tangent from \( R \) to \( L \) that does not have \( v \) as an endpoint appears on \( \partial \mathscr{C}(L' \cup R) \), by Lemma 4.9. Only the tangents with \( v \) as an endpoint need to be replaced. If \( v \) is an endpoint of only one tangent between \( \mathscr{C}(L) \) and \( \mathscr{C}(R) \), then the new tangent has one endpoint on \( \mathscr{C}(L') \) and one on \( \mathscr{C}(R) \). If \( v \) is an endpoint of both tangents, then one of two cases holds: either there are two new tangents, each with one endpoint on \( \mathscr{C}(L') \) and one on \( \mathscr{C}(R) \), or \( L' \) is contained in \( \mathscr{C}(R) \) and there are no new tangents at all. Our method determines which case holds, and, if \( L' \) is not contained in \( \mathscr{C}(R) \), finds the new common tangent(s).

The idea of our approach is to simulate a continuous movement of \( v \) into the interior of \( \mathscr{C}(L) \), computing the desired tangents in the process. Let \( a \) and \( b \) be the neighbors of \( v \) on \( \partial \mathscr{C}(L) \) before the deletion of \( v \). (Note that \( a \) and \( b \) may be equal.) If we move \( v \) continuously to the midpoint of \( ab \), then the tangents from \( v \) to \( \partial \mathscr{C}(F) \) move continuously until \( v \) disappears into the interior of \( \mathscr{C}(F) \), leaving the same \( \partial \mathscr{C}(F) \) we would get by deleting \( v \). Before we describe the algorithm in more detail, let us establish a basic lemma.

**Lemma 4.14.** The segment from \( v \) to the midpoint of \( \overline{ab} \) intersects \( \partial \mathscr{C}(L') \); let \( q \) be the point of intersection.

1. **If no vertex of** \( \partial \mathscr{C}(L' \cup R) \) **comes from** \( L' \), **then a single arc of** \( \partial \mathscr{C}(R) \) **crosses** \( x = x_{\text{split}} \) **and encloses** \( L' \). **This arc intersects the segment** \( \overline{vq} \).

2. **If at least one vertex of** \( \partial \mathscr{C}(L' \cup R) \) **comes from** \( L' \), **then each tangent arc from** \( \mathscr{C}(R) \) **to** \( \mathscr{C}(L') \) **that does not appear on the original hull** \( \partial \mathscr{C}(L' \cup R) \) **supports a circle that intersects** \( \overline{vq} \).

**Proof.** Because the midpoint of \( \overline{ab} \) is inside \( \mathscr{C}(L') \) and \( v \) is outside, \( q \) is well defined. If no vertex of \( \partial \mathscr{C}(L' \cup R) \) comes from \( L' \), then Lemmas 4.9 and 4.10 imply that a single arc of \( \partial \mathscr{C}(R) \) crosses \( x = x_{\text{split}} \) and encloses \( L' \). Because \( q \) is in \( \mathscr{C}(L') \), it is enclosed by the arc as well. The arc cannot enclose \( v \), since \( \mathscr{C}(R) = \mathscr{C}(L' \cup R) \cup \mathscr{C}(L \cup R) \). Hence the arc intersects \( \overline{vq} \). See Fig. 4a.

In the second case, the circle supporting the tangent encloses \( q \), as above, because it encloses all of \( \mathscr{C}(L') \). Since \( \mathscr{C}(L' \cup R) \) is a strict subset of \( \mathscr{C}(L \cup R) \), the circle cannot enclose \( v \); the old tangent edge passes through \( v \), the new one does not, and so \( v \) must lie outside its supporting circle. See Fig. 4b. □
A unit circle that contains \( L' \cup R \) and is tangent to both \( \mathcal{E}(L') \) and \( \mathcal{E}(R) \) is a common tangent of \( L' \) and \( R \). To find the new common tangent(s), we move \( v \) toward the midpoint of \( \overline{ab} \) while maintaining three or four unit circles passing through \( v \) and tangent to \( \mathcal{E}(L') \) or \( \mathcal{E}(R) \). More specifically, we maintain both circles that are tangent to \( \mathcal{E}(L') \), starting with the two circles that \( v \) defines with \( a \) and \( b \). We maintain either one or two circles tangent to \( \mathcal{E}(R) \), starting with the circles defined by the one or two arcs of \( \partial \mathcal{E}(L \cup R) \) from \( v \) to \( \mathcal{E}(R) \). As \( v \) moves, the circles roll around \( \mathcal{E}(L') \) and \( \mathcal{E}(R) \) to stay incident to \( v \).

We find the new tangent edges by detecting coincidences between the circles. This works because the circle supporting each new tangent edge must intersect \( \overline{vq} \), by Lemma 4.14. If we are maintaining two circles tangent to \( \mathcal{E}(R) \), and it is their coincidence we encounter first, then \( L' \) is contained in \( \mathcal{E}(R) \). The coincident circles support an edge of \( \partial \mathcal{E}(R) \). Otherwise, the first coincidence involves circles tangent to both \( \mathcal{E}(L') \) and \( \mathcal{E}(R) \) and determines a new tangent edge. If we are maintaining two circles tangent to \( \mathcal{E}(R) \), we continue to advance \( v \) toward \( q \) while maintaining the second circle tangent to \( \mathcal{E}(R) \) and the two circles tangent to \( \mathcal{E}(L') \). When we detect a second coincidence, we have found the second new tangent.

The following lemma shows that the tangent circles move monotonically along \( \partial \mathcal{E}(L') \) and \( \partial \mathcal{E}(R) \) as \( v \) advances.

**Lemma 4.15.** Let \( p \) be a point outside the circular hull \( \mathcal{E} \), but close enough to it that circular tangents from \( p \) to \( \mathcal{E} \) exist. If \( p \) moves along a
straight line into the interior of $C$, the endpoints of the circular tangents from $p$ to $C$ move monotonically along $\partial C$ until $p$ reaches $C$.

Proof. The ray along which $p$ moves always points inside the joint circular hull of $p$ and $C$, since the joint hull is convex and both $p$ and its destination lie inside. As long as $p$ is outside $C$, its direction of travel points to a consistent side of each circular tangent, which implies that the other endpoint of each circular tangent moves monotonically on $\partial C$. □

The proof of Lemma 4.13 implies that for each of the moving circles, each vertex that the point of tangency sweeps over, except the first, is either a new vertex of $\partial C(L)$ or a vertex of $\partial C(R)$ that appears for the first time on $\partial C(F)$. This is crucial to our running time bound.

We have so far described the computation as if $v$ moved continuously; in fact, the algorithm simulates the motion discretely. As $v$ moves, each of the three or four circles rotates about some vertex of $\partial C(L)$ or $\partial C(R)$. This vertex changes when the circle becomes coincident with the next circular hull arc incident to the vertex. For each circle, the algorithm notes the circular hull arc at which the vertex of rotation will change, as well as the intersection of $\bar{vq}$ with the circle supporting that arc. The algorithm also notes the position on $\bar{vq}$ where each pair of circles (rotating about their current vertices of rotation) would coincide. The simulation advances $v$ to the next such intersection or position of coincidence on $\bar{vq}$.

We have established the following lemma:

**Lemma 4.16.** We can find the common tangents between $C(L)$ and $C(R)$ after a point deletion in time $O(t + 1)$, where $t$ is the number of vertices that appear for the first time on $\partial C(L)$ or $\partial C(F)$.

This leads to the main result of this section.

**Theorem 4.17.** Given a set $U$ of $n$ points, we can compute a data structure in time $O(n \log n)$ such that (1) intersections of $\partial C(U)$ with a circle of radius $r \geq 1$ can be found in worst-case time $O(\log n)$, and (2) the data structure can be updated after a point deletion in amortized time $O(\log n)$.

Proof. The preceding discussion shows that $T(U)$ supports the desired operations; we simply prove the time bounds. Lemma 4.12 shows that the recursive algorithm for building $T(U)$ takes $O(n \log n)$ time. The search tree $B(U)$, which replicates the circular list at the root of $T(U)$, can be maintained in $O(n \log n)$ total time: by Lemma 4.9, each point of $V$ enters and leaves $\partial C(U)$ at most once, and each update to $B(U)$ takes $O(\log n)$ time.

We now prove the deletion time bound. Lemma 4.16 shows that each deletion takes $O(1)$ time at each level of $T(U)$, plus time proportional to
the number of points of $U$ that the deletion introduces onto a circular hull at that level or the one below. Because each point can move up to each level just once, the time to delete all $n$ points of $U$ is $n$ times the $O(\log n)$ deletion overhead, plus $O(n \log n)$ for all point movements. □

4.4. Partitioning $U$

In this section we show how to perform Phase-2, that is, to partition $U$ into two sets with diameters at most 1 and $d$, respectively. To do this, we need to take a closer look at the geometry of the set $U$. Without loss of generality, assume that the initial assignment is $s_1 = a$ and $s_2 = b$. Then $U$ lies in the common intersection of the disks $D(a,1)$ and $D(b,d)$. We call the region $D(a,1) \cap D(b,d)$ the lune. Because $ab > d$, the points $a$ and $b$ lie outside the boundary of the lune.

Let us consider the extremal case in which $a$ lies on the boundary of the lune. In this case $ab = d$, and the lune is as large as possible. If we can find the desired partition for this lune, we can find it for any lune; when $ab > d$, the resulting lune is a strict subset of the lune obtained by placing $b$ at distance $d$ from $a$ on the segment $ab$. Just for definiteness, we introduce a Cartesian coordinate system with $a$ at the origin and $b$ on the $y$-axis at position $(0,d)$. The upper boundary of the lune is an arc of the unit circle centered at the origin. The lower boundary of the lune is an arc from the circle of radius $d$ centered on point $b$; it passes through the origin. The lune is symmetric about the $y$-axis. We define the left endpoint of the lune to be $p$ and the right endpoint to be $q$. We draw an auxiliary unit-radius circle centered on $q$. This circle intersects the lune in two points, at $a$ and at a point $u$ on the upper arc on or to the right of $x = 0$. The intersection of the lune with the circle and its interior we call $R$, and the remainder of the lune's interior we call $L$. See Fig. 5.

We partition $U$ by putting the points of $U \cap R$ into $S_1$ and the points of $U \cap L$ into $S_2$. Thus we need to show that $R$ has diameter at most 1 and $L$ has diameter at most $d$.

**Lemma 4.18.** The region $R$ has diameter 1.

**Proof.** Consider the constant-thickness triangle (Reuleaux triangle) [17, pp. 70–82] defined by intersecting the three unit circles centered on $a$, $q$, and $u$. This region contains $R$: by definition $R$ lies inside the unit circles centered on $a$ and $q$; between $a$ and $q$, the boundary of $R$ has radius of curvature at least 1, which means that it lies inside the unit-radius arc from $a$ to $q$. Because $R$ has diameter at least 1 and lies inside a unit-diameter Reuleaux triangle, its diameter is exactly 1. □

**Lemma 4.19.** The region $L$ has diameter at most $d$. 
Proof. We first prove that the diameter of $L$ is $pu$. For any point $z$ in $L$, let us define $f(z)$ to be a point of $L$ at maximum distance from $z$. We show that $f(z) \in \{a, p, u\}$ for any $z$ in $L$. Because $f(z)$ must lie on the convex hull of $L$, it cannot lie in the interior of the arc joining $a$ to $u$. Suppose $f(z)$ lies in the interior of the arc from $p$ to $u$. The distance from $z$ to $f(z)$ must be a local extremum, so $z$ must lie on the segment from the origin to $f(z)$. But now we see that the distance from $z$ to $f(z)$ must be a local minimum, rather than a maximum: the circle centered on $z$ and passing through $f(z)$ lies inside the unit circle centered on the origin. Thus $f(z)$ cannot lie in the interior of the arc from $p$ to $u$. A similar argument shows that $f(z)$ cannot lie in the interior of the arc joining $p$ to $a$. Thus for any $z$ in $L$, $f(z)$ must be one of $a$, $p$, and $u$. For any diametral pair $i$ and $j$, we have $f(i) = j$ and $f(j) = i$, so the diameter of $L$ is one of $(ap, au, pu)$. But $ap = au = 1$, and $pu \geq 1$, so the diameter of $L$ is $pu$.

We now prove that $pu \leq d$. Consider the triangles $\triangle uaq$ and $\triangle upq$, shown in Fig. 6. Because $\triangle uaq$ is an equilateral triangle, $\angle uaq = 60^\circ$. Because $\angle upq$ is inscribed in the circle centered at $a$, its measure is $30^\circ$, or half that of $\angle uaq$. We can now apply the law of cosines to $\triangle upq$ to obtain

$$ (pu)^2 + (pq)^2 - 2(pu)(pq)\cos 30^\circ = (qu)^2. $$

But $qu = 1$, and some calculation shows that $pq = \sqrt{4 - 1/d^2}$. Thus the equation simplifies to

$$ (pu)^2 - (pu)\sqrt{12 - 3/d^2} + 3 - 1/d^2 = 0. $$
Solving the quadratic for $pu$ gives

$$pu = \frac{1}{2}\sqrt{12 - 3/d^2} \pm \frac{1}{2d}.$$ 

Plugging in $d = 1$ shows that the correct root is $pu = \frac{1}{2}\sqrt{12 - 3/d^2} - 1/2d$. By straightforward algebraic manipulation, we find that the inequality $pu \leq d$ holds if and only if $(d - 1/d)^2 \geq 0$, which proves the lemma.

This concludes our discussion of the bipartition problem for the Euclidean diameter measure. Our main result is stated in the following theorem.

**Theorem 4.20.** Given a set $S$ of $n$ points in the plane and two real numbers $\mu_1$ and $\mu_2$, we can constructively determine in $O(n \log n)$ time whether there is a partition of $S$ into sets $S_1$ and $S_2$ with Euclidean diameters at most $\mu_1$ and $\mu_2$, respectively.

In the next section, we consider rectangular and square measures.

5. **Rectangular and Square Measures**

In this section we give general definitions of rectangular and square measures, describe several natural examples of them, and show how to solve the partition problem for all of them.
We begin with rectangular measures, which are based on the notion of a bounding box. Given a set of points $X$, the bounding box of $X$, denoted $\Box(X)$, is the smallest axes-parallel rectangle that contains $X$. A rectangular measure $\mu(X)$ is a two-argument function of the height and width of $\Box(X)$, monotone nondecreasing in each argument. Rectangular measures of particular interest are the area, perimeter, and diagonal length of the bounding box.

We use a single strategy for all rectangular measures. As in the diameter case, we assign a few selected points to $S_1$ and $S_2$, then propagate the consequences of that assignment. In the diameter case, we assigned a diametral pair; in the rectangle case, the significant points are the determiners of $\Box(S)$. The bounding box of $S$ is determined by the topmost, bottommost, leftmost, and rightmost points of $S$, which we denote by $t$, $b$, $l$, and $r$. If several points of $S$ lie on a single side of $\Box(S)$, we arbitrarily pick one of them. Because a single point may determine two sides of $\Box(S)$, the points $t$, $b$, $l$, and $r$ may not all be distinct. We assume in the following that the determiners are distinct, but this is not a material restriction.

Let $\mu_1$ and $\mu_2$ be the specified measures, where $\mu_1 \leq \mu_2$. If $\mu(S) \leq \mu_1$, the algorithm returns $S_1 = \emptyset$, $S_2 = S$, as in the diameter case. Otherwise, the four determiners of $\Box(S)$ must be divided between $S_1$ and $S_2$. The algorithm considers all possible partitions of the four points between the two sets. There are essentially two different ways to partition the determiners: either one of the subsets gets three of the four determiners, or both subsets receive two determiners each.

Assigning three determiners to $S_1$ fixes three of the four sides of $\Box(S_1)$. We position the remaining side to make the rectangle as large as possible while staying within the measure $\mu_1$. (If the bounding box of the three points is already too large, then this assignment of determiners fails.) In linear time we test whether the points outside the rectangle have measure at most $\mu_2$, and return success if so. Combinatorially, there are eight distinct ways of assigning three points to $S_1$ and one to $S_2$ or vice versa.

There are six ways to assign two points to each of $S_1$ and $S_2$. In two of these, the points determine opposite sides of $\Box(S)$, and in four they determine adjacent sides.

We first consider the case in which two opposite sides of $\Box(S_1)$, say the top and bottom, are fixed by the assignment of determiners. The left and right sides of $\Box(S_2)$ are fixed. The fixed height of $\Box(S_1)$ determines a maximum width consistent with $\mu(S_1) \leq \mu_1$. Our algorithm (conceptually) slides a rectangle $R$ of this size from left to right over $S$, assigning all points within it to $S_1$, and all points outside it to $S_2$. See Fig. 7. At each sliding step the algorithm tests whether $\mu(S_2) < \mu_2$; $\mu(S_1)$ is guaranteed to be at most $\mu_1$. Any set $S_1 \subseteq S$ that contains $t$ and $b$ and satisfies
\( \mu(S_1) \leq \mu_1 \) must be contained in the rectangle \( R \) at some time during the sliding. Thus if none of the assignments to \( S_2 \) satisfies \( \mu(S_2) \leq \mu_2 \), then there is no partition that satisfies the measure constraints and has \( t, b \in S_1, l, r \in S_2 \).

To implement this algorithm, we maintain four priority queues, two for the highest and lowest points in \( S_2 \), one for the leftmost point of \( S_1 \) in \( R \), and one for the leftmost point of \( S_2 \) to the right of \( R \). At each sliding step, we transfer the leftmost point in \( R \) from \( S_1 \) to \( S_2 \), slide \( R \) to the right to the next leftmost point of \( S_1 \), and then transfer into \( S_1 \) the points of \( S_2 \) (except \( r \)) newly covered by \( R \). The priority queues for the top and bottom points of \( S_2 \) let us test in constant time whether \( \mu(S_2) \leq \mu_2 \). If there is a valid partition with \( t, b \in S_1 \) and \( l, r \in S_2 \), then \( \mu(S_2) \leq \mu_2 \) for some position of \( R \). The algorithm performs \( O(n) \) priority queue operations, for a total cost of \( O(n \log n) \).

In the final case, two adjacent sides of \( \square(S_1) \) are fixed, say the left and bottom sides. The bounding boxes of \( S \) and \( S_1 \) have the same lower left corner, denoted by \( z \). We use a method similar to the previous one to try all maximal assignments to \( S_1 \) that fix a corner of \( \square(S_1) \) at \( z \). Consider the set of rectangles with lower left corners at \( z \) and measure equal to \( \mu_1 \); their upper right corners lie on a particular curve \( \Gamma \). If \( \mu(X) \) is the area of \( \square(X) \), then \( \Gamma \) is a hyperbola; if \( \mu(X) \) is the perimeter of \( \square(X) \), then \( \Gamma \) is a segment with slope \(-1\); and if \( \mu(X) \) is the diagonal length of \( \square(X) \), then \( \Gamma \) is a quarter-circle of radius \( \mu_1 \) centered on \( z \). Figure 8 illustrates...
FINDING TAILORED PARTITIONS

Fig. 8. Sliding a changing rectangle.

this for the case of perimeter. All points of $S_1$ must lie inside $\Gamma$; in particular, if $l$ and $b$ do not, then their initial assignment to $S_1$ cannot lead to a valid partition.

We use a moving rectangle to look for valid partitions that have $l, b \in S_1, r, t \in S_2$. If there exists such a partition that satisfies the measure constraints, it must have $S_1$ fully contained in a rectangle with one corner at $z$ and another on $\Gamma$. To search for a partition, we consider all maximal sets $S_1$ obtained by intersecting $S \setminus \{r, t\}$ with such a rectangle. Our algorithm maintains the intersection of $S \setminus \{r, t\}$ with a shifting rectangle $R$ that has one corner at $z$ and another on $\Gamma$. We begin by identifying the set of points $Z \subset (S \setminus \{r, t\})$ on the same side of $\Gamma$ as $z$. The points of $S \setminus Z$ must belong to $S_2$. We initialize $R$ to have measure $\mu_1$ and to be as tall as possible while containing $l$ and $b$. We maintain four priority queues: two for the bottom and leftmost points of $S_2$, one for the top point of $S_1$, and one for the leftmost point of $Z$ to the right of $R$. At each step, we move the top point of $S_1$ into $S_2$, shift $R$ down to the next higher point of $S_1$ while keeping one corner on $\Gamma$, transfer any newly included points from $S_2$ to $S_1$, and test whether $\mu(S_2) \leq \mu_2$. We stop moving $R$ when $\mu(S_2) \leq \mu_2$ or when further movement would drop $l$ from $S_1$. This algorithm, like the preceding one, spends $O(n \log n)$ time on priority queue operations. This observation completes the proof of the following theorem.
THEOREM 5.1. Let $\mu$ be a rectangular measure with the property that $\mu(X)$ can be computed in constant time once the bounding box of $X$ is known. Given a set of points $S$ and two real numbers $\mu_1$ and $\mu_2$, we can determine in $O(n \log n)$ time whether there is a partition of $S$ into $S_1$ and $S_2$ satisfying $\mu(S_1) \leq \mu_1$ and $\mu(S_2) \leq \mu_2$.

Note that the priority queues used in the preceding algorithms may be replaced by cross-indexed sorted lists, if desired; we omit the details.

We now consider the case of square measure. A square measure $\mu(X)$ is a monotone function of the side length of the smallest axes-parallel square that contains $X$. Because a square measure is a function of a single argument, all square measures are equivalent from the standpoint of the partition problem. The side length of the smallest square containing $X$ is also the maximum horizontal or vertical distance between two points of $X$, that is, the $L_\infty$ diameter of $X$. Thus the square measure partition problem has already been solved in Section 3.

6. CIRCULAR MEASURE

We now consider the circle measure in $L_2$. The problem here is to find a partition $S = S_1 \cup S_2$ such that $S_1$ and $S_2$ can be enclosed by circles of given radii. If the radii are equal, this is the 2-centers problem. We consider two variants of the problem, one in which the circles are required to be centered at points of $S$, and one in which the circles can have arbitrary centers. Our method of the previous sections does not seem to apply to this problem directly. Nevertheless, we are able to give bounds that are better than the naïve ones: we can solve the case of constrained centers in $O(n^2)$ time, and the case of unconstrained centers in $O(n^2 \log n)$ time. Both algorithms require only linear space.

The circle measure algorithms are similar in spirit to the algorithm for rectangular measures: they consider all possible placements of one circle and test whether the points outside it can be covered by the second circle. In the case of constrained centers, there are only $n$ possible positions of the first circle; we test in linear time whether the remaining points can be covered. In the unconstrained case, there are $\Theta(n^2)$ combinatorially distinct positions of the first circle; by using circular hulls, we test each position in $O(\log n)$ amortized time.

For convenience, let us assume that the two radii specified by the circle measure problem are 1 and $r$. Both of our algorithms repeatedly place a circle of radius $r$, then test if the points outside it can be covered by a unit disk.
6.1. Constrained Circles

The algorithm for constrained circles is straightforward. We begin by sorting the points of $S$ from left to right in $O(n \log n)$ time. Then for each point $p$ of $S$, we determine the set $U$ (for uncovered) of points whose distance from $p$ is greater than $r$. This takes linear time. We then intersect all the unit disks centered at points of $U$, obtaining a region we denote by $\mathcal{A}(U)$. If $\mathcal{A}(U)$ is not empty and contains a point of $S$, then a unit circle centered at that point contains all of $U$. By using the sorted order of $S$, we compute $\mathcal{A}(U)$ and determine whether it contains anything in $O(n)$ time.

We compute $\mathcal{A}(U)$ incrementally, intersecting the circles in left-to-right order of their centers. We break the boundary of the intersection, $\partial \mathcal{A}(U)$, into two chains, upper and lower. It is not difficult to see that the left-to-right order of the arcs along the upper (or lower) chain of $\partial \mathcal{A}(U)$ is just the reverse of the left-to-right order of the centers. This means that we can construct the upper (or lower) boundary of $\mathcal{A}(U)$ in $O(n)$ time by a Graham scan [16, pp. 100–104]. If a new circle contributes to the chain at all, its arc appears at the left end of the chain, possibly removing some previously added arcs. Computing the new boundary takes constant time, plus time proportional to the number of arcs deleted. Hence the overall bound is $O(n)$.

Once we have found $\mathcal{A}(U)$, we test whether this region contains any point of $S$. This can be done in $O(n)$ time by marching through $S$ from left to right, maintaining the two arcs of $\partial \mathcal{A}(U)$ that overlap the $x$-coordinate of the current point. We sweep over each arc once, so the total cost is linear. This gives a quadratic bound for the constrained circles case.

6.2. Unconstrained Circles

The algorithm for unconstrained circles uses the circular hulls of Section 4. If we draw the arrangement of $n$ circles of radius $r$ centered on the points of $S$, each of the $O(n^2)$ faces of the arrangement corresponds to a subset of $S$ that can be covered by an $r$-disk. We want to determine, for each subset, whether its complement with respect to $S$ can be covered by a $1$-disk.

To save space, we do not build the whole arrangement, but instead explore it piecemeal. Because each region of interest lies inside at least one $r$-disk, we can explore all the regions by exploring the regions along the boundary of each $r$-disk in turn. Let $C$ be the current $r$-disk. We intersect all the other $r$-disks with $C$ and sort the intersections along $C$. Now we can walk along $C$, passing from face to face of the arrangement, maintaining the complement of the points corresponding to the current face. Each step from one face to the next adds or deletes a point of the
set. Altogether, the operations of intersecting, sorting, and walking take $O(n \log n)$ time for each $r$-disk.

Walking through the regions along $C$ corresponds to dragging the center of an $r$-disk along $C$ and maintaining the set of points outside the moving $r$-disk. At each step during the walk, we want to know whether the points outside the moving disk can be covered by a unit disk. A covering disk exists if and only if the intersection of unit disks centered on the points is nonempty.

It is not hard to see that we can drag the $r$-disk halfway around $C$ (180°) without re-covering any data point that was uncovered during the motion. We use this fact to compute the desired intersection efficiently. We maintain the intersections of two disjoint sets $A$ and $D$; $A$ is subject only to insertions and $D$ is subject only to deletions. We initialize $A$ to be empty and $D$ to contain all the data points outside the initial placement of the moving $r$-disk. As the disk moves, it covers new data points, which we delete from $D$, and uncovers others, which we add to $A$. As $A$ and $D$ change, we maintain $\mathcal{F}(A)$ and $\mathcal{F}(D)$, the intersections of the unit circles centered at their points. Whenever $\mathcal{F}(A)$ or $\mathcal{F}(D)$ changes, we compute $\mathcal{F}(A) \cap \mathcal{F}(D)$. If the result is nonempty, then the problem has a solution: any unit circle whose center lies in the intersection will cover the points outside the moving $r$-circle.

To cover the full 360° of possible placements for the moving $r$-disk, we repeat this process a second time, starting the moving disk from where it stopped the first time.

There are three subproblems we must solve to implement this scheme. We need to maintain the intersection of a set of unit disks subject to deletions, solve the same problem for insertions, and detect whether two intersections of unit disks intersect each other.

To solve the first subproblem, we use a point-circle duality under which the intersection of unit disks is equivalent to the circular hull of their centers. By definition, a circular hull is the intersection of unit disks containing a set of points $S$. The set of possible placements of the centers of these containing disks is exactly the intersection $\mathcal{F}(S)$ of the unit disks centered at the points. As we roll a unit circle along the boundary of the circular hull, its center traces out the boundary of the intersection $\mathcal{F}(S)$. Each arc of the circular hull maps to a vertex of the intersection, and each vertex maps to an arc. Thus $\mathcal{F}(S)$ is dual to $\mathcal{F}(S)$.

By duality, we can use the deletions-only data structure of Section 4 to maintain both $\mathcal{F}(D)$ and $\mathcal{F}(D)$ simultaneously. This idea has one flaw: our circular hull maintenance algorithm depends on the points lying inside a unit circle, but this may not be the case in the present setting. Thus we must extend our algorithm to handle cases in which no circular hull exists ($\mathcal{F}(D)$ is empty).
Let $T(D)$ be the tree that represents $\mathcal{E}(D)$, and let $N$ be the set of nodes of $T(D)$ whose associated point sets do not fit inside a unit circle. No leaf of $T(D)$ belongs to $N$. If a node belongs to $N$, so do all its ancestors. For each node $v$ in $N$ whose children are not in $N$, we maintain a witness to $v$'s membership in $N$. Let $L$ and $R$ be the point sets associated with the children of $v$. Our witness is a separating line between $\mathcal{F}(L)$ and $\mathcal{F}(R)$, namely an inner common tangent between the two regions. This tangent can be computed initially in $O(|L \cup R|)$ time when $T(D)$ is built. As points are deleted from $L$ and $R$, $\mathcal{F}(L)$ and $\mathcal{F}(R)$ expand, and the inner common tangent rolls monotonically along $\partial \mathcal{F}(L)$ and $\partial \mathcal{F}(R)$. When $\mathcal{F}(L)$ and $\mathcal{F}(R)$ finally intersect each other, the tangent disappears, and we compute $\mathcal{E}(L \cup R)$ in $O(|L \cup R|)$ time. Whenever a node leaves the set $N$, we check whether its sibling is in $N$; if not, we compute the circular hull for the two nodes, or the inner common tangent of their circle intersections, as appropriate. The computation and maintenance of tangents takes $O(n)$ time at every level of $T(D)$, and so the total maintenance cost is $O(n \log n)$.

To maintain $\mathcal{F}(A)$, we can either use a variation on Preparata’s convex hull algorithm [14], or we can use the circular hull data structure with deletion replaced by insertion. We can perform the data structure changes caused by an insertion in time $O(\log n + k)$, where $k$ is the number of point-level movements within the data structure. By the arguments of Section 4, this gives an $O(n \log n)$ overall maintenance bound. In this case we do not have to worry about point sets that cannot be contained in a unit disk: whenever $\mathcal{F}(A)$ becomes empty ($\mathcal{E}(A)$ ceases to exist), we can stop work, because the intersection $\mathcal{F}(A) \cap \mathcal{F}(D)$ will never again be nonempty.

At each step, the algorithm has to intersect $\mathcal{F}(A)$ and $\mathcal{F}(D)$. One possible approach is to extend a standard intersection-detection algorithm for convex polygons to work for intersections of circles. Alternatively, we can run an algorithm for detecting polygon intersection in $O(\log n)$ time, using the vertices of $\mathcal{F}(A)$ and $\mathcal{F}(D)$ as the polygon vertices. If the algorithm detects an intersection, then $\mathcal{F}(A)$ and $\mathcal{F}(D)$ intersect. If the algorithm does not detect an intersection, then it returns a separating line for the two polygons. Only one arc of $\mathcal{F}(A)$ and one arc of $\mathcal{F}(D)$ can cross the separating line, and we can find these arcs with two $O(\log n)$ binary searches. Then $\mathcal{F}(A)$ and $\mathcal{F}(D)$ intersect if and only if these arcs intersect.

There are $O(n)$ changes to $A$ and $D$, and so circular hull operations during the walk along $C$ take $O(n \log n)$ time. Multiplying this by the number of $r$-disks gives an $O(n^2 \log n)$ time bound for the full algorithm.

**Theorem 6.1.** Given $n$ points in the plane and two radii $r_1$ and $r_2$, we can determine in $O(n^2 \log n)$ time and linear space whether two disks of
radii \( r_1 \) and \( r_2 \) can be placed to cover all the points. If the disks are constrained to be centered at the data points, we need only \( O(n^2) \) time.

7. Conclusions

We have shown that a simple, general scheme can be adapted to solve many clustering problems that involve partitioning a set of points into two subsets subject to given constraints. The salient features of our scheme are to isolate a few points that, because of the given constraints, cannot all be placed into the same subset, and then to propagate constraints as new points are added to each of the two subsets. Conceivably, our technique may work for measures other than the ones discussed in this paper.

Our method of maintaining circular hulls under deletions can also be used for ordinary convex hulls (the case of infinite-radius circles). In this regard, our method has a few advantages over Chazelle's [3]. For instance, we do not need separate representations of upper and lower hulls, which eliminates the need for Chazelle's "cross-deletions." Our method supports deletions in any arbitrary order. Chazelle's method also has this feature, although his paper does not say so and it requires some extra thought to see this.

Finding partitions into an arbitrary number of subsets is probably quite hard under most of the measures we considered. In particular, the problem of deciding whether \( S \) can be partitioned into \( k \) subsets each of which has diameter less than some given bound is NP-complete (Johnson [10]); the problem remains NP-complete even for the \( L_1 \) or \( L_\infty \) metrics. Therefore, finding a partition into \( k \) subsets under the square measure is also intractable.

Our result for the Euclidean diameter is optimal, since computing the diameter itself takes \( \Omega(n \log n) \) time under the algebraic model of computation (see, e.g., Preparata and Shamos [16]). The algorithm for the \( L_1 \) and \( L_\infty \) metrics can be improved to linear, as has been noted independently by Sumanta Guha [9] and by Günter Rote and Gerhard Woeginger (personal communication). Both achieve an optimal linear running time by combining the bounding box ideas of Section 5 with the general diameter algorithm of Section 2. The same result was obtained earlier by Z. Drezner [5] (we thank Günter Rote for the reference). The optimality of the remaining algorithms remains unknown at this point.

References

9. S. Guha, An $O(n)$ time set bipartition algorithm under the $L_{\infty}$ metric, manuscript, EECS Department, University of Michigan, Ann Arbor, MI 48109-2122, 1989.