MAX-PLANCK-INSTITUT FÜR INFORMATIK

On intersection searching problems involving curved objects

Prosenjit Gupta Ravi Janardan Michiel Smid

MPI-I-93-124 June 1993

Im Stadtwald
66123 Saarbrücken
Germany

On intersection searching problems involving curved objects

Prosenjit Gupta Ravi Janardan Michiel Smid

MPI–I–93–124 June 1993

On Intersection Searching Problems Involving Curved Objects

Prosenjit Gupta* Ravi Janardan* Michiel Smid†

June 2, 1993

Abstract

Three classes of geometric intersection searching problems are considered, i.e., problems in which a set S of geometric objects is to be preprocessed into a data structure so that for any query object q, the objects of S that are intersected by q can be counted or reported efficiently. In the first class, S is a set of linear objects, such as lines or line segments, and q is a curved object, such as a circle, disk, or circular arc. In the second class, S is a set of curved objects, such as d-balls, d-spheres, circles, or circular arcs, and q is also a curved object. In the third class, which is a generalization of the first two, the objects in S are curved or linear and each is assigned a color. Given a query q, such as a disk or an annulus, the goal is to count or report the distinct colors of the objects intersected by q.

Efficient solutions are presented for a wide variety of problems from these classes. The solution techniques are based on geometric transformations, on compositions of known solutions for simplex range searching, on the locus approach, and on persistent data structures. Previously, efficient solutions for such curved intersection searching problems were known only for the case where S consists of curved objects and q is linear.

Keywords: Computational geometry, data structures, intersection searching, geometric transforms, partition trees, cutting trees, spanning paths of low stabbing number, simplex range searching, simplex composition, persistent data structures.

---

*Department of Computer Science, University of Minnesota, Minneapolis, MN 55455, U.S.A. Email: {pgupta, janardan}@cs.umn.edu. The research of these authors was supported in part by NSF grant CCR–92–00270.

†Max-Planck-Institut für Informatik, W-6600 Saarbrücken, Germany. Email: michiel@mpi-sb.mpg.de. This author was supported by the ESPRIT Basic Research Actions Program, under contract No. 7141 (project ALCOM II).

1 Introduction

Intersection searching problems are among the most fundamental and widely-studied classes of problems in computational geometry. In a generic instance of such a problem, a set, S, of n geometric objects must be preprocessed into a suitable data structure, so that for any query object, q, the objects of S that are intersected by q can be *reported* or *counted* efficiently. Specific examples of the objects in S and of q are intervals on the real line, points, lines, line segments, rays, and triangles in the plane, and, more generally, points, hyperplanes, and simplices in higher dimensions. (Of course, q need not be of the same type as the objects in S.) Intersection searching problems arise in diverse application areas, such as, for instance, robot motion planning, computer graphics, and computer-aided design, and so efficient solutions (measured in terms of the space used by the data structure and the query time) are of great interest. Such space- and query-time-efficient solutions have been devised for many of these problems. See, for instance, [Cha86, CW89, DE87, Aga89, Mat92, CJ92] for a sampling of such results.

With a few exceptions, all previous work on these problems assumes that the input objects and the query object are linear (e.g., lines or line segments) or piecewise-linear (e.g., polygonal or polyhedral). The case where the input and/or the query are non-linear (e.g., circles, disks, circular arcs, Jordan curves) has been largely ignored. Yet, such problems arise often in practice, such as, for instance, in planning a collision-free path for a disk-shaped robot in an environment consisting of polygonal obstacles. The only previous work on intersection searching in this setting that we are aware of is [AvKO, vKOA90, Sha91]. In [AvKO], efficient solutions are given for several problems where the input objects are non-linear (e.g., circles, disks, circular arcs, Jordan arcs) and the query object is linear (e.g., lines, line segments, halfspaces, rays). In [vKOA90], intersection searching with lines or line segments in sets of disks is considered. In [Sha91], the problem of stabbing a set of disks in the plane with a query point is considered.

In this paper, we make further contributions to such curved intersection searching problems by presenting efficient solutions to three broad classes of problems. (In the context of this paper, the term "curved" means circular or circle-like objects such as circles, disks, circular arcs, annuli, d-balls, and d-spheres.)

The first thrust of this paper is the design of efficient solutions to the following general problem: Preprocess a set S of n linear objects (e.g., points, lines, or line segments), so that the k objects that are intersected by a curved query object (e.g., a circle, disk, or circular arc) can be counted or reported efficiently. Thus, this part of our work complements the results

Input objects	*Query object*	*Query mode*	*Space*	*Query time*
Points	Variable-radii annulus	Count/Report	$n^{3+\epsilon}$ $n \log n$	$\log n$ $(+k)$ $\sqrt{n} \log^2 n$ $(+k)$
Lines	Variable-radius circle/disk	Count/Report	$n^{3+\epsilon}$ n	$\log n$ $(+k)$ $n^{2/3+\epsilon}$ $(+k)$
		Report	$n \log^2 n$	$n^{2/3} \log^2 n + k$
	Fixed-radius circle/disk	Count/Report	n	$n^{1/2+\epsilon}$ $(+k)$
		Report	$n \log n$	$\min\{\sqrt{n} \log^2 n + k,$ $\sqrt{n} \log n + k \log n\}$
	Variable-radius circular arc	Count/Report	$n^{3+\epsilon}$ n	$\log n$ $(+k)$ $n^{2/3+\epsilon}$ $(+k)$
Horizontal line segments	Variable-radius circle	Report	$n \log^5 n$	$\sqrt{n} \log^5 n + k$
Arbitrary line segments	Variable-radius disk	Report	$n^{3+\epsilon}$ n	$\log n + k$ $n^{2/3+\epsilon} + k$
	Fixed-radius disk	Report	$n \log^2 n$	$\min\{\sqrt{n} \log^2 n + k,$ $\sqrt{n} \log n + k \log n\}$
	Variable-radius circle	Report	n $n^{3+\epsilon}$	$n^{2/3+\epsilon} + k$ $\log n + k$

Table 1: Summary of main results for intersection searching on linear objects with curved query objects. k denotes the output size for the reporting problem. All bounds are big-oh and worst-case. Additional space-query time tradeoffs are possible but not shown.

in [AvKO]. Table 1 summarizes our results.[1] Our goal has been to obtain useful space and query time trade-offs and, as can be seen from the table, we obtain polylogarithmic query time at one extreme and linear or almost-linear space at the other extreme. We remark that other intermediate trade-offs are also easily derived from our results; however, in order to keep the paper to reasonable length, we do not discuss these here.

The second thrust of the paper is efficient solutions for the problem of preprocessing a set S of n curved objects (e.g., d-balls, d-spheres[2], circles, circular arcs) so that the ones that are intersected by a curved query object can be counted or reported efficiently. This problem was left open in [AvKO]. Table 2 summarizes our results for this class of problems.

[1]Throughout the paper, a query time written in the form $O(t(n)\ (+k))$ should be interpreted as $O(t(n))$ for the counting problem and $O(t(n) + k)$ for the reporting problem. We use this convention for brevity. Also, throughout $\epsilon > 0$ is an arbitrarily small constant. Whenever ϵ appears in a query time (resp. space) bound, the corresponding space (resp. query time) bound contains a multiplicative factor which goes to ∞ as $\epsilon \to 0$.

[2]A *d-sphere* is the boundary of a closed d-ball

Input objects	*Query object*	*Query mode*	*Space*	*Query time*
d-ball	d-ball	Count/Report	n	$n^{1-1/(d+2)}(\log\log n)^{O(1)}$ $(+k)$
			$n^{d+2+\epsilon}$	$\log n$ $(+k)$
		Report	$n \log\log n$	$n^{1-1/\lfloor (d+2)/2 \rfloor}(\log n)^{O(1)} + k$
d-sphere	d-sphere	Count/Report	n	$n^{1-1/(d+2)}(\log\log n)^{O(1)}$ $(+k)$
			$n^{d+2+\epsilon}$	$\log n$ $(+k)$
Circular arcs	Circle	Report	n	$n^{3/4+\epsilon} + k$
			$n^{3+\epsilon}$	$\log n + k$
Circles	Circular arc	Report	n	$n^{3/4+\epsilon} + k$
			$n^{3+\epsilon}$	$\log n + k$

Table 2: Summary of results for intersection searching on curved objects using a curved query object. The input objects are of arbitrary radii and the radius of the query object is variable. All bounds are big-oh and worst-case. Additional space-query time tradeoffs are possible but not shown.

Here again, we only show the tradeoffs at the two extremes.

Finally, we also consider a generalized version of these intersection searching problems. In this setting, the set S consists of n linear or curved objects that are colored and the goal is to count or report the distinct colors of the objects that are intersected by a query object. Such generalized problems have been considered recently in [JL93, GJS93, AvK93] in the context of intersection searching involving linear input and query objects and have been shown to be rich in applications. The goal in these problems is to obtain efficient solutions whose query times are sensitive to the output size, namely the number, i, of intersected colors (not the number, k, of intersected objects, which can be much larger). Typically, we seek query times of the form $O(f(n)\ (+i))$, where $f(n)$ is polylogarithmic. Table 3 summarizes our results for these problems. Note that these problems contain, as a special case, the standard intersection searching problems where we are interested in all the objects intersected by the query—the standard problem is obtained by assigning each input object a distinct color—hence the term "generalized". Thus, for instance, the fixed-radius bounds in Table 3 also apply to the corresponding standard problems. (To avoid repetition, we have omitted these results from the earlier tables.)

Our results are based on two main approaches. The first approach is to convert the curved problem at hand to an instance of an appropriate simplex range searching problem by the application of one or more geometric transforms. We then solve the latter problem by suitably composing together well-known techniques such as partition trees, cutting trees, and spanning paths of low stabbing number. This general approach has been used in [AvKO].

<table>
<tr><th>Input objects</th><th>Query object</th><th>Query mode</th><th>Space</th><th>Query time</th></tr>
<tr><td>Points</td><td>Fixed-radius annulus</td><td rowspan="4">Count/Report</td><td rowspan="4">n^2</td><td rowspan="4">$\log n$ $(+i)$</td></tr>
<tr><td>Lines</td><td rowspan="2">Fixed-radius circle/disk</td></tr>
<tr><td>Arbitrary line segments</td></tr>
<tr><td>Disks of arbitrary radii</td><td>Fixed-radius disk</td></tr>
<tr><td rowspan="2">Points</td><td>Variable-radii annulus</td><td>Count
Report</td><td>$n^4 \log^2 n$
$n^4 \log n$</td><td>$\log^2 n$
$\log n + i$</td></tr>
<tr><td>Variable-radius disk</td><td rowspan="4">Count

Report</td><td rowspan="4">$n^4 \log n$

n^4</td><td rowspan="4">$\log n$ $(+i)$</td></tr>
<tr><td>Lines</td><td>Variable-radius circle/disk</td></tr>
<tr><td>Arbitrary line segments</td><td rowspan="2">Variable-radius disk</td></tr>
<tr><td>Disks of arbitrary radii</td></tr>
</table>

Table 3: Summary of results for generalized intersection searching with curved objects. All bounds are big-oh and worst-case. Here i is the output size for the generalized problem, i.e., the number of distinct colors intersected by the query.

What makes this part of our work interesting is that the characterization of intersection and the appropriate transform(s) to use is not always apparent; indeed, in some cases, we need to apply successively more than one transform in order to obtain a problem of the desired form. We use several well-known transforms and also introduce some new ones.

Our second approach, which we use for the generalized problems, is based on a combination of the locus approach and persistent data structures: We partition the plane into suitable regions within which the answer to a query is invariant and store with each region the associated answer. However, storing this information in its entirety entails a large storage overhead. We show how to reduce the storage by roughly an order of magnitude, without affecting the query time, by ordering the regions suitably and applying persistence techniques. Furthermore, for the variable-radius problems, the querying strategy is somewhat subtle because the answer to the query is not available explicitly in the information stored with each region but rather is embedded in a region-specific total order on the input objects. We show how the search within this total order can be reduced to a generalized 1-dimensional range searching problem, which is solvable efficiently.

Thus the contribution of the paper is a uniform framework to solve efficiently a wide variety of intersection searching problems involving curved objects. We believe that our methods will be useful in solving intersection searching problems involving other kinds of curved objects as well.

The rest of the paper is organized as follows: Section 2 reviews several techniques that we use frequently in the paper. Sections 3-5 discuss intersection searching on linear objects with curved query objects. The problem of querying curved objects with curved objects is considered in Section 6. Section 7 discusses the generalized problems. We conclude in Section 8 with a discussion of some open problems.

In general, in this paper we will ignore the issue of degenerate configurations, such as, for instance, a query object merely touching an input object. Our algorithms can be modified easily to handle degenerate cases without any loss of asymptotic efficiency (as we illustrate for the algorithm in Section 5.1), but at the expense of complicating the exposition.

2 Preliminaries

In this section, we review certain geometric transformations and data structuring techniques that we use frequently in the paper. Our review is brief and we refer the reader to the references provided for further details.

2.1 Geometric transformations

We consider the following three groups of transformations. Further details can be found in [Ede87, AvKO].

1. Let C be a circle in the plane, with center (a, b) and radius r and let $p = (c, d)$ be a point in $\mathcal{R}^2$. Define $\varphi(C)$ to be the plane $z = a(2x - a) + b(2y - b) + r^2$ in $\mathcal{R}^3$ and define $\psi(p)$ to be the point $(c, d, c^2 + d^2)$ in $\mathcal{R}^3$. $\psi(p)$ is the vertical projection of p onto the paraboloid $U : z = x^2 + y^2$. $\varphi(C)$ is the unique plane in $\mathcal{R}^3$ which intersects the paraboloid U in the vertical projection of C onto U. Note that $\psi(p)$ lies below (resp. on, above) $\varphi(C)$ iff p lies inside (resp. on, outside) C.

2. Let C be as in (1) above and let ℓ be the non-vertical line $y = mx + c$. Let ℓ^+ (resp. ℓ^-) be the closed halfplane lying above (resp. below) ℓ.[3] Define $\mu(C)$ to be the point (a, b, r) in $\mathcal{R}^3$ and $\omega(\ell^+)$ (resp. $\omega(\ell^-)$) to be the halfspace $z \geq (mx - y + c)/\sqrt{1 + m^2}$ (resp. $z \geq (-mx + y - c)/\sqrt{1 + m^2}$) in $\mathcal{R}^3$. If ℓ is the vertical line $x = c$, then define $\omega(\ell^+)$ (resp. $\omega(\ell^-)$) to be the halfspace $z \geq x - c$ (resp. $z \geq -x + c$). It can be verified easily that ℓ^+ (resp. ℓ^-) intersects C iff $\mu(C) \in \omega(\ell^+)$ (resp. $\mu(C) \in \omega(\ell^-)$). Moreover, ℓ intersects C iff $\mu(C) \in \omega(\ell^+) \cap \omega(\ell^-)$.

3. The following transformation, $\mathcal{F}$, is the well-known point-hyperplane duality. Consider the point $p = (a, b)$ and the non-vertical line $\ell : y = sx - t$ in $\mathcal{R}^2$. Define $\mathcal{F}(p)$ to be the line $y = ax - b$ and $\mathcal{F}(\ell)$ to be the point (s, t). Likewise, given the point $p = (a, b, c)$ and the non-vertical plane $\ell : z = sx + ty - u$ in $\mathcal{R}^3$, let $\mathcal{F}(p)$ be the plane $z = ax + by - c$ and $\mathcal{F}(\ell)$ the point (s, t, u). It is easy to see that parallel lines (resp. planes) are mapped to points with the same y-coordinate (resp. x- and y-coordinate) and, moreover, p lies above (resp. on, below) ℓ iff $\mathcal{F}(p)$ lies below (resp. on, above) $\mathcal{F}(\ell)$.

In addition to the above transforms, we also introduce some new transforms in Section 6 for intersection searching on curved objects with curved query objects.

2.2 Data structures for simplex range searching

In the *simplex range searching* problem, a set S of n points in $\mathcal{R}^d$ is to be preprocessed so that the points inside a query simplex can be reported or counted efficiently. An important special case is *halfspace range searching*, where the query is a halfspace. Our interest in this

[3] In general, if h is a non-vertical hyperplane in $\mathcal{R}^d$, $d \geq 2$, then we use h^+ (resp. h^-) to denote the closed halfspace lying above (resp. below) h.

problem stems from the fact that many of our curved intersection searching problems can be reduced, usually via a geometric transformation, to simplex range searching.

Recently, several techniques, exhibiting different space-query time tradeoffs, have been developed for this problem. These include spanning paths of low stabbing number [CW89], partition trees [Mat92], and cutting trees [Mat91a]. We review these and a technique called simplex composition [vK92] in this subsection. (A very good exposition on most of these topics can be found in [vK92, dB92].)

2.2.1 Spanning paths of low stabbing number

Let S be any set of n points in $\mathcal{R}^d$ and let $A \subseteq S$. Following Chazelle and Welzl [CW89], we say that a query range q *stabs* A if there exist $x, y \in A$ such that $x \in q$ and $y \notin q$. In particular, if $|A| = 2$, say $A = \{x, y\}$, then q stabs the line segment (edge) $\overline{xy}$ if q stabs A. Chazelle and Welzl have shown that S admits a polygonal path Π, which spans the points of S, such that any query halfspace in $\mathcal{R}^d$ stabs at most $c(n) = O(n^{1-1/d})$ edges of Π. The number $c(n)$ is called the *stabbing number of* Π.

Given Π, we can construct a binary tree, T, of height $O(\log n)$ which stores the points of S at its leaves, in the order in which they appear on Π. This tree is called an *s-tree* [AvKO]. Let $T(v)$ be the subtree of T rooted at a node v, let $S(v)$ be the points of S stored at the leaves of $T(v)$, and let $\Pi(v)$ be the subpath of Π spanning $S(v)$. For any query halfspace, H, in $\mathcal{R}^d$ the *canonical nodes of* T *w.r.t.* H are the highest nodes v of T such that $\Pi(v)$ is not stabbed by H. Let V_H be the set of canonical nodes w.r.t. H. As shown in [CW89], $|V_H| = O(n^{1-1/d} \log n)$. The sets $S(v), v \in V_H$, partition S and each $S(v)$ either lies completely in H or is disjoint from it. Thus, once V_H has been identified, the points lying in H can be counted by simply adding up $|S(v)|$ for each $v \in V_H$ such that $S(v)$ lies completely in H. ($|S(v)|$ is computed during preprocessing.) Thus, the query time is upper-bounded by the time to identify V_H. Similarly, for the reporting case. (Here we also need to include k—the number of points reported—in the query time.)

As noted in [CW89], all of the above discussion also applies to the case where the query range is a d-dimensional simplex or a d-ball (d fixed), rather than a halfspace.

How (and how fast) can V_H be computed? In [CW89], it has been shown that if $d = 2, 3$, then for any halfspace H, V_H can be computed in time $O(n^{1-1/d} \log^{d-1} n)$, as follows: Do a depth-first search of T but not searching below a node v iff H does not stab $S(v)$, i.e., iff the bounding plane of H does not intersect the convex hull $\mathrm{CH}(v)$ of $S(v)$. Each such v is included in V_H. Since the intersection test takes $O(\log n)$ time [DK83], the total time is $O(n^{1-1/d} \log^2 n)$. However, for $d = 2$, a logarithmic factor can be trimmed via fractional

cascading [CG86]. This establishes the claimed bound. The space used by T is $O(n \log n)$. Clearly, this bound also extends to a query simplex.

As shown in [CW89], if $d = 2$ then for any query disk D (a 2-ball), V_H can be found in $O(\sqrt{n} \log^2 n)$ time. The idea is to do a depth-first search of T but not search below a node v iff D does not stab $S(v)$. (Each such v is included in V_H.) Let C be the bounding circle of D. By using the transformations φ and ψ of Section 2.1 the above stabbing test becomes equivalent to testing whether, in $\mathcal{R}^3$, the plane $\varphi(C)$ intersects the convex hull of the point-set $\{\psi(p) \mid p \in S(v)\}$. Since this takes $O(\log n)$ time [DK83], the claimed bound follows. Again, T uses $O(n \log n)$ space.

2.2.2 Simplicial partitions and partition trees

Let S be a set of n points in $\mathcal{R}^d$. In [Mat92], Matoušek has shown that for any parameter $r \leq n$, $\mathcal{R}^d$ can be decomposed into $m = O(r)$ simplices $s_1, s_2, \ldots, s_m$ such that (i) each s_i contains a subset S_i of S, (ii) the sets S_i together partition S, (iii) $|S_i| \leq 2n/m$, and (iv) any hyperplane intersects only $O(r^{1-1/d})$ simplices. (Note that simplices may overlap and a simplex s_i can contain other points of S besides S_i.) Such a decomposition is called a *fine simplicial partition for* S.

This decomposition is the basis for the construction of an efficient *partition tree*, T, for simplex range searching on S. T is defined recursively as follows: We associate the pair $(\mathcal{R}^d, S')$ with the root v of T, where $S' = S$. If S' is greater than some prescribed constant, then we compute a fine simplicial partition for S', with $r = |S'|^{1-1/d}$, and, for $1 \leq i \leq m$, we associate the pair (s_i, S_i) with a child of v. We then construct recursively the subtree rooted at each child of v.

To answer a counting or reporting query for a simplex q, we search in T starting at v. Let u be the current node in the search, with associated pair (s_i, S_i). If u is a leaf, then we check the points of S_i against q one by one. Suppose that u is not a leaf. If q contains S_i, then we add $|S_i|$ (which is computed during preprocessing) to a global count or report the points in S_i. If q is disjoint from S_i, we stop searching below u. If q intersects S_i, then we recursively search all of u's children.

Theorem 2.1 [Mat92] *A set S of n points in $\mathcal{R}^d$ can be preprocessed, in time $O(n \log n)$, into a data structure of size $O(n)$ such that a simplex counting (resp. reporting) query can be answered in time $O(n^{1-1/d}(\log \log n)^{O(1)} (+k))$, where k is the number of points reported.* □

2.2.3 Cuttings and cutting trees

Whereas simplicial partitions are defined on a set of points, cuttings are defined on a set of hyperplanes and hence, in a sense, operate in the dual space.

Let H be a set of n hyperplanes in $\mathcal{R}^d$. In [CF91], Chazelle and Friedman have shown that for any parameter $r \leq n$, $\mathcal{R}^d$ can be partitioned into $m = O(r^d)$ simplices, $s_1, s_2, \ldots, s_m$, such that any simplex intersects at most n/r hyperplanes of H. Such a partition is called a $\frac{1}{r}$*–cutting for* H.

A *cutting tree*, T, for H is defined as follows: With the root v of T we associate the 4–tuple (s, H', H^a, H^b), where $s = \mathcal{R}^d$, $H' = H$, and $H^a = \emptyset$ (resp. $H^b = \emptyset$) is the subset of hyperplanes of H lying above (resp. below) s. If $|H'|$ is larger than some prescribed constant then we compute a $\frac{1}{r}$–cutting for H', for some $r \leq n$. For $1 \leq i \leq m$, let s_i be a simplex in the cutting and let $H_i \subseteq H'$ consist of the hyperplanes that intersect s_i. Let H_i^a (resp. H_i^b) consist of the hyperplanes of H' that are above (resp. below) s_i. We associate (s_i, H_i, H_i^a, H_i^b) with a child of v and then recursively construct its subtree.

Let S be a set of n points in $\mathcal{R}^d$. Let us consider how to use the cutting tree T to answer a halfspace counting or reporting query using a query halfspace h^+. By duality, we are equivalently searching in a set, H, of hyperplanes for those hyperplanes that lie below the query point $q = \mathcal{F}(h)$. We search from the root of T, always maintaining the invariant that q is in the simplex associated with the current node u. Let (s_i, H_i, H_i^a, H_i^b) be the 4–tuple associated with u. We first add $|H_i^b|$ to a global count (or report its hyperplanes). If u is a leaf then we also check the hyperplanes in H_i against q one by one. Otherwise, we determine the child of u whose associated simplex contains q and search its subtree recursively.

Theorem 2.2 [Cha91] *A set S of n points in $\mathcal{R}^d$ can be preprocessed, in time $O(n^d)$, into a data structure of size $O(n^d)$ such that the points that lie in a query halfspace can be counted (resp. reported) in time $O(\log n\ (+k))$, where k is the number of reported points.* □

Now suppose that the query is a simplex q. Then q dualizes to a set q' of $d+1$ points in $\mathcal{R}^d$, one per bounding hyperplane of q. Thus, we are searching for the hyperplanes of H that satisfy the appropriate above/below relationship w.r.t. each point in q'. We can solve this problem as above, except that at each node u of T we recursively append an auxiliary structure of the same kind for H_i^a and for H_i^b. (Thus, there are d levels of auxiliary structures.) To answer a query, we search in the outermost tree with one of the points of q' and identify a set of $O(\log n)$ nodes whose associated simplices contain the point. We then search the auxiliary structures of these nodes recursively with the remaining points in

q'. This approach yields a total query time of $O(\log^{d+1} n)$. By using additional techniques (see [Cha91, CSW90, Mat91a] or [dB92]) this can be reduced to $O(\frac{1}{\epsilon} \log n)$ while the storage becomes $O(n^{d+\epsilon})$, where $\epsilon > 0$ is an arbitrarily small constant.

Theorem 2.3 [Cha91, CSW90, Mat91a] *A set S of n points in $\mathcal{R}^d$ can be preprocessed, in time $O(n^{d+\epsilon})$, into a data structure of size $O(n^{d+\epsilon})$ such that the points that lie in a query simplex can be counted (resp. reported) in time $O(\log n\ (+k))$, where k is the number of reported points and $\epsilon > 0$ is an arbitrarily small constant.* □

2.2.4 Simplex composition

Let $\mathcal{S}$ be a set of n geometric objects in $\mathcal{R}^d$ and let D be a data structure for some query problem on $\mathcal{S}$. Let the space and query time bounds for D be $O(f(n))$ and $O(g(n))$, respectively. Suppose that we wish to now solve our query problem not w.r.t. $\mathcal{S}$ but w.r.t. a subset $\mathcal{S}'$ of $\mathcal{S}$ that satisfies some condition. Moreover, suppose that $\mathcal{S}'$ can be specified by putting $\mathcal{S}$ in 1–1-correspondence with a set $\mathcal{P}$ of points in $\mathcal{R}^d$ and letting $\mathcal{S}'$ correspond to the subset $\mathcal{P}'$ of $\mathcal{P}$ that is contained in a query simplex.

How can we solve the query problem on $\mathcal{S}'$ and what is its complexity as a function of n, d, $f(n)$, and $g(n)$? In [vK92], van Kreveld investigates this general problem, which he calls *a simplex composition on $\mathcal{P}$ to D*, and proves the following result:

Theorem 2.4 [vK92] *Let $\mathcal{P}$ be a set of n points in $\mathcal{R}^d$ in 1–1-correspondence with a set $\mathcal{S}$ of n objects in $\mathcal{R}^d$. Let D be a data structure on S of size $O(f(n))$ and with query time $O(g(n))$. For an arbitrarily small constant $\epsilon > 0$, the application of simplex composition on $\mathcal{P}$ to D results in a data structure*

(i) *of size $O(n^{\epsilon}(n^d + f(n)))$ and query time $O(\log n + g(n))$,*

(ii) *of size $O(n + f(n))$ and query time $O(n^{\epsilon}(n^{1-1/d} + g(n)))$,*

(iii) *of size $O(m^{\epsilon}(m + f(n)))$ and query time $O(n^{\epsilon}(g(n) + n/m^{1/d}))$, for any $n \leq m \leq n^d$,*

assuming that $f(n)/n$ is nondecreasing and $g(n)/n$ is nonincreasing. □

We note that for a reporting problem, $g(n)$ represents just the time to search in D for the answer and does not include the time to actually output the answer. For the reporting problem, if the query time of D is of the form "$O(g(n))$ plus the output size", then for the

composed problem we must include the overall output size as a linear additive term in the query times given in Theorem 2.4(i)–(iii).

Very briefly, part (i) of the theorem is obtained by building a cutting tree T on the set of hyperplanes that are dual to the points in $\mathcal{P}$ and storing at each node u an instance of D for the subset of $\mathcal{S}$ that is in correspondence with the hyperplanes that miss u's simplex in the cutting. Given the query simplex, a subset of the nodes of T is identified (as in Section 2.2.3) and the D-structure at these nodes is queried. Part (ii) is obtained by building a partition tree on $\mathcal{P}$ and storing at each node u an instance of D for the subset of $\mathcal{S}$ that is in correspondence with the points of $\mathcal{P}$ associated with u's simplex in the simplicial partition. Part (iii) is obtained by combining parts (i) and (ii) suitably.

We will use Theorem 2.4 extensively in this paper. For brevity, throughout we will apply only parts (i) and (ii) to illustrate the two extremes in the space-query time trade-off that are attainable. The reader should be aware that part (iii) can also be applied to get intermediate trade-offs.

Let us illustrate the above ideas with the following (somewhat contrived) example: Let $\mathcal{S}$ be a set of n vertical line segments in $\mathcal{R}^2$ and let D be a data structure for the following query problem on $\mathcal{S}$: "Given a query line ℓ, find the segment of $\mathcal{S}$ whose midpoint is closest to ℓ." Now suppose that we wish to restrict our search to only those segments of $\mathcal{S}$ that are intersected by ℓ. Since a segment $s \in \mathcal{S}$ intersects ℓ iff its upper endpoint is in ℓ^+ and its lower endpoint is in ℓ^-, we can cast the intersection condition as two halfplane (i.e., simplex) compositions: In the first, we associate $\mathcal{S}$ with the set $\mathcal{U}$ of upper endpoints and use ℓ^+; in the second, we associate $\mathcal{S}$ with the set $\mathcal{L}$ of lower endpoints and use ℓ^-. We then apply these two compositions successively using Theorem 2.4.

As another example (and the kind that we encounter most frequently in this paper), suppose that we wish to count or report the segments of $\mathcal{S}$ that are intersected by ℓ. We take D to be a linked list of the objects of $\mathcal{S}$ and store its size at its head. Clearly, D solves the trivial "query" problem, "count or report the segments of $\mathcal{S}$," in $f(n) = O(n)$ space and $g(n) = O(1)$ query time. Thus, as in the previous example, the restricted problem (namely, "count or report the segments in $\mathcal{S}$ that are also intersected by ℓ") can be solved via two successive halfplane compositions. The query time is $O(\log n\ (+k))$ (resp. $O(n^{1/2+\epsilon}\ (+k))$) and the space is $O(n^{2+\epsilon})$ (resp. $O(n)$).

3 Variable-radii annulus range searching

As a warm-up problem, we consider the following problem whose solution is fairly simple but illustrates some of the ideas: Let S be a set of n points in the plane. Let $Ann(q, r_1, r_2)$ be a query *annulus*, i.e., the closed region of the plane bounded by the circles C_1 and C_2 of radius r_1 and r_2 ($r_1 \leq r_2$), respectively, centered at q. We assume that r_1 and r_2 are part of the query and not known beforehand. The problem is to preprocess S so that the points that lie in $Ann(q, r_1, r_2)$ can be counted or reported efficiently. Our approach is based on geometric duality and simplex composition.

Let C_1 and C_2 be the inner and outer circle, respectively, of the query annulus. From the discussion of the transforms φ and ψ in Section 2.1 it follows that our problem is equivalent to counting or reporting those points in the set $S' = \{\psi(p) \mid p \in S\}$ in $\mathcal{R}^3$ that lie between the planes $\varphi(C_1)$ and $\varphi(C_2)$, i.e., in $\varphi(C_1)^+ \cap \varphi(C_2)^-$. We solve this by two applications of halfspace composition in $\mathcal{R}^3$ (Theorem 2.4).

With reference to Theorem 2.4, let $\mathcal{S} = \mathcal{P} = S'$ and let the structure D be a linked list of points, with its size stored at the head. Clearly, $f(n) = O(n)$ and $g(n) = O(1)$. Applying Theorem 2.4(i) on D (with $d = 3$) gives a structure D' of size $f'(n) = O(n^\epsilon(n^3+n)) = O(n^{3+\epsilon})$ and query time $g'(n) = O(\log n)$ to count or report the points lying in $\varphi(C_1)^+$. (For the reporting problem, the overall time is $g'(n)$ plus the output size.) Applying Theorem 2.4(i) again, with $D = D'$ and $\mathcal{S} = \mathcal{P} = S'$, gives a structure of size $O(n^\epsilon(n^3 + n^{3+\epsilon})) = O(n^{3+\epsilon'})$ and query time $O(\log n + \log n\ (+k)) = O(\log n\ (+k))$ to count or report the points lying in $\varphi(C_1)^+ \cap \varphi(C_2)^-$.

Theorem 3.1 *A set S of n points in the plane can be preprocessed into a data structure of size $O(n^{3+\epsilon})$, where $\epsilon > 0$ is an arbitrarily small constant, such that the points lying inside a variable-radius query annulus can be counted (resp. reported) in time $O(\log n\ (+k))$, where k is the number of reported points.* □

Note that Theorem 2.4(ii) can be similarly applied to get a structure of size $O(n)$ and query time $O(n^{2/3+\epsilon})$. However, we can use a different approach to get the much lower query time of $O(\sqrt{n}\log^2 n\ (+k))$, while the space increases only slightly to $O(n \log n)$. This approach is similar to the one given in [CW89] for disk range searching.

As before, our goal is to report or count those points in the set $\{\psi(p) \mid p \in S\}$ that lie in $\varphi(C_1)^+ \cap \varphi(C_2)^-$. In preprocessing, we construct a spanning path Π of stabbing number $c(n) = O(\sqrt{n})$ (w.r.t. disks) on the points of S and build an s-tree on it (cf. the

discussion in Section 2.2.1). At each node v, we store the convex hull $\mathrm{CH}(v)$ of the point-set $\{\psi(p) \mid p \in S(v)\}$.

To answer a query, we compute the set V_1 of canonical nodes of T w.r.t. the disk bounded by C_1, as described in Section 2.2.1. Let $V_1' \subseteq V_1$ consist of those nodes v such that $\mathrm{CH}(v)$ is in $\varphi(C_1)^+$. We next compute (a subset of) the canonical nodes of T w.r.t. the disk bounded by C_2 by searching only in the subtrees rooted at the nodes belonging to V_1'. Let $V_2' \subseteq V_2$ consist of those nodes v such that $\mathrm{CH}(v)$ lies in $\varphi(C_2)^-$, and, therefore, in $\varphi(C_1)^+ \cap \varphi(C_2)^-$. For each $v \in V_2'$, we simply add up the $|S(v)|$'s for a counting query and report the points in $S(v)$ for a reporting query. The query time and space follow from the discussion in Section 2.2.1 and we thus conclude:

Theorem 3.2 *A set S of n points in the plane can be preprocessed into a data structure of size $O(n \log n)$ such that the points lying inside a variable-radius query annulus can be counted (resp. reported) in time $O(\sqrt{n} \log^2 n\ (+k))$, where k is the output size.* □

4 Intersection searching on lines

In this section, we consider how to preprocess a set S of lines in the plane so that we can count or report efficiently the intersections of a query circle or circular arc with the lines. We note that all our results in this section also hold for query disks since a line in S intersects a circle iff it intersects the corresponding disk.

4.1 Querying with a variable-radius circle

We successively apply two of the transformations from Section 2.1 to convert the problem to one in $\mathcal{R}^3$, concerning line segments and a query plane.

Let $q = (a, b)$ be the center and r the radius of the query circle $C = C(q, r)$. Let $\ell : y = mx + c$ be any line in the input set S. From Section 2.1, we know that C intersects ℓ iff the point $p = \mu(C) = (a, b, r)$ is contained in the wedge $\omega(\ell^+) \cap \omega(\ell^-)$. By definition of ω, there exist planes H_1 and H_2, such that $\omega(\ell^+) = H_1^+$ and $\omega(\ell^-) = H_2^+$. Thus, C intersects ℓ iff $p \in H_1^+ \cap H_2^+$.

Let us now apply the transform $\mathcal{F}$ to map p to a non-vertical plane and H_1 and H_2 to points in $\mathcal{R}^3$. Since p is above H_1 iff $\mathcal{F}(p)$ is below $\mathcal{F}(H_1)$ (and similarly for H_2), it follows that C intersects ℓ iff $\mathcal{F}(p)$ is below the line segment $\overline{\mathcal{F}(H_1)\mathcal{F}(H_2)}$. Hence, we have reduced

the problem of counting or reporting the lines intersected by a variable-radius query circle to the following problem:

Problem 4.1 *Preprocess a set S' of n line segments in $\mathcal{R}^3$ so that for any non-vertical query plane H, the segments that lie completely in H^+ can be counted or reported efficiently.*

We can solve this problem using halfspace composition, as follows: With reference to Theorem 2.4, let $\mathcal{S} = S'$ and let $\mathcal{P}$ be the set of left endpoints (say) of the segments in S'. Let D be a linked list of the segments, with its size stored at its head. As before, $f(n) = O(n)$ and $g(n) = O(1)$. Applying Theorem 2.4(i) to D, with $d = 3$, we get a structure D' of size $f'(n) = O(n^{3+\epsilon})$ and query time $g'(n) = O(\log n)$. Applying Theorem 2.4(i) again, with $D = D'$ and $\mathcal{S}$ and $\mathcal{P}$ being the set of right endpoints, gives the desired data structure, which uses $O(n^{3+\epsilon'})$ space and has a query time of $O(\log n\ (+k))$, where $\epsilon' > 0$ is an arbitrarily small constant.

Alternatively, we can apply Theorem 2.4(ii). This yields a structure with space $O(n)$ and query time $O(n^{2/3+\epsilon'}\ (+k))$.

Finally, we can also obtain a faster query time for the reporting problem, while increasing the space slightly. We build a spanning path of stabbing number $O(n^{2/3})$ (w.r.t. halfspaces) on the endpoints of the segments in S' and then build an s-tree, T, on it. At each node v, we store an instance, $I(v)$ of the halfspace reporting structure of Aggarwal et al. [AHL90]. (This is in addition to the convex hull $\mathrm{CH}(v)$ of $S'(v)$—cf. Section 2.2.1.) $I(v)$ is built on a set, $S''(v)$, of points, defined as follows: for each point $p \in S'(v)$, we include in $S''(v)$ the other endpoint of the segment to which p belongs. $I(v)$ uses $O(m \log m)$ space and has a query time of $O(\log m + k)$, where $m = |S''(v)|$.

Given a query halfspace H^+, we identify the set, V_{H^+}, of canonical nodes and then determine a subset V'_{H^+} of V_{H^+} consisting of the nodes v of T such that $S'(v) \in H^+$. At each node $v \in V'_{H^+}$, we query $I(v)$ with H^+. The correctness of the method is clear. From the discussion in Section 2.2.1, V'_{H^+} has size $O(n^{2/3} \log n)$ and can be found in $O(n^{2/3} \log^2 n)$ time. The halfspace queries take $O(n^{2/3} \log^2 n + k)$ total time. The space used per level of T is $O(n \log n)$ and so the overall storage is $O(n \log^2 n)$.

Thus, we have shown:

Theorem 4.1 *A set S of n lines in the plane can be preprocessed into a data structure of size $O(n^{3+\epsilon})$ (resp. $O(n)$) such the lines that are intersected by any variable-radius query circle, can be counted or reported in time $O(\log n\ (+k))$ (resp. $O(n^{2/3+\epsilon}\ (+k))$), where $\epsilon > 0$ is an arbitrarily small constant and k is the number of reported lines. Moreover, for the*

reporting problem, there is also a solution which uses $O(n \log^2 n)$ space and has a query time of $O(n^{2/3} \log^2 n + k)$. □

4.1.1 The case of a fixed-radius query circle

If the radius, r, of C is fixed, then there is a simpler and more efficient solution, which is based on the following easily-proved lemma.

Lemma 4.1 *A circle C of radius r intersects a line ℓ iff the center of C lies within the closed strip,* strip_ℓ*, whose bounding lines are parallel to ℓ and at a distance of r on each side of ℓ.* □

Using the transform $\mathcal{F}$, we dualize the bounding lines of each non-vertical strip to two points that have the same x-coordinate; thus a strip dualizes to the vertical line segment joining the two points. (As we will see, vertical strips can be handled easily.) The query point q dualizes to a line. By the incidence-preserving property of $\mathcal{F}$, the point enclosure problem for the strips is now equivalent to: preprocess a set S' of n vertical line segments in $\mathcal{R}^2$ so that the segments that are intersected by a query line ℓ can be counted or reported efficiently. But this is precisely the example discussed in Section 2.2.4. It was shown there that the problem is solvable in $O(n)$ space and $O(n^{1/2+\epsilon}\ (+k))$ query time. (There is also an $O(n^{2+\epsilon})$-space and $O(\log n\ (+k))$-query time solution. However, this is inferior to the bounds that can be obtained from Section 7.3 for the more general problem of querying colored lines with a fixed-radius query circle.)

For the reporting problem, yet another tradeoff is possible. The segments in S' that are intersected by ℓ can be reported in $O(n \log n)$ space and $O(\min\{\sqrt{n} \log n + k \log n, \sqrt{n} \log^2 n + k\})$ query time using an algorithm given in [CJ92], which is based on spanning paths of low stabbing number.

Finally, we discuss how vertical strips are handled. Let $\text{HD}(C)$ be the horizontal line segment that extends a distance of r on each side of q. It is easy to prove that a vertical line $\ell \in S$ intersects $C(q, r)$ iff ℓ intersects $\text{HD}(C)$. This observation reduces the problem to a 1-dimensional range reporting problem since ℓ intersects $\text{HD}(C)$ iff ℓ's projection on the x-axis (a point) lies inside the x-projection of $\text{HD}(C)$ (an interval). The counting (resp. reporting) problem can be solved in $O(n)$ space and $O(\log n\ (+k))$ query time by storing the x-projections of the vertical lines in sorted order in an array and doing a binary search for each endpoint of the query interval. Thus, vertical lines do not affect the asymptotic complexity of the solutions presented earlier.

Theorem 4.2 *A set S of n lines in the plane can be stored in a data structure of size $O(n^2)$, (resp. $O(n)$), such that the lines that are intersected by a fixed-radius query circle can be counted (resp. reported) in $O(\log n\ (+k))$ (resp. $O(n^{1/2+\epsilon}\ (+k))$) time, where $\epsilon > 0$ is an arbitrarily small constant. Moreover, the reporting problem can also be solved in $O(n \log n)$ space with a query time of $O(\min\{\sqrt{n}\log n + k\log n, \sqrt{n}\log^2 n + k\})$.* □

4.2 Querying with a variable-radius circular arc

Let γ be the variable-radius query arc. Let $circ(\gamma)$ denote the circle that γ is a part of, let γ' denote the closure of $circ(\gamma) - \gamma$, let $chord(\gamma)$ denote the line segment joining the endpoints of γ, and let $center(\gamma)$ and $radius(\gamma)$ denote, respectively, the center and radius of $circ(\gamma)$.

Lemma 4.2 [AvKO] *A line ℓ intersects a circular arc γ iff one of the following two conditions is satisfied:*

(i) ℓ separates the endpoints of γ, or

(ii) both endpoints of γ and the circular arc γ' lie on the same side of ℓ, and ℓ intersects circ(γ). □

Let us first consider only the non-vertical lines of S. Lines ℓ satisfying condition (i) of the lemma must intersect $chord(\gamma)$ and so can be handled as follows: By duality, ℓ intersects $chord(\gamma)$ iff $\mathcal{F}(\ell)$ lies in the doublewedge, W, formed by $\mathcal{F}(a)$ and $\mathcal{F}(b)$, where a and b are the endpoints of γ. In [CW89] it is shown how to count or report the points in a fixed-size convex polygon in $O(n)$ space and $O(\sqrt{n}\log n\ (+k))$ time. We simply apply this query twice, once for each wedge of W. Alternatively, we can use two halfplane compositions based on cutting trees (Theorem 2.4(i)) to solve the problem in $O(n^{2+\epsilon})$ space and $O(\log n\ (+k))$ query time.

For condition (ii), we need to (a) find those lines ℓ such that $\overline{ab}$ and γ' are on the same side of ℓ and (b) among these lines find those that intersect $circ(\gamma)$. Wlog assume that γ lies above $\overline{ab}$; if $\overline{ab}$ is vertical, then wlog assume that γ lies to its left. Then it is clear that a and b must both lie below any line ℓ satisfying condition (a) above. That is, by duality, $\mathcal{F}(\ell)$ must lie in $\mathcal{F}(a)^- \cap \mathcal{F}(b)^-$. Thus condition (a) reduces to two halfplane compositions.

Moreover, as seen in Section 4.1, condition (b) can be reduced to two halfspace compositions. It follows now that we can determine lines satisfying condition (ii) by using two halfplane compositions and then two halfspace compositions. Applying Theorem 2.4(i) (resp.

Theorem 2.4(ii)), we get a data structure of size $O(n^{3+\epsilon'})$ (resp. $O(n)$) and a query time of $O(\log n\ (+k))$ (resp. $O(n^{2/3+\epsilon'}\ (+k))$).

Finally, we discuss how to handle vertical lines in S. Clearly, a vertical line ℓ intersects γ iff ℓ's abscissa lies in the interval that is obtained by projecting γ onto the x-axis. Thus the problem reduces to 1-dimensional range searching, which can be solved in $O(n)$ space and $O(\log n\ (+k))$ query time.

Note that the sets of lines in S satisfying condition (i), condition (ii), and the set of vertical lines are mutually disjoint. Thus for the counting problem, we simply sum up the counts obtained for each case. We may now conclude:

Theorem 4.3 *A set S of n lines in the plane can be preprocessed into a data structure of size $O(n^{3+\epsilon})$ (resp. $O(n)$), so that given any variable-radius query circular arc γ, we can count or report the lines intersected by γ in $O(\log n(+k))$ (resp. $O(n^{2/3+\epsilon}\ (+k))$) time. Here $\epsilon > 0$ is an arbitrarily small constant.* □

5 Intersection searching on line segments

In this section we assume that the given set S consists of finite line segments and that queries are variable-radius circles or disks. We first consider the case where all the segments are parallel (say horizontal), which, for query circles, already turns out to be an interesting problem, and later consider the case of arbitrary segments.

First some notation. We denote the query disk by D, its center by $center(D)$, and its radius by $radius(D)$. Let $\mathrm{VD}(D)$ denote the vertical line segment of length $2 \cdot radius(D)$ centered at $center(D)$. Analogous definitions apply when the query is a circle C.

5.1 Querying horizontal line segments with a variable-radius circle

The following lemma characterizes the intersections between a horizontal line segment and the query circle C.

Lemma 5.1 *Let* $\mathrm{int}(C)$ *denote the interior of the closed disk bounded by C. A circle C intersects a horizontal line segment s iff at least one of the following two conditions is satisfied:*
(i) $\mathrm{int}(C)$ *or C contains exactly one of the endpoints of s, or*
(ii) $VD(C)$ intersects s and $\mathrm{int}(C)$ *contains no endpoints of s.*

Proof Let $s = \overline{ab}$ and let $ext(C)$ be the exterior of the closed disk bounded by C. If s intersects C only once, then it must be of one of the following types: (a) $a \in int(C)$ and $b \in ext(C)$, or (b) $a \in int(C)$ and $b \in C$, or (c) $a \in C$ and $b \in ext(C)$. All three cases are covered by condition (i). If s intersects C twice, then it must intersect $\mathrm{VD}(C)$ and, moreover, a and b must both be in $C \cup ext(C)$. This is covered by condition (ii). Finally, if s does not intersect C at all, then either a and b are both in $ext(C)$ and s does not intersect $\mathrm{VD}(C)$ or a and b are both in $int(C)$. In either case, neither condition (i) nor condition (ii) holds. □

We determine the segments satisfying condition (i) by extending to $\mathcal{R}^3$ the algorithm of Cheng and Janardan [CJ92] which reports intersections between a query line and a set of possibly intersecting line segments in $\mathcal{R}^2$.

Let us map C to the plane $H = \varphi(C)$ and map each segment $s = \overline{ab}$ to a segment $s' = \overline{\psi(a)\psi(b)}$ in 3-space. Recall that H^+ (resp. H^-) denotes the closed halfspace above (resp. below) H. Also, let $\check{H}^+$ and $\check{H}^-$ be the corresponding open halfspaces. Referring to the proof of Lemma 5.1, if $s = \overline{ab}$ is of type (a), (b), or (c), then correspondingly we have: (a) $\psi(a) \in \check{H}^-$ and $\psi(b) \in \check{H}^+$, or (b) $\psi(a) \in \check{H}^-$ and $\psi(b) \in H$, or (c) $\psi(a) \in H$ and $\psi(b) \in \check{H}^+$.

We now show how to report the types (a)–(c) segments. (Throughout, we use the notation of Section 2.2.1.) We build a spanning path Π of stabbing number $O(\sqrt{n})$ (w.r.t. disks) on the endpoints of the segments in S and then build an s-tree T on Π. For any node $v \in T$, let $S'(v) = \{\psi(p) \mid p \in S(v)\}$. At v, we store an instance, $I(v)$, of the halfspace reporting structure given in [AHL90] (this is in addition to $\mathrm{CH}(v)$). $I(v)$ is constructed on a set $S''(v)$ of points that is obtained by including for each point $\psi(p) \in S'(v)$, the other endpoint of the segment of which $\psi(p)$ is an endpoint.

To answer a query, we first find the set V_{H^+} of canonical nodes w.r.t. H^+. At each node $v \in V_{H^+}$, we apply a halfspace reporting query using $\check{H}^-$ and for each point $\psi(p)$ found, we report the segment $s \in S$ having p as an endpoint. Next, we repeat the above steps with H^+ replaced by H^- and $\check{H}^-$ replaced by $\check{H}^+$.

Lemma 5.2 *Given a set S of n horizontal line segments in the plane, the k segments that satisfy condition (i) of Lemma 5.1 w.r.t. any query circle C can be found in $O(\sqrt{n}\log^2 n + k)$ time using $O(n \log^2 n)$ space.*

Proof From the preceding discussion it should be clear that the first query, using H^+, reports all type (a) and type (b) segments and that the second query, using H^-, reports all type (a) and type (c) segments. Morever, any segment s which does not satisfy condition (i) of Lemma 5.1 must have both endpoints in $int(C)$ or in $ext(C)$ or on C. Thus, the corresponding

segment s' has both endpoints in $\check{H}^-$ or in $\check{H}^+$ or on H. Since in each query we query first with a closed halfspace, the first two types of segments will not be reported. Moreover, since in each query the halfspace reporting query is done with an open halfspace, the third type of segment is also not reported. This proves the correctness of the method.

For the running time, note that $|V_{H^+}| = O(\sqrt{n}\log^2 n)$, since Π is constructed w.r.t. disks. As seen in Section 2.2.1, the time to compute V_{H^+} is $O(\sqrt{n}\log^2 n)$. Since a halfspace query on an m-point set takes $O(\log m)$ time [AHL90], the total time for the first query (and similarly for the second one) is $O(\sqrt{n}\log^2 n)$. The space used by the halfspace query structure is $O(m \log m)$, which sums up to $O(n \log n)$ across each level and, hence, to $O(n \log^2 n)$ overall. □

Let us now consider how to find the segments satisfying condition (ii) of Lemma 5.1. First, we note that the data structure of Lemma 5.2 also allows us to find, within the same bounds, the k segments $s = \overline{ab} \in S$ that have both endpoints in $C \cup ext(C)$, i.e., intersect C twice. This is because then $s' = \overline{\psi(a)\psi(b)}$ is such that both endpoints are in $H \cup \check{H}^+ = H^+$. Thus we can first find V_{H^+} and then answer a halfspace reporting query at each $v \in V_{H^+}$ using H^+ again as the query. Let us call this structure D.

Second, let us represent each $s \in S$ as a triplet (s_1, s_2, s_3), where (s_1, s_3) and (s_2, s_3) are the left and right endpoints of s, respectively. Similarly, we represent $\mathrm{VD}(C)$ as a triplet (v_1, v_2, v_3), where (v_1, v_2) and (v_1, v_3) are the lower and upper endpoints of $\mathrm{VD}(C)$. Then, s intersects $\mathrm{VD}(C)$ iff $s_1 \leq v_1$ and $s_2 \geq v_1$ and $v_2 \leq s_3 \leq v_3$.

Thus, our problem reduces to finding among the segments that satisfy the above inequalities the ones that are intersected twice by C. We can do this by building a 4–level binary tree, where the innermost level is the structure D described above and each of the other levels merely adds a range restriction as specified by the above inequalities.

Thus we have shown:

Lemma 5.3 *Given a set S of n horizontal line segments in the plane, the k segments that satisfy condition (ii) of Lemma 5.1 w.r.t. any query circle can be found in $O(\sqrt{n}\log^5 n + k)$ time using $O(n \log^5 n)$ space.*

Proof The correctness of the method is clear from the preceding discussion. For the time and space bounds, we note that each of the first three levels contributes a logarithmic factor to the bounds of the structure D. □

From Lemmas 5.1–5.3 we get the main result of this section:

Theorem 5.1 *A set S of n horizontal line segments in the plane can be preprocessed into a data structure of size $O(n \log^5 n)$ such that the k segments that are intersected by a variable-*

radius query circle C can be reported in time $O(\sqrt{n}\log^5 n + k)$. □

5.1.1 The case of a variable-radius query disk

For a query circle, we had to be careful not to report those segments that had both endpoints in $int(C)$—which is what made the problem challenging. This restriction does not apply to a query disk and leads to a simpler and more efficient solution, which is based on the following easily-proved lemma:

Lemma 5.4 *A disk D intersects a horizontal segment s iff at least one of the following two conditions is satisfied:*
(i) D contains at least one of the endpoints of s, or
(ii) $VD(D)$ intersects s. □

Segments satisfying condition (i) in Lemma 5.4 can be reported by preprocessing the endpoints of the segments so that the points lying inside D can be reported efficiently. This can be done in $O(n \log n)$ space and $O(\log n + k)$ query time using the algorithm of Aggarwal et al. [AHL90]. (If $radius(D)$ is fixed, then the space drops to $O(n)$ using the algorithm of Lenhof and Smid [LS91].) To report the segments satisfying condition (ii), we organize the segments into a hive graph [Cha86] and query it with VD(C). This takes $O(n)$ space and $O(\log n + k)$ query time.

Theorem 5.2 *A set S of n horizontal line segments in the plane can be preprocessed into a data structure of size $O(n \log n)$, so that the k segments that are intersected by a variable-radius query disk D can be reported in $O(\log n + k)$ time. For the fixed-radius case, the space bound is $O(n)$.* □

5.2 Querying arbitrary line segments with a variable-radius disk

Unlike the case of horizontal segments, it is now possible for a segment of S to intersect D twice without intersecting VD(D). Hence, we use the following characterization:

Lemma 5.5 [AvKO] *For any segment s, let* strip_s *be the closed region of the plane that is bounded by the two lines perpendicular to s and passing through its endpoints. A segment $s \in S$ is intersected by a disk D iff at least one of the following conditions is satisfied:*

(i) D contains at least one of the endpoints of s, or

(ii) center$(D) \in$ strip$_s$ *and D intersects* ℓ_s*, where* ℓ_s *is the supporting line of s.* □

To report the k segments of S that satisfy condition (i) of Lemma 5.5, we can use the approach given in Section 5.1.1, which takes $O(n \log n)$ space and $O(\log n + k)$ query time. Alternatively, we can solve the problem as follows: We map each segment endpoint p in S to a point $\psi(p)$ in $\mathcal{R}^3$ and then build a partition tree on the set of transformed points. Clearly, given a query disk D, with bounding circle C, we can solve our problem by determining the points that lie in $\varphi(C)^-$ and for each such point $\psi(p)$ reporting the segment of S for which p is an endpoint. The halfspace query can be done in $O(n)$ space and $O(n^{2/3+\epsilon} + k)$ query time by using Theorem 2.4(ii).

For condition (ii), let us first consider how to report the k segments $s \in S$ such that $center(D) \in strip_s$. By applying the transform $\mathcal{F}$, we can dualize each strip to a vertical line segment and can dualize $center(D)$ to a line. Thus the problem is equivalent to reporting the vertical line segments that are intersected by $\mathcal{F}(center(D))$. As seen in Section 4.1.1, this problem is solvable in $O(n)$ space and $O(n^{1/2+\epsilon}+k)$ query time via two halfplane compositions based on Theorem 2.4(ii). It is also solvable in $O(n^{2+\epsilon})$ space and $O(\log n + k)$ query time by using Theorem 2.4(i).

Now, let us consider how to report the k segments $s \in S$ such that ℓ_s is intersected by D. As seen in Section 4.1, this problem can be transformed to that of reporting among a set of line segments in $\mathcal{R}^3$ the ones that lie completely on one side of a query halfspace. The latter problem is solvable via two halfspace compositions in $O(n)$ space and $O(n^{2/3+\epsilon} + k)$ query time or, alternatively, in $O(n^{3+\epsilon})$ space and $O(\log n + k)$ query time.

It should now be clear that condition (ii) of Lemma 5.5 can be handled by applying two halfplane and two halfspace compositions in the context of Theorem 2.4. The time and space bounds are dominated by those for the halfspace compositions.

From the preceding discussion we get:

Theorem 5.3 *A set S of n arbitrarily-oriented line segments in the plane can be preprocessed into a data structure of size* $O(n)$ *(resp.* $O(n^{3+\epsilon})$*) such that the k line segments that are intersected by a variable-radius query disk D can be reported in* $O(n^{2/3+\epsilon}+k)$ *(resp.* $O(\log n + k)$*) time, where* $\epsilon > 0$ *is an arbitrarily small constant.* □

5.2.1 The case of a fixed-radius disk

The k segments of S that satisfy condition (i) of Lemma 5.5 can be found in $O(\log n+k)$ time and $O(n)$ space using the approach given in Section 5.1.1. To find the segments satisfying condition (ii), note that, in addition to lying in $strip_s$, $center(D)$ must be within distance $radius(D)$ of each such segment s. In other words, $center(D)$ must lie in the rectangle obtained by truncating $strip_s$ at a distance of $radius(D)$ on each side of s. Thus, our problem reduces to point enclosure searching in a set of arbitrarily-oriented rectangles. This can be solved by dividing each rectangle into two triangles and using the triangle stabbing algorithm given in [CJ92], which uses $O(n \log^2 n)$ space and reports the triangles that contain a query point in time $O(\min\{\sqrt{n}\log^2 n + k, \sqrt{n}\log n + k\log n\})$.

Theorem 5.4 *A set S of n arbitrarily-oriented line segments in the plane can be preprocessed into a data structure of size $O(n\log^2 n)$ such that the k segments that are intersected by a fixed-radius query disk D can be reported in $O(\min\{\sqrt{n}\log^2 n+k, \sqrt{n}\log n+k\log n\})$ time.* □

5.3 Querying with a variable-radius circle

The following lemma, which is easily derived from Lemmas 5.1 and 5.5, characterizes intersections between an arbitrary segment and a circle C.

Lemma 5.6 *A circle C intersects an arbitrarily-oriented segment $s = \overline{ab}$ iff at least one of the following conditions is true:*

(i) $\text{int}(C)$ *or C contains exactly one endpoint of s, or*

(ii) $\text{center}(C) \in \text{strip}_s$ *and C intersects the supporting line, ℓ_s, of s.* □

As seen in Section 5.1 (proof of Lemma 5.1), segments that satisfy condition (i) of Lemma 5.6 are of one of the types (a)–(c) and can be found via two halfspace compositions, one using H^+ and $\check{H}^-$ and the other using H^- and $\check{H}^+$. This takes $O(n)$ (resp. $O(n^{3+\epsilon})$) space and $O(n^{2/3+\epsilon} + k)$ (resp. $O(\log n + k)$) query time. Segments satisfying condition (ii) can be found as we did for variable-radius disk reporting (see the discussion preceding Theorem 5.3) in the same bounds as for condition (i). Thus we have:

Theorem 5.5 *A set S of n arbitrarily-oriented line segments in the plane can be preprocessed into a data structure of size $O(n)$ (resp. $O(n^{3+\epsilon})$) such that the k segments intersecting a variable radius query circle C, can be reported in time $O(n^{2/3+\epsilon}+k)$ (resp. $O(\log n+k)$).* □

6 Intersection searching on curved objects with curved query objects

So far we have considered problems where the input objects are linear and the query object is curved. We now turn to the second class of problems considered in this paper, namely where both the input objects and the query objects are curved. Table 2 summarizes our results.

6.1 Querying d-balls with a d-ball

In this section, we prove the following:

Theorem 6.1 *Let $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$ be a collection of closed d-balls in $\mathcal{R}^d$, $d \geq 2$, of possibly different radii. $\mathcal{B}$ can be preprocessed into a data structure of size $O(n)$ (resp. $O(n^{d+2+\epsilon})$) such that the d-balls that are intersected by a variable-radius d-ball, Q, can be counted or reported in time $O(n^{1-1/(d+2)}(\log\log n)^{O(1)}\ (+k))$ (resp. $O(\log n\ (+k))$). Here $\epsilon > 0$ is an arbitrarily small constant and k is the number of intersected d-balls. Moreover, for the reporting case, there is also a solution which uses $O(n \log\log n)$ space and has a query time of $O(n^{1-1/\lfloor (d+2)/2 \rfloor}(\log n)^{O(1)} + k)$.*

Proof First some notation. We use the coordinates $x_1, x_2, \ldots, x_d$ in $\mathcal{R}^d$. Let $C_i = (b_{i1}, \ldots, b_{id})$ be the center of B_i and r_i its radius, $1 \leq i \leq n$. Let $C_Q = (q_1, \ldots, q_d)$ be the center of Q and r_Q its radius.

If Q intersects some B_i, then either their boundaries intersect or one is contained inside the other. It is easy to prove that Q intersects B_i iff the Euclidean distance between their centers is at most $r_i + r_Q$, i.e.,

$$Q \text{ intersects } B_i \text{ iff } \sum_{j=1}^{d}(b_{ij} - q_j)^2 \leq (r_i + r_Q)^2. \tag{1}$$

Let us define a transformation, τ, which maps B_i to a point in $\mathcal{R}^{d+2}$, as follows:

$$\tau(B_i) = (b_{i1}, \ldots, b_{id}, r_i, b_{i1}^2 + \cdots + b_{id}^2 - r_i^2). \tag{2}$$

Also, let us define a transformation, α, which maps Q to a hyperplane in $\mathcal{R}^{d+2}$, as follows:

$$\alpha(Q) : x_{d+2} = 2q_1x_1 + \cdots + 2q_dx_d + 2r_Qx_{d+1} - q_1^2 - \cdots - q_d^2 + r_Q^2. \tag{3}$$

Claim: Q intersects B_i iff $\tau(B_i) \in \alpha(Q)^-$, where $\alpha(Q)^-$ is the closed halfspace lying below $\alpha(Q)$.

Proof of claim:

By Equations (1)–(3), it is sufficient to show that

$$\sum_{j=1}^{d}(b_{ij}-q_j)^2 \leq (r_i+r_Q)^2 \text{ iff } b_{i1}^2+\cdots+b_{id}^2-r_i^2 \leq 2q_1b_{i1}+\cdots+2q_db_{id}+2r_Qr_i-q_1^2-\ldots-q_d^2+r_Q^2. \tag{4}$$

This can be verified by straightforward algebraic manipulation.

End of proof of claim

Thus, we have transformed our d-ball problem in $\mathcal{R}^d$ to halfspace searching in $\mathcal{R}^{d+2}$. In [Mat92], it is shown how the latter problem can be solved in $O(n^{1-1/(d+2)}(\log\log n)^{O(1)}\ (+k))$ query time using $O(n)$ space. Alternatively, we can use Theorem 2.2 to solve the problem in $O(\log n\ (+k))$ query time using $O(n^{d+2+\epsilon})$ space. If we are interested only in the reporting problem, then we can also use the solution given in [Mat91b] to solve the problem with a query time of $O(n^{1-1/\lfloor(d+2)/2\rfloor}(\log n)^{O(1)}+k)$ using $O(n\log\log n)$ space. □

Remark 6.1 For $d=2$, the $O(n^{4+\epsilon})$ space bound given by Theorem 6.1 can be improved to $O(n^{3+\epsilon})$, without affecting the $O(\log n\ (+k))$ query time, as follows: In Section 2.1 of [AS91], Agarwal and Sharir show how to count efficiently the intersections between m red circles and n blue circles in the plane. Their approach can be adapted to the query mode by taking one red query circle and n blue input circles and can be made to work for disks as well.

Moreover, their approach can be extended to the reporting case also. A straightforward implementation gives an $O(n^4)$ space bound and $O(\log n+k)$ query time: As in [AS91], we compute a certain family of n arrangements in two dimensions, each composed of $\Theta(n)$ conic plane curves. For each arrangement, we store with each face a list of $O(n)$ blue circles that satisfy a certain incidence property w.r.t. the face. As there are $O(n^2)$ faces per arrangement, the space used is $O(n^3)$ per arrangement, hence $O(n^4)$ overall. To improve upon this, we store the lists for each arrangement in a persistent way. (We will not discuss this in detail, as we will examine this idea at length in Section 7 for a different set of problems.) This reduces the space to $O(n^2)$ per arrangement, hence $O(n^3)$ overall. The overall space is now dominated by the $O(n^{3+\epsilon})$ space required by a spatial point location scheme employed in [AS91].

6.2 Querying d-spheres with a d-sphere

By a *d-sphere*, S_i, we mean the boundary of the closed d-ball B_i, with radius r_i and center $C_i = (b_{i1}, b_{i2}, \ldots, b_{id})$, $1 \leq i \leq n$. (For example, a 2-sphere is a circle.) We show how to extend the ideas of Section 6.1 to preprocess a set $\mathcal{S} = (S_1, S_2, \ldots, S_n)$ of d-spheres so that counting and reporting queries with a variable-radius d-sphere Q can be answered efficiently.

Let $C_Q = (q_1, q_2, \ldots, q_d)$ be the center of Q and r_Q its radius. Q intersects S_i iff the d-balls that they bound intersect and neither d-ball is contained in the other, i.e., iff the Euclidean distance between C_i and C_Q is at most $r_i + r_Q$ and at least $|r_i - r_Q|$. In other words,

$$Q \text{ intersects } S_i \text{ iff } (r_i - r_Q)^2 \leq \sum_{j=1}^{d} (b_{ij} - q_j)^2 \leq (r_i + r_Q)^2. \tag{5}$$

Consider the following transformation, β, which maps Q to a hyperplane in $\mathcal{R}^{d+2}$:

$$\beta(Q) : x_{d+2} = 2q_1x_1 + \cdots + 2q_dx_d - 2r_Qx_{d+1} - q_1^2 - \cdots - q_d^2 + r_Q^2. \tag{6}$$

In addition, consider the transformations τ and α of Section 6.1, which can be extended in the obvious way to d-spheres. It is easily verified that Q intersects S_i iff $\tau(S_i) \in \alpha(Q)^- \cap \beta(Q)^+$, where $\beta(Q)^+$ is the closed hyperplane above $\beta(Q)$. The points $\tau(S_i)$ satisfying the above condition can be found via two halfspace compositions. From Theorem 2.4 we conclude directly:

Theorem 6.2 *Let $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ be a collection of d-spheres in $\mathcal{R}^d$, $d \geq 2$, of possibly different radii. $\mathcal{S}$ can be preprocessed into a data structure of size $O(n)$ (resp. $O(n^{d+2+\epsilon})$) such that the d-spheres that are intersected by a variable-radius d-sphere, Q, can be counted or reported in time $O(n^{1-1/(d+2)}(\log\log n)^{O(1)}\ (+k))$ (resp. $O(\log n\ (+k))$). Here $\epsilon > 0$ is an arbitrarily small constant and k is the number of intersected d-spheres.* □

Remark 6.2 From Remark 6.1, the above problem can be solved in $O(n^{3+\epsilon})$ space and $O(\log n\ (+k))$ query time for circles.

6.3 Querying circular arcs with a circle

We show how to preprocess a set S of n circular arcs, of possibly different radii, so that the ones that are intersected by a variable-radius query circle C can be reported efficiently.

Wlog we may assume that the arcs in S are *x-monotone*, i.e., any vertical line intersects an arc in at most one point, since any arc can be decomposed into at most three x-monotone pieces.

Recall that for an arc $\gamma \in S$, $chord(\gamma)$ is the line segment joining its endpoints, $circ(\gamma)$ is γ's supporting circle, and $center(\gamma)$ is the center of $circ(\gamma)$. Let $l(\gamma)$ and $r(\gamma)$ be γ's left and right endpoints, respectively. Let $disk(C)$ be the closed disk bounded by C. Define ℓ_l^{in} (resp. ℓ_l^{out}) to be the halfplane bounded by the line joining $center(\gamma)$ and $l(\gamma)$ and containing (resp. not containing) γ. Define ℓ_r^{in} and ℓ_r^{out} similarly. The following lemma follows from the discussions in [Pel92].

Lemma 6.1 *A circle C and an x-monotone arc γ intersect iff one of the following conditions is true:*

(i) C separates the endpoints of γ

(ii) C intersects $\mathrm{circ}(\gamma)$ *and*

(a) $l(\gamma) \notin \mathrm{disk}(C)$ and $r(\gamma) \notin \mathrm{disk}(C)$, and $\mathrm{center}(C) \in \ell_l^{in} \cap \ell_r^{in}$ *or*

(b) $l(\gamma) \in \mathrm{disk}(C)$ and $r(\gamma) \in \mathrm{disk}(C)$, and $\mathrm{center}(C) \in \ell_l^{out} \cap \ell_r^{out}$. □

An arc γ satisfies condition (i) iff C stabs $chord(\gamma)$. To report such arcs, we apply the transform ψ to the endpoints of γ and the transform φ to C and convert the problem to one of reporting those line segments $\overline{\psi(l(\gamma))\psi(r(\gamma))}$ in $\mathcal{R}^3$ whose endpoints are in opposite halfspaces of $\varphi(C)$. This can be done via two halfspace compositions in $O(n)$ (resp. $O(n^{3+\epsilon})$) space and $O(n^{2/3+\epsilon} + k)$ (resp. $O(\log n + k)$) query time (Theorem 2.4).

Let us now consider how to report arcs that satisfy condition (ii). First consider condition (ii)(a). Using the transforms ψ and φ, the condition "$l(\gamma) \notin disk(C)$ and $r(\gamma) \notin disk(C)$" transforms to "$\overline{\psi(l(\gamma))\psi(r(\gamma))}$ lies above $\varphi(C)$ in $\mathcal{R}^3$". This in turn can be expressed as two halfspace compositions in $\mathcal{R}^3$.

Next, note that the wedge $\ell_l^{in} \cap \ell_l^{out}$ is of one of the three forms $t_1^+ \cap t_2^+$, $t_1^- \cap t_2^-$, or $t_1^+ \cap t_2^-$ for some lines t_1 and t_2. Thus, by applying the transform $\mathcal{F}$ to C, t_1, and t_2, the condition "$center(C) \in \ell_l^{in} \cap \ell_r^{in}$", transforms to "$\mathcal{F}(center(C))$ is below or above or intersects $\overline{\mathcal{F}(t_1)\mathcal{F}(t_2)}$". Each of these three cases can in turn be expressed as two halfplane compositions in $\mathcal{R}^2$.

Similarly, condition (ii)(b) can be expressed as two halfspace and two halfplane compositions.

We are now ready to apply Theorem 2.4. As the structure D of that theorem, we take the $O(n)$-space structure of Theorem 6.1, for $d = 2$, to report the circles $circ(\gamma)$ intersected by C.

This structure has a query time of $O(n^{3/4+\epsilon}+k)$. Alternatively, we can take the $O(n^{3+\epsilon})$-space structure with query time $O(\log n+k)$ mentioned in Remark 6.2. We then build a 4-level tree structure, where the nodes at the innermost level are augmented with instances of D. The two outermost levels apply halfspace compositions and the next two levels apply halfplane compositions. To answer a query, we first apply the halfspace compositions corresponding to the first part of condition (ii)(a), then, in turn, each of the three halfplane compositions corresponding to the second part, and finally query the structure D at the nodes of the innermost tree that are visited. We then repeat this for condition (ii)(b).

The correctness and bounds follow from the above discussion and from Theorem 2.4.

Theorem 6.3 *A set S of n circular arcs in the plane, of possibly different radii, can be preprocessed into a data structure of size $O(n)$ (resp. $O(n^{3+\epsilon})$) such that the k arcs that are intersected by a variable-radius query circle C can be reported in time $O(n^{3/4+\epsilon}+k)$ (resp. $O(\log n+k)$).* □

Remark 6.3 In [AS91], the counting problem is solved in $O(n^{3+\epsilon})$ space and $O(\log n)$ query time. As in Remark 6.1, we can extend this approach to solve the reporting problem in $O(n^{3+\epsilon})$ space and $O(\log n+k)$ query time.

6.4 Querying circles with a circular arc

Again, we may assume that the query arc, γ, is x-monotone. Lemma 6.1 still applies; however, the approach is somewhat different since the roles of arcs and circles is now reversed.

To report the circles that separate the endpoints of γ (condition (i) of Lemma 6.1), we map each circle C to the point $\mathcal{F}(\varphi(C))$ in $\mathcal{R}^3$ and the endpoints $l(\gamma)$ and $r(\gamma)$ of γ to the planes $H_l=\mathcal{F}(\psi(l(\gamma)))$ and $H_r=\mathcal{F}(\psi(r(\gamma)))$, respectively. It is then clear that the original problem has been transformed to one of reporting among a set of points in $\mathcal{R}^3$ those that lie in a query doublewedge formed by H_l and H_r. This problem can be solved by two applications of two halfspace compositions in $\mathcal{R}^3$.

Let us now consider condition (ii)(a) of Lemma 6.1. By using $\mathcal{F}$, φ, and ψ as above, the condition "$l(\gamma)\notin \mathit{disk}(C)$ and $r(\gamma)\notin \mathit{disk}(C)$" transforms to "$\mathcal{F}(\varphi(C))$ lies in $H_l^+\cap H_r^+$". This in turn can be expressed as two halfspace compositions in $\mathcal{R}^3$. Moreover, the condition "$\mathit{center}(C)\in \ell_l^{in}\cap \ell_r^{in}$" can be expressed immediately as two halfplane compositions in $\mathcal{R}^2$. A similar discussion applies for condition (ii)(b).

We can now apply Theorem 2.4 to report the segments that satisfy condition (ii). We choose the structure D as in Section 6.3 and build a 4-level tree structure as before. To

query, we first apply the halfspace compositions, then the halfplane compositions, and then finally query D at the nodes of the innermost tree that are visited.

Theorem 6.4 *A set S of n circles in the plane, of possibly different radii, can be preprocessed into a data structure of size $O(n)$ (resp. $O(n^{3+\epsilon})$) such that the k circles that are intersected by a variable-radius query circular arc γ can be reported in time $O(n^{3/4+\epsilon}+k)$ (resp. $O(\log n+k)$).* □

7 Generalized intersection searching with curved objects

We now turn to the third class of problems considered in this paper: Each object (points, lines, line segments, or disks) in the input set S is assigned a color. (The number of colors used can range from 1 to n; thus several objects can receive the same color. Wlog we assume that the colors are integers in the range $[1, n]$.) The goal is to preprocess S so that the i distinct colors of the objects that are intersected by a query object (fixed- or variable-radius, circle, disk, or annulus) can be counted or reported efficiently. Table 3 summarizes our results.

We begin with a brief review of persistent data structures and also introduce a method for ordering the faces of a planar subdivision—techniques that we use extensively in this section.

7.1 Persistent data structures

Ordinary data structures are *ephemeral* in the sense that once an update is performed the previous version is no longer available. In contrast, a *persistent* data structure supports operations on the most recent version as well as on previous versions. A persistent data structure is *partially persistent* if any version can be accessed but only the most recent one can be updated; it is *fully persistent* if any version can be both accessed and updated.

In [DSST89], Driscoll *et al* describe a general technique to make persistent any ephemeral linked data structure. A *linked data structure* consists of a finite collection of nodes, each with a fixed number of fields. Each field can hold either a piece of data such as, say, an integer or a real, or a pointer to another node. The *in-degree* of a node is the number of other nodes pointing to it. Access to the structure is accomplished via one or more access pointers. Examples of linked structures include linked lists and balanced binary search trees.

An update operation typically modifies one or more fields in the structure. We will call each modification a *memory modification.* Driscoll *et al* showed that any linked structure whose nodes have constant in-degree can be made partially or fully persistent such that each memory modification in the ephemeral structure adds just $O(1)$ amortized space to the persistent structure and, moreover, the query time of the persistent structure is only a constant factor larger than that of the ephemeral structure.

For our purposes, partial persistence suffices. Our general approach is to begin with some ephemeral dynamic linked structure (e.g., a linked list or a binary search tree), which has one access pointer. Starting with an empty structure, we then perform a suitable sequence of $O(n)$ updates using the technique of Driscoll *et al* and obtain a partially persistent structure, which contains all versions of the ephemeral structure. Each version has an associated "timestamp", usually an x-coordinate (or a y-coordinate) in the input. We store the access pointers of the different versions in an array, sorted by timestamp; thus any desired version can be accessed for querying by doing a binary search in the array. If $m(n)$ is the total number of memory modifications made by the update sequence, then the persistent structure uses $O(m(n))$ space.

7.1.1 Ordering the faces of a planar subdivision

Our solutions for the generalized problems involve constructing a planar subdivision of some kind and storing suitable information with each face of the subdivision. A query is answered by locating a suitable face and reading off the associated information. The straightforward approach of storing the information for each face in its entirety takes up too much space. However, we can exploit the fact that the information associated with "nearby" faces is not too different and apply persistence to reduce the storage significantly. To do so we need a suitable ordering of the faces. The following lemma is the basis for such an ordering; we will use appropriate instantiations of this in later sections. We note that the lemma holds not only for conventional linear subdivisions but also for subdivisions composed of curves.

Lemma 7.1 *Let P be a planar subdivision with m vertices. There exists an ordering, T_P, of the faces of P such that T_P has length $O(m)$, consecutive faces in T_P share an edge, and T_P visits each face of P at least once.*

Proof Let G_P be the graph-theoretic dual of P, defined as follows: For each face (including unbounded faces) in P, create a vertex in G_P. Connect two vertices in G_P by a pair of edges if the corresponding faces in P share an edge. It can be shown that G_P is connected. Moreover, since P has $O(m)$ faces (resp. edges), G_P has $O(m)$ vertices (resp. edges). Since

every vertex of G_P has even degree, G_P contains an Eulerian walk. This walk visits each edge of G_P exactly once and so has length $O(m)$. Also, it visits each vertex at least once. The theorem follows. □

7.2 Generalized annular range searching

Let S be a set of n colored sites (points) in the plane. Let $Ann(q, r_1, r_2)$ be the query annulus of radius r_1 and r_2 ($r_1 \leq r_2$), centered at q. We first consider the case where r_1 and r_2 are fixed and only q varies with the query.

7.2.1 Querying with a fixed-radii annulus

Clearly, a site $p \in S$ is in $Ann(q, r_1, r_2)$ iff q is in $Ann(p, r_1, r_2)$. Extending the idea of Bentley and Maurer [BM79], we construct $Ann(p, r_1, r_2)$ for each $p \in S$ and give it p's color. Let $\mathcal{A}$ be the arrangement of the corresponding $2n$ circles. Each face of $\mathcal{A}$ is the intersection of zero or more of the annuli. We preprocess $\mathcal{A}$ for fast planar point location using, for instance, a straightforward extension of the point location scheme of Sarnak and Tarjan [ST86]. Since $\mathcal{A}$ has $O(n^2)$ vertices, edges (i.e. circular arcs) and faces, this takes $O(n^2)$ space and allows point location in $O(\log n)$ time. To solve the counting (resp. reporting) problem, we determine for each face f the annuli whose intersection is f and store with f, a count, $count_f$, (resp. a list L_f) of the distinct colors of these annuli. To answer the query $Ann(q, r_1, r_2)$, we locate the face f containing q and output $count_f$ or L_f as appropriate. The total space for the counting (resp. reporting) problem is $O(n^2)$ (resp. $O(n^3)$) and the query time is $O(\log n\ (+i))$.

We can reduce the space for the reporting problem to $O(n^2)$ by storing the L-lists in a persistent way. Let $T_{\mathcal{A}} = f_1, f_2, \ldots, f_l$, $l = O(n^2)$, be the ordering of $\mathcal{A}$'s faces, as provided by Lemma 7.1. We initialize an array $C[1:n]$ of colors to zero. Then we construct L_{f_1} (as an unordered doubly-linked list) and increment $C[i]$ for each color i in L_{f_1}. We then scan $T_{\mathcal{A}}$. Let $f_i, i \geq 2$, be the current face. Let a be the edge shared by f_{i-1} and f_i. Assume that a belongs to the outer circle of some annulus A and that a appears convex to points in f_{i-1} and concave to points in f_i. (See Figure 1, which illustrates this for an arrangement of two annuli.) Let A's color be j. If $C[j] = 0$ then we increment it and insert j into $L_{f_{i-1}}$ in a persistent way to get L_{f_i}; otherwise, we merely record the fact that $L_{f_{i-1}} = L_{f_i}$. On the other hand, suppose that a appears concave to points in f_{i-1} and convex to points in f_i (i.e., f_{i-1} and f_i are exchanged in Figure 1). We decrement $C[j]$. If $C[j] = 0$ now then we delete it from $L_{f_{i-1}}$ in a persistent way to get L_{f_i}; otherwise, we just record the fact that

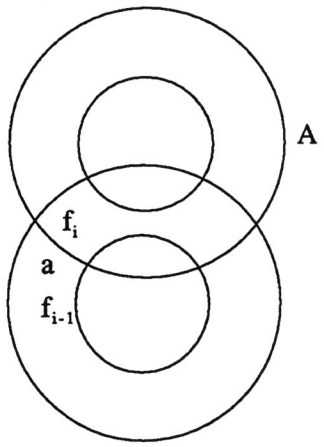

Figure 1: Faces f_i and f_{i-1} sharing an edge which belongs to the outer circle of annulus A. Edge a is convex w.r.t. f_{i-1} and concave w.r.t. f_i.

$L_{f_{i-1}} = L_{f_i}$. (To do the deletion efficiently, we also maintain in $C[j]$ a pointer to j in the current L-list.) A symmetric discussion applies if a belongs to the inner circle of A. Since $T_{\mathcal{A}}$ has length $O(n^2)$, there are $O(n^2)$ updates and so the total space is $O(n^2)$.

Note that the same face can appear in $T_{\mathcal{A}}$ more than once. We scan $T_{\mathcal{A}}$ and pick the first occurrence of each face. Let $f_1 = f'_1, f'_2, \ldots$ be the sequence of faces picked. We store access pointers into the persistent structure to only those versions that correspond to $f'_1, f'_2, \ldots$. To answer a query $Ann(q, r_1, r_2)$, we determine the face f of $\mathcal{A}$ containing q. Let $f = f'_i$. We then query the ith version of the persistent structure and output $L_{f'_i}$. The query time is $O(\log n + i)$.

Theorem 7.1 *A set S of n colored points in the plane can be preprocessed into a data structure of size $O(n^2)$ such that the i distinct colors of the points lying inside any fixed-radii query annulus can be counted (resp. reported) in $O(\log n\ (+i))$ time.* □

7.2.2 Querying with a variable-radii annulus

The approach of Section 7.2.1 does not work if the radii of the query annulus are not known beforehand. In this case, we resort to a different approach of partitioning the plane into regions, as stated in the following lemma.

Lemma 7.2 *Let S be a set of n sites in the plane. Let $\mathcal{A}$ be the arrangement of the perpendicular bisectors of the line segments joining pairs of points in S. Let f be any face of $\mathcal{A}$ and let p_1 and p_2 be any two points in f. Then the ordering of the sites by non-decreasing distance from p_1 is the same as the ordering of the sites by non-decreasing distance from p_2.*

Proof Let a and b be any two sites in S. Let $P(a,b)$ be the perpendicular bisector of the line segment $\overline{ab}$ and let $P_a(a,b)$ (resp. $P_b(a,b)$) be the closed halfplane of $P(a,b)$ which contains a (resp. b). For any point p in the plane $d(p,a) \leq d(p,b)$ (resp. $d(p,a) \geq d(p,b)$) if $p \in P_a(a,b)$ (resp. $P_b(a,b)$), where $d(\cdot,\cdot)$ is the Euclidean distance function.

For a face $f \in \mathcal{A}$, any two sites of S can be compared w.r.t their distances from any point $p \in f$. Thus the sites can be totally ordered by their distances from p. Now f is the intersection of halfplanes $P_x(a,b)$ for all pairs (a,b) of sites in S, where x is either a or b. It follows that the ordering of the sites by nondecreasing distances from different points of f is invariant. □

Let us first consider the reporting problem. We preprocess $\mathcal{A}$ for fast planar point location [ST86]. Given the query $Ann(q, r_1, r_2)$ we locate q in a face f of $\mathcal{A}$. Let $\mathcal{E}(f) = s_1, s_2, \ldots, s_n$ be the ordering of the sites w.r.t. f, as given by Lemma 7.2. By definition of $\mathcal{E}(f)$, the sites (if any) that are at distance at least r_1 and at most r_2 from q (i.e., the ones that are in $Ann(q, r_1, r_2)$) are contiguous in $\mathcal{E}(f)$, say $s_j, \ldots, s_k$, for some $j \geq 1$ and $k \leq n$. Thus, we can restate our generalized annulus problem as: "Given a sequence of colored integers $1, 2, \ldots, n$ on the real line (namely, the indices of the sites in $\mathcal{E}(f)$), report the i distinct colors of the points lying in the interval $[j,k]$." This is just (a special case of) the *generalized 1-dimensional range searching* problem considered in [GJS93]. In [GJS93] it is shown that this problem can be solved in $O(n)$ space and $O(\log n + i)$ query time. Thus, given j and k, we can answer our annulus reporting query in $O(\log n + i)$ time.

But how do we find j and k efficiently? We find j using binary search as follows: Note that s_j is the leftmost site in $\mathcal{E}(f)$ such that $d(q, s_j) \geq r_1$. Let T_f be a red-black tree [CLR90] storing the sites according to the order given by $\mathcal{E}(f)$. Let s be the site at T_f's root. If $d(q,s) \geq r_1$ (resp. $d(q,s) < r_1$) then we visit the left (resp. right) subtree of the root recursively. Let l be the leaf where the search runs off T and let s' be the site stored at l. If the search at l branched left then j is simply the index of s' in $\mathcal{E}(f)$; otherwise, j is the index of the site stored in the inorder successor of l. (If the inorder successor is *nil*, then we stop querying since no site can lie in the query annulus.) Clearly, the time to find j is $O(\log n)$. The index k can be found symmetrically.

Thus the overall query time for the annulus reporting problem is $O(\log n + i)$. For the space bound, note that $\mathcal{A}$ is an arrangement of $\binom{n}{2}$ lines and hence has $O(n^4)$ faces. With each face we store T_f and D_f, where D_f is an instance of the generalized 1-dimensional range searching data structure given in [GJS93]. Each structure uses $O(n)$ space. Thus the total space is $O(n^5)$.

We can reduce the space to $O(n^4 \log n)$ by using persistence, as follows: For faces f and f' of $\mathcal{A}$, let $\Delta(f, f')$ denote the number of positions in which $\mathcal{E}(f)$ and $\mathcal{E}(f')$ differ.

Lemma 7.3 *Let S be a set of n sites in the plane. Let $\mathcal{A}$ be the arrangement of the perpendicular bisectors of the line segments joining pairs of sites in S. There is an ordering, $f'_1, f'_2, f'_3, \ldots, f'_t$, of the t faces of $\mathcal{A}$ such that $\sum_{i=1}^{t-1} \Delta(f'_i, f'_{i+1}) = O(n^4)$.*

Proof Let $T_{\mathcal{A}} = f_1, f_2, \ldots, f_l$ be the ordering of $\mathcal{A}$'s faces, as given by Lemma 7.1, where $l = O(n^4)$. Consider any two consecutive faces f_i and f_{i+1}, $1 \leq i \leq l-1$. Let the supporting line of the edge that f_i and f_{i+1} share be the perpendicular bisector of the sites a and b of S. Then $\mathcal{E}(f_i)$ and $\mathcal{E}(f_{i+1})$ are the same except that the positions of a and b are swapped. Thus $\Delta(f_i, f_{i+1}) = 2$ and so $\sum_{i=1}^{l-1} \Delta(f_i, f_{i+1}) = O(n^4)$. The desired sequence $f'_1, f'_2, \ldots, f'_t$ is obtained by scanning $f_1, f_2, \ldots, f_l$ and taking the first occurrence of each face. The lemma follows since $\sum_{i=1}^{t-1} \Delta(f'_i, f'_{i+1}) \leq \sum_{i=1}^{l-1} \Delta(f_i, f_{i+1})$. □

We can now use the ordering $f'_1, \ldots, f'_t$ provided by Lemma 7.3 to store all the T_f's and D_f's persistently. First recall that T_f is a red-black tree and so supports updates in $O(\log n)$ time with $O(1)$ memory modifications. D_f is essentially a *priority search tree* [McC85] (see [GJS93]) and so supports updates in $O(\log n)$ time with $O(\log n)$ memory modifications. We build $T_{f'_1}$ and $D_{f'_1}$ and then scan $f'_2, \ldots, f'_t$. For $i \geq 2$, we determine the elements in $\mathcal{E}(f'_{i-1})$ whose ranks change in $\mathcal{E}(f'_i)$, delete them from $T_{f'_{i-1}}$ and $D_{f'_{i-1}}$ and reinsert them with their new ranks. These updates are done in a persistent way and yield $T_{f'_i}$ and $D_{f'_i}$. By Lemma 7.3, there are $O(n^4)$ updates. Since each causes $O(\log n)$ memory modifications, the total space is $O(n^4 \log n)$.

We can solve the generalized annulus counting problem in a similar way. For D_f we use the structure given in [GJS93] for generalized 1-dimensional range counting. This structure uses $O(n \log n)$ space and has a query time of $O(\log^2 n)$. The update time is $O(\log^2 n)$ amortized; thus the number of memory modifications per update is also $O(\log^2 n)$ amortized. (As shown in [GJS93], the $O(\log^2 n)$ update time also holds in the worst-case; however, for our discussions the amortized bound is more convenient as it gives the upper bound on the number of memory modifications directly.) By proceeding as in the reporting case, we get a space bound of $O(n^5 \log n)$, which we can reduce, as before, using persistence to $O(n^4 \log^2 n)$. The overall query time is $O(\log^2 n)$. (For the non-persistent scheme, we can also use a static version of D_f, which takes $O(n \log n)$ space and has a query time of $O(\log n)$. This yields an alternative solution for the annulus problem, whose space bound is $O(n^5 \log n)$ and query time is $O(\log n)$.)

Theorem 7.2 *A set S of n colored points in the plane can be preprocessed into a data structure of size $O(n^4 \log^2 n)$ (resp. $O(n^4 \log n)$) such that the i distinct colors of the points lying inside any variable-radii query annulus can be counted (resp. reported) in $O(\log^2 n)$ (resp. $O(\log n + i)$) time. The counting problem can also be solved in $O(n^5 \log n)$ space and $O(\log n)$ query time* □

7.2.3 The special case of a variable-radius disk

If the query annulus is $Ann(q, 0, r_2)$, then we are searching with a variable-radius disk, of some radius r_2. In this case, a slightly better solution than Theorem 7.2 is possible. We discuss this in some detail now, since many of our subsequent schemes use variable-radius query disks. As before we construct the arrangement $\mathcal{A}$ of the perpendicular bisectors and obtain the ordering $\mathcal{E}(f)$ for each face f. Assuming that q falls inside f, the sites at distance at least zero and at most r_2 (the ones of interest) form a prefix, $s_1, \ldots, s_k$, of $\mathcal{E}(f) = s_1, \ldots, s_n$, for some $k \leq n$. Thus our problem translates to a generalized 1-dimensional range searching problem on $1, 2, \ldots, n$ with the query interval $(-\infty, k]$. As shown in [GJS93], for this special query the problem is solvable in $O(n)$ space and $O(\log n\ (+i))$ query time. Moreover, the number of memory modifications is $O(\log n)$ for the counting problem and just $O(1)$ for the reporting problem. By applying the persistence-based approach described in Section 7.2, we immediately get:

Theorem 7.3 *A set S of n colored points in the plane can be preprocessed into a data structure of size $O(n^4 \log n)$ (resp. $O(n^4)$) such that the i distinct colors of the points lying inside any variable-radius query disk can be counted (resp. reported) in $O(\log n\ (+i))$ time.*

7.3 Generalized intersection searching on lines with a query circle

Here S is a set of n colored lines in the plane. Let $C(q, r)$ denote the query circle of radius r, centered at q.

7.3.1 The fixed-radius case

Recall Lemma 4.1. We replace each line ℓ by $strip_\ell$ and give it ℓ's color. We then compute the arrangement $\mathcal{A}$ of the lines bounding the strips and preprocess it for fast planar point location. For each face f of $\mathcal{A}$, we determine the strips whose intersection is f and store with

f a count, $count_f$, or a list, L_f, of the distinct colors of these strips. To answer a query, we locate the face containing q and output the associated information. This takes $O(\log n\ (+i))$ time. The overall space is $O(n^2)$ (resp. $O(n^3)$) for the counting (resp. reporting) problem. By using the persistence-based approach as described in Section 7.2.1, the space for the reporting problem can be reduced to $O(n^2)$.

Theorem 7.4 *A set S of n colored lines in the plane can be preprocessed into a data structure of size $O(n^2)$ such that the i distinct colors of the lines that are intersected by a fixed-radius query circle (or disk) can be counted (resp. reported) in $O(\log n\ (+i))$ time.* □

7.3.2 The variable-radius case

In this case, r is not known beforehand. Given $C(q,r)$, we need to count or report the distinct colors of the lines that are at (perpendicular) distance at most r from q. For this, we partition the plane into regions such that for any point p in a region the ordering of the lines in S by their distances from p is invariant. The following lemma summarizes this approach.

Lemma 7.4 *Let S be a set of n lines in the plane. Let a and b be any two lines in S and consider the two lines that bisect the angles at the point of intersection of a and b. Let $\mathcal{A}$ be the arrangement of such bisectors for all pairs of lines $a,\ b \in S$. Let f be any face of $\mathcal{A}$ and let p_1 and p_2 be any two points in f. Then the ordering of the lines in S by non-decreasing distance from p_1 is the same as their ordering by non-decreasing distance from p_2.*

Proof Let a and b be any two lines in S. At their intersection we have a pair of doublewedges, $DW_a(a,b)$ and $DW_b(a,b)$, containing a and b respectively. (See Figure 2.) Let $\rho(x,y)$ denote the Euclidean distance between point x and line y. By a straightforward geometric argument, for any point p in the plane $\rho(p,a) \leq \rho(p,b)$ (resp. $\rho(p,a) \geq \rho(p,b)$) if $p \in DW_a(a,b)$ (resp. $p \in DW_b(a,b)$).

Given a face $f \in \mathcal{A}$, any two lines of S can be compared w.r.t their distances from any point $p \in f$. Thus the lines can be totally ordered by their distances from p. Now f is the intersection of doublewedges $DW_x(a,b)$ for all pairs (a,b) of lines in S, where x is either a or b. It follows that the ordering of the lines by nondecreasing distances from different points in f is invariant. □

Let $\mathcal{E}(f)$ denote the above ordering for face f and suppose that q lies in f. As in Section 7.2.3, we can interpret our problem as a generalized 1-dimensional counting or reporting problem on $1,2,\ldots,n$ with a query interval of the form $(-\infty,k]$. Thus the counting (resp.

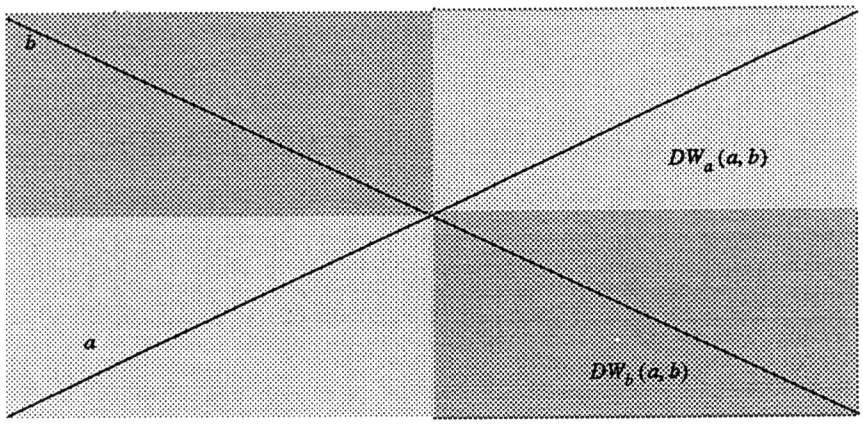

Figure 2: Doublewedges formed by angular bisectors at the intersection of a pair of lines a and b.

reporting) problem is solvable in $O(n^4 \log n)$ (resp. $O(n^4)$) space and $O(\log n\ (+i))$ query time.

Theorem 7.5 *A set S of n colored lines in the plane can be preprocessed into a data structure of size $O(n^4 \log n)$ (resp. $O(n^4)$) such that the i distinct colors of the lines intersected by a variable-radius query circle (or disk) can be counted (resp. reported) in $O(\log n\ (+i))$ time.* □

7.4 Generalized intersection searching on line segments with a query disk or circle

Here S is a set of n arbitrarily-oriented line segments in the plane and the query is a disk $D = D(q, r)$, with radius r and center q.

7.4.1 The case of a fixed-radius query disk

Instead of Lemma 5.5, we use the following alternative characterization of intersections between a disk and a line segment, which is more convenient.

For any line segment $s = \overline{ab} \in S$, we define the *drum of radius r around s*, denoted $Drum(r, s)$, as follows: Let $D_a(r, s)$ and $D_b(r, s)$ denote, respectively, the closed disks of radius r centered at a and at b respectively. Let $R(r, s)$ be the rectangular region of the plane defined by the four points at which $strip_s$ intersects $D_a(r, s)$ and $D_b(r, s)$. Then $Drum(r, s) = D_a(r, s) \cup D_b(r, s) \cup R(r, s)$. (See Figure 3 for an example.)

Lemma 7.5 *A disk D with radius r and center q intersects a segment s iff $q \in$ Drum(r, s).*

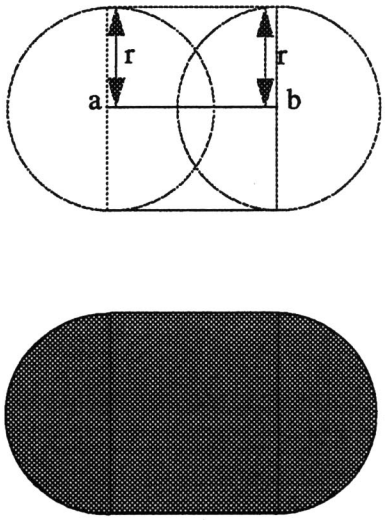

Figure 3: The drum of radius r around a segment $s = \overline{ab}$

Proof Let $\bar{\rho}(p, s)$ denote the Euclidean distance between a point p and a line segment $s = \overline{ab}$. That is, if the normal projection $\hat{p}$ of p on the line supporting s falls on s, then $\bar{\rho}(p, s) = d(p, \hat{p})$; otherwise, $\bar{\rho}(p, s) = \min\{d(p, a), d(p, b)\}$. (Recall that $d(\cdot, \cdot)$ is the Euclidean distance function for points.) Now, D intersects s iff they have a common point, i.e., iff $\bar{\rho}(q, s) \leq r$. By construction, $Drum(r, s)$ is the locus of all points at distance at most r from s. The lemma follows. □

We construct the arrangement $\mathcal{A}$ of the drums and preprocess it for fast planar point location using a simple extension of the algorithm in [ST86]. As before, for each face f we determine the drums whose intersection is f and store with f a count, $count_f$, or a list, L_f, of the distinct colors of these drums. The total space is $O(n^2)$ for the counting problem and $O(n^3)$ for the reporting problem. We can reduce the latter to $O(n^2)$ by applying persistence techniques as in Section 7.2.1. To answer a query, we locate the face containing q and output the associated information.

Theorem 7.6 *A set S of n arbitrarily-oriented colored line segments in the plane can be preprocessed into a data structure of size $O(n^2)$ such that the i distinct colors of the segments that are intersected by a fixed-radius query disk D can be counted (resp. reported) in time $O(\log n\ (+i))$* □

7.4.2 The case of a fixed-radius query circle

In fact, we can extend Lemma 7.5 to characterize intersections between a circle C with radius r and center q, and a line segment $s = \overline{ab}$, as follows: Let $int(R)$ denote the interior of any region R in the plane. We define the *cut drum of radius r around s*, denoted by $Drum'(r, s)$, as the region $Drum(r, s) - int(D_a(r, s) \cap D_b(r, s))$.

Lemma 7.6 *A circle C with radius r and center q intersects a line segment $s = \overline{ab}$ iff* $q \in \text{Drum}'(r, s)$.

Proof Let D be the closed disk bounded by C. Then C intersects s iff D intersects s and a and b are not both in $int(D)$, i.e., iff D intersects s and a and b are not both at distance less than r from q. The region $int(D_a(r,s) \cap D_b(r,s))$ is the locus of all points at distance less than r from q. By Lemma 7.5, D intersects s iff $q \in Drum(r,s)$. It follows that C intersects s iff q is in $Drum(r,s) - int(D_a(r,s) \cap D_b(r,s)) = Drum'(r,s)$. □

We can now use a structure similar to that of Theorem 7.6. We omit the details and conclude directly:

Theorem 7.7 *A set S of n arbitrarily-oriented colored line segments in the plane can be preprocessed into a data structure of size $O(n^2)$ such that the i distinct colors of the segments that are intersected by a fixed-radius query circle C can be counted (resp. reported) in time* $O(\log n\ (+i))$. □

7.4.3 Querying with a variable-radius disk

Let s_i and s_j $(i \neq j)$ be two segments in S. Let B_{ij} be the locus of all points p such that $\bar{\rho}(p, s_i) = \bar{\rho}(p, s_j)$. As shown in [LD81], B_{ij} is in general made up of five pieces: one line segment, two rays, and two parabolic arcs. Let $\mathcal{A}$ be the arrangement of the B_{ij}, $1 \leq i < j \leq n$. We preprocess $\mathcal{A}$ for planar point location using a simple extension of the Sarnak–Tarjan algorithm [ST86]. Since the pieces of B_{ij} are of constant degree, any two can intersect only $O(1)$ times, which implies that $\mathcal{A}$ has $O(n^4)$ vertices, faces, and edges (line segments, rays, or subarcs of parabolic arcs). As in Lemma 7.2, we can prove that for any face f and for different points $p \in f$, the ordering of the s_i's by nondecreasing $\bar{\rho}(p, s_i)$ is invariant. Call this ordering $\mathcal{E}(f)$.

Proceeding as in Section 7.2.3, we can now interpret the problem at hand as a generalized 1-dimensional range searching problem on $1, 2, \ldots, n$, with $(-\infty, k]$ being the query. Together with the persistence-based approach for space reduction, this immediately gives us the following:

Theorem 7.8 *A set S of n arbitrarily-oriented colored line segments in the plane can be preprocessed into a data structure of size $O(n^4 \log n)$ (resp. $O(n^4)$) such that the i distinct colors of the segments intersected by a variable radius query disk D, can be reported in time* $O(\log n\ (+i))$. □

7.5 Generalized intersection searching on disks with a query disk

Here $S = \{D_1, D_2, \ldots, D_n\}$ is a collection of closed disks in the plane, of possibly different radii. Let Q be the closed query disk. Let C_i be the center of D_i and r_i its radius, $1 \leq i \leq n$. Let C_Q and r_Q denote the corresponding quantities for Q.

Recall the intersection condition for two disks Q and D_i, namely: Q intersects D_i iff $d(C_Q, C_i) \leq r_Q + r_i$.

7.5.1 The fixed-radius case

Let D_i' be the closed disk of radius $r_Q + r_i$ centered at C_i. From the above-mentioned intersection condition it is clear that our problem is to count or report the distinct colors of the disks D_i' that contain C_Q.

We construct the arrangement $\mathcal{A}$ of the disks D_i', process it for fast planar point location, and store with each face f either $count_f$ or L_f (whose definitions are analogous to those in previous sections). In the reporting case, we also apply persistence techniques, as in the previous sections, to reduce the total space to $O(n^2)$. Queries are answered by locating the face containing C_Q and reading off the associated information.

Theorem 7.9 *A set S of n disks in the plane (of possibly different radii) can be preprocessed into a data structure of size $O(n^2)$ such that the i distinct colors of the disks that are intersected by a fixed-radius query disk can be counted or reported in $O(\log n\ (+i))$ time.* □

7.5.2 The variable-radius case

The approach of Section 7.5.1 cannot be applied if r_Q is part of the query. In this situation, it is advantageous to interpret the above-mentioned intersection condition for Q and D_i as: Q intersects D_i iff $d(C_Q, C_i) - r_i \leq r_Q$.

For any D_i and D_j $(i \neq j)$, let L_{ij} be the locus of the points p such that $d(p, C_i) - r_i = d(p, C_j) - r_j$. It is well-known [Sha85] that L_{ij} is a ray or a hyperbolic arc. We construct the arrangement $\mathcal{A}$ of the L_{ij}, $1 \leq i < j \leq n$. Since the L_{ij}'s are of constant degree, any two can intersect only $O(1)$ times. Thus, $\mathcal{A}$ consists of $O(n^4)$ vertices, faces, and edges (rays, line segments, or subarcs of hyperbolic arcs).

Using a proof similar to Lemma 7.2 it is easy to show that within each face f of $\mathcal{A}$, for different points $p \in f$ the ordering of the C_i's by nondecreasing $d(p, C_i) - r_i$ is invariant. Let $\mathcal{E}(f)$ denote this ordering for f. As in Section 7.2.3, we can solve our generalized problem for disks by answering a generalized 1-dimensional range searching query on $1, 2, \ldots, n$ using a

query interval of the form $(-\infty, k]$. Together with the persistence-based approach for space reduction, this immediately gives us the following:

Theorem 7.10 *A set S of n disks in the plane, of possibly different radii, can be preprocessed into a data structure of size $O(n^4 \log n)$ (resp. $O(n^4)$) such that the i distinct colors of the disks that are intersected by a variable-radius query disk can be counted (resp. reported) in $O(\log n\ (+i))$ time.* □

8 Conclusions and further research

We have investigated the problem of intersection searching involving curved (specifically, circular and circle-like objects) and have presented efficient solutions for three broad classes of problems, namely (i) for linear input objects and curved query objects, (ii) for curved input objects and curved query objects, and (iii) for a generalized version of the previous two classes.

We close by mentioning two interesting directions for further research: First, can our solutions be made dynamic, so that in addition to answering queries we can also accommodate efficiently insertions and deletions of objects in the input set? Second, can efficient solutions be designed for other types of curved objects?

References

[Aga89] P.K. Agarwal. Ray shooting and other applications of spanning trees with low stabbing number. In *Proceedings of the 5th Annual Symposium on Computational Geometry*, pages 315–325, 1989.

[AHL90] A. Aggarwal, M. Hansen, and T. Leighton. Solving query-retrieval problems by compacting Voronoi diagrams. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, pages 331–340, 1990.

[AS91] P.K. Agarwal and M. Sharir. Counting circular arc intersections. In *Proceedings of the 6th Annual Symposium on Computational Geometry*, pages 10–20, 1991.

[AvK93] P.K. Agarwal and M. van Kreveld. Connected component and simple polygon intersection searching. In *Proceedings of the 1993 Workshop on Algorithms and Data Structures*, August 1993. To appear.

[AvKO] P.K. Agarwal, M. van Kreveld, and M. Overmars. Intersection queries in curved objects. Manuscript. Preliminary version in Proc. 1991 Symp. on Computational Geometry, pp. 41–50.

[BM79] J.L. Bentley and H. A. Maurer. A note on Euclidean near neighbor searching in the plane. *Information Processing Letters*, 8:133–136, 1979.

[CF91] B.M. Chazelle and J. Friedman. Point location among hyperplanes and unidirectional ray-shooting. Technical Report TR–333–91, Dept. of Computer Science, Princeton University, 1991.

[CG86] B.M. Chazelle and L.J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1:133–162, 1986.

[Cha86] B.M. Chazelle. Filtering search: a new approach to query-answering. *SIAM Journal on Computing*, 15:703–724, 1986.

[Cha91] B.M. Chazelle. An optimal convex hull algorithm and new results on cuttings. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science*, pages 29–38, 1991.

[CJ92] S.W. Cheng and R. Janardan. Algorithms for ray-shooting and intersection searching. *Journal of Algorithms*, 13:670–692, 1992.

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* MIT Press and McGraw-Hill, 1990.

[CSW90] B.M. Chazelle, M. Sharir, and E. Welzl. Quasi-optimal upper bounds for simplex range searching and new zone theorems. In *Proceedings of the 6th Annual Symposium on Computational Geometry*, pages 23–33, 1990.

[CW89] B. Chazelle and E. Welzl. Quasi-optimal range searching in spaces of finite VC-dimension. *Discrete and Computational Geometry*, 4:467–489, 1989.

[dB92] M. de Berg. *Efficient algorithms for ray-shooting and hidden surface removal.* PhD thesis, Department of Computer Science, University of Utrecht, Utrecht, the Netherlands, 1992.

[DE87] D.P. Dobkin and H. Edelsbrunner. Space searching for intersecting objects. *Journal of Algorithms*, 8:348–361, 1987.

[DK83] D.P. Dobkin and D.G. Kirkpatrick. Fast detection of polyhedral intersection. *Theoretical Computer Science*, 27:241–253, 1983.

[DSST89] J.R. Driscoll, N. Sarnak, D.D. Sleator, and R.E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38:86–124, 1989.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer–Verlag, 1987.

[GJS93] P. Gupta, R. Janardan, and M. Smid. Further results on generalized intersection searching problems: counting, reporting, and dynamization. In *Proceedings of the 1993 Workshop on Algorithms and Data Structures*, August 1993. To appear. Available as Tech. Rept. TR–92–72, Dept. of Computer Science, University of Minnesota, Minneapolis, MN. Also submitted.

[JL93] R. Janardan and M. Lopez. Generalized intersection searching problems. *International Journal of Computational Geometry & Applications*, 3:39–69, 1993.

[LD81] D.T. Lee and R.L. Drysdale. Generalization of Voronoi diagrams in the plane. *SIAM Journal on Computing*, 10:73–87, 1981.

[LS91] H-P. Lenhof and M. Smid. An optimal construction method for generalized convex layers. In *Proceedings of the 2nd International Symposium on Algorithms*, pages 349–361, Taiwan, December 1991.

[Mat91a] J. Matoušek. Cutting hyperplane arrangements. *Discrete & Computational Geometry*, 6:385–406, 1991.

[Mat91b] J. Matoušek. Reporting points in halfspaces. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science*, pages 207–215, 1991.

[Mat92] J. Matoušek. Efficient partition trees. *Discrete & Computational Geometry*, 8:315–334, 1992.

[McC85] E.M. McCreight. Priority search trees. *SIAM Journal on Computing*, 14:257–276, 1985.

[Pel92] M. Pellegrini. A new algorithm for counting circular arc intersections. Technical Report TR–92–010, International Computer Science Institute, Berkeley, CA, 1992.

[Sha85] M. Sharir. Intersection and closest-pair problems for a set of planar disks. *SIAM Journal on Computing*, 14:448–468, 1985.

[Sha91] M. Sharir. The k-set problem for arrangement of curves and surfaces. *Discrete and Computational Geometry*, 6:593–613, 1991.

[ST86] N. Sarnak and R.E. Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29:669–679, 1986.

[vK92] M. van Kreveld. *New results on data structures in computational geometry.* PhD thesis, Department of Computer Science, University of Utrecht, Utrecht, the Netherlands, 1992.

[vKOA90] M. van Kreveld, M. Overmars, and P. K. Agarwal. Intersection queries in sets of disks. In *Proceedings of the 1990 Scandinavian Workshop on Algorithm Theory, LNCS 447*, pages 393–403. Springer–Verlag, 1990.