Intersection queries in curved objects

P.K. Agarwal, M. van Kreveld, M. Overmars

RUU-CS-91-12
May 1991

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

Intersection queries in curved objects

P.K. Agarwal, M. van Kreveld, M. Overmars

Technical Report RUU-CS-91-12
May 1991

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

Intersection Queries in Curved Objects*

Pankaj K. Agarwal[1] Marc van Kreveld[2] Mark Overmars[2]

[1] Computer Science Department, Duke University,
Durham, NC 27706, USA.

[2] Department of Computer Science, Utrecht University,
P.O. Box 80.089, 3508 TB Utrecht, the Netherlands.

Abstract

A number of problems of the following type are studied: Given a set of n arcs (disks, circles, circular arcs, Jordan arcs) in the plane, preprocess it into a data structure, such that for a query line (or segment) one can quickly (i) report all arcs intersecting it, or (ii) count the number of arcs intersecting it. We also study the ray shooting problem in disjoint Jordan arcs, and circular arcs. Most of the data structures presented here use near to linear space and have query time near to $O(\sqrt{n} + K)$ or near to $O(n^{2/3} + K)$, where K is the size of the output.

1 Introduction

An interesting field within computational geometry is *range searching*, where one stores a given set of objects in a data structure so that certain types of queries can be answered efficiently. Queries generally ask for all objects of the set that are intersected by some query range. For example, the objects could be sets of points in the plane, and query ranges could be axis parallel rectangles. During the last few years, research has developed rapidly in searching with non-isothetic figures, e.g. with triangles etc. [see [1, 3, 15, 23, 26]]. The lower bound result of Chazelle shows that searching with non-isothetic objects is much harder than with isothetic objects [8].

*The research of the first author was supported by DIMACS (Center for Discrete Mathematics and Theoretical Computer Science), a National Science Foundation Science and Technology Center – NSF-STC88-09648. The research of the second and third authors was supported by the ESPRIT Basic Research Action No. 3075 (project ALCOM). The research of the third author was also supported by the Dutch Organization for Scientific Research (N.W.O.).

ISSN: 0924–3275

One direction for extending this branch of research is shifting to higher dimensions, see [14, 15, 26], whereas another direction is storing a set of objects more general than points [1, 18, 33]. But in all these papers the underlying objects were polygonal and the structures rely on the linearity of the objects. Surprisingly, very little attention has been paid to non-linear objects. Some of the known results in this direction are stabbing a set of disks by points in the plane [37], and line or segment intersection searching among disks [28].

In this paper, we study the problem of efficiently storing a set of curved objects for various kinds of intersection queries. For instance, we discuss the problem of reporting (or counting the number of) the curved objects intersected by a query half-plane, line, or segment. We also study the ray shooting problem in arrangements of arcs, that is: "Given a set of arcs, preprocess it into a data structure, such that for a (directed) query ray ρ, one can quickly determine the first arc intersected by ρ as we follow ρ from the starting point." Ray shooting problems have applications in hidden surface removal [6, 34], contour tracing, etc.

The methods used here apply a variety of known techniques, including spanning trees with low stabbing number [1, 3, 15], partition trees [15, 23, 26], k-sets and $(\leq k)$-sets [20, 37], fractional cascading [10, 11], persistent data structures [19, 36], and Davenport-Schinzel sequences [4, 25]. Furthermore, new ideas and techniques are exploited to deal with curved objects. In some cases, our results on curved objects match the best known results of their linear object counterparts. An overview of our main results is given in Table 1.

Before we start any discussion on arcs, we need to make some assumptions on arcs and also on our model of computation. We assume that arcs are of simple shape, that is, an arc is connected, and an arc intersects another arc or a line in a constant number of points. As to our model of computation, we assume that all basic operations involving arcs, namely, computing intersections between a pair of arcs, intersections between an arc and a segment or a line, common tangent to a pair of arcs, and points of vertical tangencies, can be computed in $O(1)$ time.

The rest of this paper is organized as follows. In Section 2, we review some known results including spanning trees with low stabbing number, partition trees and $(\leq k)$-sets. In Section 3, we give geometric transforms and observations, which will help to efficiently solve some of the problems studied in this paper. Section 4 discusses the problem of preprocessing a set of arcs in the plane, such that the arcs intersecting a query half-plane can be retrieved efficiently. Section 5 deals with the problem of reporting arcs intersecting a query line and Section 6 considers the same problem except that the query objects are segments instead of lines. In Section 7, the ray shooting problem in sets of arcs is discussed. We conclude with some final remarks in Section 8.

Reporting intersections

Range	Objects	Space	Query time
Lines	Arcs that intersect at most s times	$O(\lambda_{s+2}(n)\log^2 n)$	$O(\sqrt{n}\log^2 n + K)$
Lines	Circular arcs	$O(n)$	$O(n^{2/3+\epsilon} + K)$
Lines	Arcs in m connected components	$O(n\log n)$	$O(\sqrt{m}\log n + K\log(n/K))$
Segments	Disks	$O(n\log^2 n)$	$O(\sqrt{n}\log^2 n + K)$
Segments	Circles	$O(n\log^2 n)$	$O(n^{2/3}\log^2 n + K)$
Segments	Circular arcs	$O(n\log n)$	$O(n^{2/3+\epsilon} + K)$
Segments	Disjoint arcs	$O(n\log n)$	$O(\sqrt{n}\log n + K\log n)$

Counting intersections

Range	Objects	Space	Query time
Lines	Circles	$O(n\log n)$	$O(n^{2/3}\log^2 n)$
Lines	Circular arcs	$O(n)$	$O(n^{2/3+\epsilon})$
Segments	Disks	$O(n\log n)$	$O(n^{2/3}\log^2 n)$
Segments	Circular arcs	$O(n)$	$O(n^{6/7+\epsilon})$

Ray shooting

Objects	Space	Query time
Disjoint arcs	$O(n\log n)$	$O(\sqrt{n}\log n)$
Circular arcs	$O(n\cdot 2^{\alpha(n)})$	$O(n^{2/3+\epsilon})$

Table 1: Overview of the results; ϵ is an arbitrarily small constant, K is the output size, and $\lambda_{s+2}(n)$ is the maximum length of an $(n, s+2)$-Davenport-Schinzel sequence (which is extremely close to linear in n for any constant s).

2 Geometric Preliminaries

In this section we describe some known data structures and techniques. We begin with a brief discussion on partition trees and "spanning trees with low stabbing number". Then we discuss a technique of space reduction, which we use several times. Finally, we direct our attention to k-sets and related concepts.

2.1 Partition trees

During the last few years, a number of papers have appeared on *simplex range searching*: "Given a set S of n points in $\mathbf{R}^d$, preprocess it into a data structure, so that all K points that lie in a query simplex can be reported efficiently." A query time of $O(\log n + K)$ can be achieved using $O(n^{d+\epsilon})$ space, for any $\epsilon > 0$ [14, 17]. But generally, one would like a data structure that uses roughly linear space, and answers queries as efficiently as possible. In the plane, a number of data structures have been proposed that use linear or close-to-linear space and answer a simplex range query in sublinear time [14, 15, 23, 26, 32, 39]. Most of these data structures are based on *partition trees.*

Generally, a partition tree $\mathcal{T}$ on a set S of n points in $\mathbf{R}^d$ is a tree, of which every node is associated with a subset of points. At each node v, associated with a subset S_v, we decompose the space into r convex regions (often r is some fixed constant), each of constant complexity, such that every hyperplane intersects only a fraction of the regions, and they contain at most a fraction of points of S_v. We create a child w for each region τ and set S_w to be $S_v \cap \tau$. We say that a hyperplane h *intersects* a node v if h intersects the region corresponding to v. The highest nodes of $\mathcal{T}$ that do not intersect h are called *canonical nodes* of $\mathcal{T}$ with respect to h. Let $\sigma_{\mathcal{T}}(n)$ denote the maximum number of canonical nodes of $\mathcal{T}$ with respect to any hyperplane.[1] Then a simplex range query can be answered in time $O(\sigma_{\mathcal{T}}(n) + K)$.

Using the theory of random sampling, Haussler and Welzl [26] proved that if we randomly choose a subset of r points, construct the arrangement of $\binom{r}{d}$ hyperplanes defined by the d-tuples of these points, and triangulate every cell of their arrangement, then a hyperplane intersects only $O(r^{d(d-1)})$ simplices of the triangulated arrangement and, with high probability, the total number of points contained in these simplices is at most $\frac{cn}{r}\log r$, where c is a constant depending only on d. Based on this partitioning scheme, they constructed a partition tree $\mathcal{T}$ such that $\sigma_{\mathcal{T}}(n) = O(n^{\beta})$, where $\beta = \frac{d(d-1)}{d(d-1)+1} + \epsilon$, for arbitrarily small positive ϵ.

Theorem 2.1 (Haussler and Welzl [26]) *Given a set S of n points in d-dimensional space. There exists a tree $\mathcal{T}$ that stores S in $O(n)$ space, such that for any query simplex X, the set $X \cap S$ is represented by $O(n^{\beta})$ nodes of $\mathcal{T}$, where $\beta = \frac{d(d-1)}{d(d-1)+1} + \epsilon$ for arbitrarily small positive ϵ.*

An important feature of the partition trees is that they support multi-level structures without affecting the asymptotic query time, that is, one can store a secondary structure at each node which may again be a partition tree, or an entirely different structure. This was observed by Dobkin and Edelsbrunner [18] (see also [24]). They used multi-level partition trees to solve a variety of problems. For example, they showed how to store a

[1] The notion of canonical nodes can be generalized to a simplex or to any other object τ. If τ is bounded by k hyperplanes, the maximum number of canonical nodes with respect to τ is $\leq k\sigma_{\mathcal{T}}(n)$.

set S of segments in the plane for intersection queries with segments, with complexity bounds equal to those of simplex range queries. To this end, they observe that two segments s_1 and s_2 intersect if and only if the line supporting s_1 intersects s_2, and the line supporting s_2 intersects s_1. Then they define a two-level structure. The first level stores the set S of segments and supports line intersection queries. The second level stores the lines supporting the segments of S and supports segment intersection queries. For a query segment s, the search starts in the first level with the line containing s, and selects all segments of S that intersect this line in a collection of canonical nodes. Then the search continues with the query segment s in the second-level structures. Now the lines supporting the segments of S which intersect s are selected. By the above observation, the segments of S that intersect s are found after the two selections.

In view of the above theorem and the observations made by Dobkin and Edelsbrunner [18], we have

Corollary 2.2 *Let S be a set of n points in d-dimensional space, and let $\mathcal{T}$ be a partition tree on S. Suppose that $\mathcal{T}'$ is an associated structure of $\mathcal{T}$, which uses $M(m)$ space and $Q(m)$ query time, for m objects. Then $\mathcal{T}$ uses $O(n + M(n)\log n)$ space, and queries take $O(\max\{n^\beta, Q(n)\})$ time (with β as above).*

2.2 Spanning trees with low stabbing number

The *stabbing number* of a tree embedded in $\mathbf{R}^d$ is the maximum number of its edges that can be intersected by a hyperplane. Chazelle and Welzl [15] have shown that, given a set of n points, there exists a spanning path Π with stabbing number $O(n^{1-1/d})$. One can construct a minimum height binary tree on Π, of which every node is associated with a contiguous portion of Π (see [1, 15]). We will call this binary tree an *s-tree*. We now define the canonical nodes with respect to a hyperplane h to be the highest nodes of the tree whose associated subpaths do not cross h. It follows that, for any hyperplane or simplex, the number of canonical nodes is $O(n^{1-1/d}\log n)$ in an s-tree and, for $d = 2, 3$, they can be computed in time $O(n^{1-1/d}\log^{d-1} n)$ time (see [1, 15]). Thus, a simplex range query (for $d = 2, 3$) can be answered in time $O(n^{1-1/d}\log^{d-1} n + K)$.

Theorem 2.3 (Chazelle and Welzl [15]) *Given a set S of n points in $\mathbf{R}^d$ ($d = 2, 3$), there exists a tree $\mathcal{T}$ that stores S in $O(n\log n)$ space, such that for any query simplex X, there are $O(n^{1-1/d}\log n)$ canonical nodes with respect to X, and they can be computed in $O(n^{1-1/d}\log^{d-1} n)$ time.*

Although such trees were originally used for simplex range searching problems, they have found applications in several other problems; see Agarwal [1] for a list of such problems. Disregarding logarithmic factors, his structures use linear space and answer queries in $O(\sqrt{n})$ time.

A disadvantage of s-trees is that, unlike partition trees, they do not support multi-level structures without affecting the query time. In particular, for s-trees the following weaker result exists:

Corollary 2.4 *Let S be a set of n points in $\mathbf{R}^d$ ($d = 2, 3$), and let $\mathcal{T}$ be an s-tree on S. Suppose that we store at each node of $\mathcal{T}$ a secondary structure $\mathcal{T}'$ that uses $M(m)$ space and $Q(m)$ query time, for m objects. Then $\mathcal{T}$ uses $O(n + M(n) \log n)$ space, and answers queries in $O(n^{1-1/d} \log n \cdot Q(n^{1/d}) + Q(n))$ time.*

2.3 Reducing space requirements

Dobkin and Edelsbrunner also presented a technique to reduce the space used by the associated structures [18]. Instead of storing associated structures at all nodes of the main tree, one only stores associated structures with nodes v that are on some level $i \cdot \lfloor \frac{\epsilon}{2} \log n \rfloor$, for all integers i between 1 and $2/\epsilon$. Since one point appears in the associated structures of all nodes along its search path, it follows that each point is stored only $2/\epsilon$ times instead of $O(\log n)$ times, for a balanced main tree.

But how do we query in this tree? We begin — as usual — to select the set of canonical nodes of the main tree. Let v be a canonical node on a level l. If $l = i \cdot \lfloor \frac{\epsilon}{2} \log n \rfloor$ for some integer i, then we simply query in the associated structure of v. Otherwise, we descend in the subtree rooted at v to the level $l' = i \cdot \lfloor \frac{\epsilon}{2} \log n \rfloor$ for some integer i, such that $i \cdot \lfloor \frac{\epsilon}{2} \log n \rfloor > l > (i-1) \cdot \lfloor \frac{\epsilon}{2} \log n \rfloor$. This is the first level below v that stores associated structures. Here we query in the assocated structures of all $O(n^{\epsilon/2})$ nodes below v on the level l'.

The next two corollaries are variants of the above, but here the space reduction technique is applied. In the first corollary, the change in query time disappears in the ϵ.

Corollary 2.5 *Let S be a set of n points in $\mathbf{R}^d$, and let $\mathcal{T}$ be a partition tree on S. Suppose that $\mathcal{T}'$ is an associated structure of $\mathcal{T}$, which uses $M(m)$ space and $Q(m)$ query time, for m objects, and suppose that $\mathcal{T}'$ is stored at a constant number of levels. Then $\mathcal{T}$ uses $O(n + M(n))$ space, and queries take $O(\max\{n^\beta, Q(n)\})$ time (with $\beta = \frac{d(d-1)}{d(d-1)+1} + \epsilon$, and any $\epsilon > 0$).*

Corollary 2.6 *Let S be a set of n points in $\mathbf{R}^d$ ($d = 2, 3$), and let $\mathcal{T}$ be a s-tree on S. Suppose at a constant (dependent on ϵ) number of levels of $\mathcal{T}$ we store a secondary structure $\mathcal{T}'$ that uses $M(m)$ space and $Q(m)$ query time, for m objects, then $\mathcal{T}$ uses $O(n + M(n))$ space, and answers queries in $O(n^{1-1/d+\epsilon} \cdot Q(n^{1/d}) + Q(n))$ time, for any $\epsilon > 0$.*

2.4 k-Sets and related concepts

An important topic in combinatorial geometry is the study of k-sets because of its applications in various problems [13, 16]. For a set S of n points in the plane, a set $S' \subseteq S$ of size k is a k-set if there exists a half-plane h such that $h \cap S = S'$. A j-set of S, for $j \leq k$, is called an $(\leq k)$-set of S. It is well-known that the maximum number of $(\leq k)$-sets for any set of points in the plane is $O(nk)$, and this bound is sharp in the worst case (see e.g. [20]). If we dualize the set S of points to a set S^* of lines, then $S_k \subseteq S$ is a k-set if and only if there is a point in the dual plane that lies above (resp. below) all the lines of S_k^* and below (resp. above) all the lines of $S^* - S_k^*$. In view of this dual correspondence, the notion of k-sets can be generalized to plane separating Jordan curves as follows: Let Γ be such a set of n Jordan curves in the plane. For each curve, one of the sides is called the interior, and the other side is called the exterior. A k-set of Γ is a subset Γ' of Γ of k curves, such that there exists a point p that lies in the interior of all curves in Γ', and in the exterior of all curves in $\Gamma - \Gamma'$. We define $(\leq k)$-sets in an analogous way.

Sharir [37] showed that if any two curves of Γ intersect at most twice, then the maximum number of $(\leq k)$-sets is $\Theta(nk)$. Furthermore, he proved that for a set of x-monotone curves, the maximum number of $(\leq k)$-sets is $\Theta(k^2\lambda_s(n/k))$, where s is the maximum number of pairwise intersections and $\lambda_s(n)$ is the maximum length of a Davenport-Schinzel sequence with n symbols and of order s. For any constant s, $\lambda_s(n)$ is linear or almost linear. The best known bounds on $\lambda_s(n)$ are as follows:

$$\begin{aligned}
\lambda_1(n) &= n, \quad \lambda_2(n) = 2n - 1 \quad \text{(trivial)} \\
\lambda_3(n) &= \Theta(n\alpha(n)) \quad [25] \\
\lambda_4(n) &= \Theta(n2^{\alpha(n)}) \quad [4] \\
\lambda_{2s+2}(n) &= n \cdot 2^{\Theta(\alpha^s(n))} \quad [4] \\
\lambda_{2s+3}(n) &= n \cdot 2^{O(\alpha^s(n)\log\alpha(n))} \quad [4]
\end{aligned}$$

As we will see later, $(\leq k)$-sets in curves play an important role in the analysis of some of the results in this paper.

3 Geometric Transforms

In this section we describe various geometric results involving curves, lines and segments. We first introduce some geometric transforms, which map circles, lines and points in the plane to points, planes and wedges in $\mathbf{R}^3$. These transforms allow us to reduce various intersection searching problems to simplex range searching problems. Secondly, we prove necessary and sufficient conditions for intersections between circular arcs (or circles, disks) and segments (or lines), which help to devise multi-level data structures to solve intersection searching problems.

The first two transforms that we describe map circles in $\mathbf{R}^2$ to planes in $\mathbf{R}^3$, and points in $\mathbf{R}^2$ to points in $\mathbf{R}^3$, respectively. Let C be a circle in the plane, with center (a, b) and radius r. Define $\varphi_1(C) \rightarrow z = a(2x - a) + b(2y - b) + r^2$. Let $p = (c, d)$ be a point in the plane. Define $\psi_1(p) \rightarrow (c, d, c^2 + d^2)$.

The next two transforms are merely an adaption of the first two; here circles are mapped to points in $\mathbf{R}^3$, and points are mapped to planes in $\mathbf{R}^3$. Let C and p be as above. Define $\varphi_2(C) \rightarrow (a, b, a^2 + b^2 - r^2)$, and $\psi_2(p) \rightarrow z = 2cx + 2dy - c^2 - d^2$.

Lemma 3.1 *For a circle C and a point p, the following three statements are equivalent:*

(i) p lies in the interior of C,

(ii) $\psi_1(p)$ lies below $\varphi_1(C)$,

(iii) $\varphi_2(C)$ lies below $\psi_2(p)$.

The next two transforms express the intersection between lines and circles as the containment of points in wedges in $\mathbf{R}^3$. Let C be as above. Define $\varphi_3(C) \rightarrow (a, b, r)$. Let ℓ be the non-vertical line $y = mx + c$. Define $\chi_3(\ell) \rightarrow z \geq |(mx - y + c)|/\sqrt{m^2 + 1}$. If ℓ is the vertical line $x = c$, then define $\chi_3(\ell) \rightarrow z \geq |x - c|$.

Lemma 3.2 *ℓ intersects C if and only if the point $\varphi_3(C)$ lies in the wedge $\chi_3(\ell)$.*

Finally, we also use the standard "duality" transforms that map a point $p = (a, b)$ to a line $p^* : y = ax + b$ and a line $\ell : y = cx + d$ to a point $\ell^* = (-c, d)$.

The above transformations are useful only for circles and disks with respect to lines and points. When we deal with circular arcs or segments, we have to use some additional techniques. Let s be a line segment in the plane. Let p_s and q_s denote the endpoints of s, and ℓ_s the line containing s. Furthermore, let ℓ_p (resp. ℓ_q) be the line passing through p_s (resp. q_s) and perpendicular to s, and let $strip_s$ be the strip bounded by ℓ_p and ℓ_q. Finally, let $center(D)$ denote the center of a disk or circle D.

Lemma 3.3 *A segment s intersects a disk D if and only if at least one of the following two conditions is satisfied (see Figure 1 (a)).*

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

(ii) $center(D)$ lies in $strip_s$ and ℓ_s intersects D.

Lemma 3.4 *A segment s intersects a circle C if and only if*

(i) exactly one of the endpoints of s lies inside C, or

(ii) both endpoints of s lie outside C, $center(C)$ lies in $strip_s$, and ℓ_s intersects C.

Figure 1: How disks, circles and circular arcs can lie with respect to lines and segments.

Next, we describe necessary and sufficient conditions for a circular arc to intersect a line or a segment. For a circular arc γ, let $circ(\gamma)$ denote the circle containing γ. Let γ' denote the closure of $circ(\gamma) - \gamma$, i.e., the part of $circ(\gamma)$ that is not in γ, but including the endpoints.

Lemma 3.5 *A line ℓ intersects a circular arc γ if and only if one of the following two conditions is satisfied (see Figure 1 (b)).*

(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 (i.e., γ' does not intersect ℓ), and ℓ intersects $circ(\gamma)$.

If the first condition of the above lemma is satisfied, then ℓ intersects γ at one point, whereas in the second case it intersects γ at two points. Finally, we deal with the intersections of segments and circular arcs.

Lemma 3.6 *A segment s intersects a circular arc γ if and only if one of the following conditions is satisfied (see Figure 1 (c)). Let h_γ denote the half-plane containing the arc γ and bounded by the line passing through the endpoints of γ.*

(i) ℓ_s separates the endpoints of γ, and one of the endpoints of s, say p_s lies in h_γ and in the exterior of $circ(\gamma)$, and

(a) q_s lies in the interior of $circ(\gamma)$, or

(b) $q_s \notin h_\gamma$ and in the exterior of $circ(\gamma)$,

or

(ii) γ' does not intersect ℓ_s and s intersects $circ(\gamma)$.

4 Half-plane Intersection Queries

In this section we consider the *half-plane query* problem, defined as follows: "Given a set Γ of n arcs in the plane, preprocess it so that, for a query half-plane h, the arcs of Γ intersecting h can be reported efficiently." It turns out that studying this problem will be useful for the problems to be treated later on, but it is interesting in its own right as well. We present two different methods, the first is based on the transformations of the previous section, and the second method relies on persistent data structures [19, 36], $(\leq k)$-sets in curves [37], and filtering search [7].

4.1 The transformation method

Let $\mathcal{C}$ be a set of n circles in the plane. We transform $\mathcal{C}$ into the set $\varphi_3(\mathcal{C})$ of points in $\mathbf{R}^3$ and preprocess it into a data structure of size $O(n \log n)$ for half-space range searching, using the algorithm of Aggarwal et al. [5]. Let $h : y \geq mx + c$ be the query half-plane. Following the same argument as in Lemma 3.2, we can show that a circle $C \in \mathcal{C}$ intersects h if and only if $\varphi_3(C)$ lies in the half-space $z \geq (mx - y + c)/\sqrt{m^2 + 1}$. Thus, using the above structure, all K circles intersecting h can be reported in time $O(\log n + K)$ (cf. [5]). Hence, we obtain

Lemma 4.1 *Given a set $\mathcal{C}$ of n circles in the plane, we can build a data structure of size $O(n \log n)$, so that all K circles intersecting a query half-plane can be reported in time $O(\log n + K)$.*

Remark. Using the counting version of Chazelle and Welzl's simplex range searching algorithm, the number of circles intersecting a query half-plane can be counted in time $O(n^{2/3} \log^2 n)$.

4.2 The similar lists method

In this subsection we present a method that works for arbitrary arcs, and which is almost as efficient as the above solution for circles. It makes use of persistent data structures [36], upper bounds on $(\leq k)$-sets [37], and filtering search [7]. We assume the query half-planes to be positive, thus, the region above a line. For negative half-planes we build a similar structure, so there is no loss of generality. Let Γ be a set of n Jordan arcs, of which each pair intersects in at most s points (s a constant).

The basic idea of our algorithm is as follows. Suppose the slope of the lines bounding the query half-planes is fixed, say horizontal, then the following straightforward algorithm reports all K arcs of Γ intersecting the query half-plane in time $O(K)$: Sort the arcs of Γ in the non-increasing order of their topmost point. If $\gamma \in \Gamma$ is the first arc whose topmost

point lies below the line bounding the query half-plane, then all arcs lying before γ in the sorted order intersect the query half-plane. Hence, a query can be answered by traversing an ordered list up to some position.

Define an *upper tangent* of two arcs γ_1 and γ_2 to be a (non-vertical) line ℓ that contains a point of γ_1 and a point of γ_2, and such that no point of γ_1 or γ_2 lies in ℓ^+. Recall that every pair of arcs has at most s intersection points (for some constant s). Consequently, a pair of arcs has at most $s+2$ upper tangents. Now let $d_1 \leq d_2 \leq \cdots \leq d_t$, $t = O(n^2)$, be the slopes of upper tangents to all pairs of arcs in Γ. For all slopes $d \in [d_{i-1} : d_i]$, the ordering of Γ, by their topmost points in direction d, remains the same. Moreover, the ordering of Γ for the interval $[d_i : d_{i+1}]$ can be obtained from that of $[d_{i-1} : d_i]$ by swapping two adjacent elements. Let $\mathcal{E} = (E_1, E_2, \ldots, E_t)$ be the sequence of orderings (also called lists), where E_i corresponds to the interval $[d_i : d_{i+1}]$. Using persistent data structures [19], the lists of $\mathcal{E}$ can be stored using $O(n+t) = O(n^2)$ space, such that a half-plane query can be answered in time $O(\log n + K)$. We show that the filtering search technique allows us to reduce the space complexity to $O(\lambda_{s+2}(n)\log n)$ without affecting the asymptotic query time.

Before describing the algorithm, we need some notation. If E_{i+1} is obtained from E_i by swapping the k^{th} and $(k+1)^{th}$ elements, we call this a *k-swap*, and denote it as $\varsigma(E_i) = k$. The total number of k-swaps in $\mathcal{E}$ is denoted by $|k\text{-swap}|$. A j-swap, for some $j \leq k$, is called an $(\leq k)$-*swap*, and $|(\leq k)\text{-swap}|$ denotes the total number of $(\leq k)$-swaps in $\mathcal{E}$.

Lemma 4.2 *The number of $(\leq k)$-swaps in $\mathcal{E}$ is $O(k\lambda_{s+2}(n))$, where $\lambda_{s+2}(n)$ is the maximum length of an $(n, s+2)$-Davenport-Schinzel sequence.*

Proof: For each arc $\gamma \in \Gamma$, let γ' be the Jordan curve obtained by adding to the endpoints of γ the vertical downward rays (the endpoints of γ are the leftmost and rightmost points of γ, by the assumptions in the introduction). Let Γ' be the set of resulting Jordan curves. An arc $\gamma \in \Gamma$ intersects a positive half-plane ℓ^+ if and only if the line ℓ intersects γ'. Thus, if we sort Γ' in non-increasing order of the 'topmost' points in a direction d, the ordering is the same as of Γ in that direction. As a result, we can think of $\mathcal{E}$ to be the sequence of lists containing the elements of Γ'. Let δ be the set of points dual (see Section 3 for definition) to the tangents of the curve γ' (see Figure 2), and let the portion of the dual plane lying below δ be its interior (this definition makes sense because δ is easily seen to be a plane separating x-monotone Jordan curve). Then a point p lies in the interior of δ if and only if the line p^* intersects γ'. Let Δ be the set of dual x-monotone Jordan curves. Since the arcs in Γ are x-monotone and intersect at most s times, there are at most $s+2$ upper tangents to a pair of curves in Γ', which implies that two curves in Δ intersect in at most $s+2$ points. Then, by the result of Sharir [37], $|(\leq k)\text{-sets}|$ for curves in Δ is $O(k^2\lambda_{s+2}(n/k)) = O(k\lambda_{s+2}(n))$.

Observe that if $\varsigma(E_i) = j$, the upper tangent ℓ of the j^{th} and $(j+1)^{th}$ curves of E_i intersects exactly $j-1$ other curves of Γ', therefore ℓ^* lies in the interior of $j-1$ arcs of

Δ. Hence, the number of j-swaps in $\mathcal{E}$ is the same as $|(j-1)\text{-set}|$ in Δ. Now the claim is immediate. □

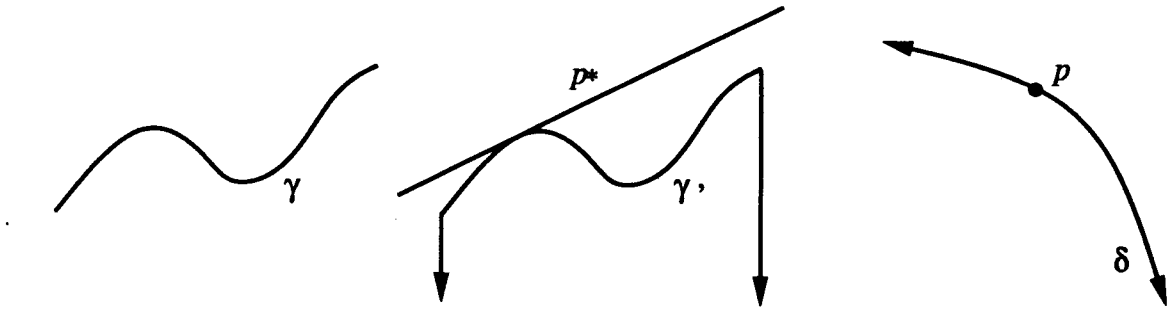

Figure 2: γ, γ' and δ.

From the sequence $\mathcal{E}$, we compute a new sequence $\mathcal{E}' = \{E'_1, E'_2, \ldots, E'_u\}$ of $O(\lambda_{s+2}(n)\log n)$ lists with the properties: (i) E'_{i+1} is obtained from E'_i by swapping two elements, and (ii) $\mathcal{E}'$ approximates $\mathcal{E}$ in the sense that for each $E_i \in \mathcal{E}$, there is an $E'_j \in \mathcal{E}'$ such that any first m items of E_i are within the first $3m$ items of E'_j. Moreover, the value of j can be computed in $O(\log n)$ time. Once having computed $\mathcal{E}'$, all K arcs of Γ intersecting a query half-plane can be reported in $O(\log n + K)$ time, as we will see below.

The sequence $\mathcal{E}'$ is computed in two steps. We first find a sequence $1 = b_1 < b_2 < \cdots < b_r$ with $b_r \geq n/3$, of distinguished positions, called *borders*, such that there are not too many b_i-swaps in $\mathcal{E}$ for any $i \leq r$. The following algorithm does the job, where c is the constant hidden in the big-Oh of Lemma 4.2.

Algorithm 1: COMPUTING b_i'S
$b_1 := 1$; $i := 1$; $b := 2$;
while $b \leq n-1$ **do**
  if $|b\text{-swap}| \leq 4c\lambda_{s+2}(n)$
    then $i := i+1$; $b_i := b$; $b := 2 \cdot b_i$;
    else $b := b+1$
  end if
end while

Lemma 4.3 *For every $1 \leq i \leq r$, $|b_i\text{-swap}| \leq 4c\lambda_{s+2}(n)$ (c as in the algorithm). Furthermore, for every $2 \leq i \leq r$, we have $2 \leq b_i/b_{i-1} < 3$, and $b_r \geq n/3$.*

Proof: The inequalities $|b_i\text{-swap}| \leq 4c\lambda_{s+2}(n)$ and $b_i/b_{i-1} \geq 2$ follow immediately from the above algorithm. Suppose that $b_i \geq 3b_{i-1}$, then for all $2b_{i-1} \leq b < b_i$, $|b\text{-swap}| > 4c\lambda_{s+2}(n)$, because all these positions were tested by the algorithm and none of

them were chosen. Thus

$$\begin{aligned} |(\leq b_i)\text{-swap}| &\geq \sum_{b=2b_{i-1}}^{b_i-1} |b\text{-swap}| \\ &> (b_i - 2b_{i-1})4c\lambda_{s+2}(n) \\ &\geq \left(\frac{b_i}{3} + 2b_{i-1} - 2b_{i-1}\right) 4c\lambda_{s+2}(n) \\ &> b_i c\lambda_{s+2}(n). \end{aligned}$$

This contradicts with Lemma 4.2. With the same argument, $b_r \geq n/3$ follows. □

Obviously, there are at most $\log_2 n + 2$ borders in the sequence. Next, we do the actual construction of the new sequence $\mathcal{E}'$. Intuitively, a new list is chosen for $\mathcal{E}'$ after every swap across one of the borders. Therefore, the number of lists is equal to the total number of swaps across borders, which is $O(\lambda_{s+2}(n)\log n)$ as required. The following algorithm describes the construction of the new sequence $\mathcal{E}'$.

Algorithm 2: COMPUTING $\mathcal{E}'$
$E'_1 := E_1$; $j := 1$;
for $i := 2$ to t do
  $k := \varsigma(E_i)$;
  if $k \in \{b_1, \ldots, b_r\}$
    then (* Let e_l be the l^{th} element of E_i *)
      $E'_{j+1} :=$ Swap e_k and e_{k+1} from E'_j;
      $j := j + 1$;
  end if
end for

Thus the elements betweens two borders are treated as an unordered list. However, since $b_i/b_{i-1} < 3$ and $b_r \geq n/3$, property (ii) — for every list $E_i \in \mathcal{E}$ there is a list $E'_j \in \mathcal{E}'$, such that the first m elements of E_i are among the first $3m$ elements of E'_j — is satisfied. Property (i) — two adjacent lists in $\mathcal{E}$ differ by a swap of two elements — is also satisfied by construction.

Using persistent search trees[19], we store the sequence $\mathcal{E}'$ of lists in $O(\lambda_{s+2}(n)\log n)$ space (by property (i)). Suppose the query half-plane is ℓ^+, and the slope of $\ell \in [d_i : d_{i+1}]$. Then list $E'_j \in \mathcal{E}'$ in which we should search can be determined by binary search. Next, we traverse E'_j sequentially and report all arcs that intersect ℓ^+, until we find an arc γ whose topmost point in the direction normal to ℓ does not lie in ℓ^+. We continue our traversal and report all arcs that intersect ℓ^+ until we reach the next border position b_m. By our construction, the topmost point (in the direction perpendicular to the slope of $\ell \in [d_i : d_{i+1}]$) of all arcs of E'_j after the border position lie below that of γ, and therefore

these arcs do not intersect ℓ^+, so we stop. By property (ii), we visit at most $3K$ elements when there are K answers to the query. Hence,

Theorem 4.4 *Given a set Γ of n arcs in the plane, each pair intersecting in at most s points, it can be preprocessed into a data structure of size $O(\lambda_{s+2}(n)\log n)$, such that for a query half-plane, all K arcs of Γ intersecting it can be reported in time $O(\log n + K)$.*

5 Line Intersection Queries

In this section we present efficient algorithms for the following problem: "Given a set Γ of arcs in the plane, preprocess it in into a data structure, such that, for a query line ℓ, all arcs of Γ intersecting ℓ can be reported efficiently."

We present three solutions for this problem. We first describe a data structure that works for general arcs. Then we present another structure — for sets of circular arcs — that uses only linear storage, but takes more time to answer a query. The third method considers an improvement in case there are only few answers, or when the planar graph formed by the arrangement of these arcs has only few connected components.

The main data structure consists of an s-tree, or a partition tree, of which each node is associated with a secondary data structure that solves a simpler problem, usually the half-plane query problem.

5.1 Line intersection queries for arcs

Our solution for general arcs works as follows. Let L be the set of left endpoints of arcs in Γ (recall that the arcs are x-monotone). Construct an s-tree $\mathcal{T}$ on L. Each node of $\mathcal{T}$ is associated with a subset $L_v \subseteq L$ of points stored at the leaves of the subtree rooted at v. Let $\Gamma_v \subseteq \Gamma$ be the set of arcs whose left endpoints are in L_v. At v we preprocess Γ_v for half-plane queries (see Section 4), and we also store $CH(L_v)$, the boundary of the convex hull of L_v.

As to processing a query, we visit $\mathcal{T}$ in a top-down fashion starting at the root. At each node v we determine whether the query line ℓ intersects $CH(L_v)$. If the answer is positive, we visit both children of v. Otherwise, $CH(L_v)$ lies in one of the half-planes determined by ℓ, say ℓ^+. In this case, an arc $\gamma \in \Gamma_v$ intersects ℓ if and only if it intersects the half-plane ℓ^-, because its left endpoint lies above ℓ. Thus, using the half-plane query structure of Section 4.2, we can report all K_v arcs of Γ_v intersecting ℓ in time $O(\log n + K_v)$. In view of Corollary 2.4 and Theorem 4.4, we have

Theorem 5.1 *Given a set Γ of n arcs in the plane with at most s pairwise intersection points, we can build a data structure of size $O(\lambda_{s+2}(n)\log^2 n)$, such that for a query line ℓ, all K arcs of Γ intersecting ℓ can be reported in time $O(\sqrt{n}\log^2 n + K)$.*

Remark. If Γ is a collection of homothets, it can be preprocessed into a data structure of size $O(n \log^2 n)$, such that line intersection queries can be answered in $O(\sqrt{n} \log^2 n + K)$ time (because a pair of homothets has only two supporting lines with the homothets to the one side).

5.2 Line intersection queries for circular arcs

We next offer an alternative data structure that allows for reporting intersections between a set Γ of circular arcs and a query line in time $O(n^{2/3+\epsilon})$, using $O(n)$ space, for any $\epsilon > 0$. The solution is based on Lemma 3.5 and uses multi-level partition trees based on ϵ-nets.

We build two data structures. The first one reports the arcs that intersect the query line in one point, while the other structure reports the arcs that intersect the query line in two points. The first data structure is constructed as follows (cf. Lemma 3.5, case (i)).

Let L denote the set of left endpoints of arcs in Γ. Construct an ϵ-net based partition tree $\mathcal{T}$ on L. Each node of $\mathcal{T}$ is associated with a subset $L_v \subset L$. Let R_v denote the set of right endpoints of arcs whose left endpoints are in L_v. At each node v, we preprocess R_v into a linear size data structure for half-plane range searching, using the algorithm of Chazelle et al. [12].

To answer a query, we first determine the set of canonical nodes $v \in \mathcal{T}$. For each canonical node v, we access its secondary structure. If the left endpoints associated with v lie in the half-plane ℓ^+ (resp. ℓ^-), we report all points of R_v that lie in ℓ_v^- (resp. ℓ_v^+). We can apply the space reduction technique of Section 2.3 to this structure. By Corollary 2.5, the overall query time is $O(n^{2/3+\epsilon}+K)$ and the space used is $O(n)$. If we use the algorithm of Chazelle and Welzl [15] for half-plane range searching instead of Chazelle et al. [12], we count, in time $O(n^{2/3+\epsilon})$, the number of arcs intersecting ℓ in one point.

Next we construct a data structure to report the arcs that satisfy (ii) of Lemma 3.5. A line ℓ intersects an arc γ at two points if and only if

(i) the left endpoint of γ lies above (resp. below) ℓ,

(ii) the right endpoint of γ lies above (resp. below) ℓ,

(iii) ℓ intersects the circle containing γ, and

(iv) $m \in \delta_\gamma$, where m is the slope of ℓ, and δ_γ is defined to be the interval
$\{ m \in \mathbf{R} \mid$ lower (resp. upper) tangent of $circ(\gamma)$ with slope m touches $\gamma \}$.

The first three conditions are clear. The fourth condition, in combination with the first two, ensures that γ' lies above (resp. below) ℓ. (Recall that γ' is the other part of $circ(\gamma)$, see Section 3.) Notice that the interval δ_γ is either empty, or consists of one single interval, or consists of two intervals.

We construct the following four-level data structure, whose j^{th} level sifts out the arcs satisfying the j^{th} condition. We only describe a structure which tests whether the left endpoint, the right endpoint, and γ' lie *above* ℓ; the other case (*below* ℓ) is completely symmetric, and can be solved with another structure of the same type. First we construct an ϵ-net based partition tree on the left endpoints of arcs of Γ. For a node v of the tree, let R_v be the set of right endpoints of arcs whose left endpoints are associated with the node v. We now construct an ϵ-net based partition tree on R_v and store it at v as its secondary structure. Next, for a node w of a second-level structure, let $\mathcal{C}_w$ denote the set of circles containing the arcs whose right endpoints are associated with w. We construct an s-tree on the set $\varphi_3(\mathcal{C}_w)$ of points in $\mathbf{R}^3$ (see Section 2 for the transform φ_3), and store it at w. Finally, for each third-level node z, we consider all arcs corresponding to points associated with z; let Γ_z be the set of these arcs. For all arcs $\gamma \in \Gamma_z$, we store the intervals δ_γ in an interval tree $\mathcal{I}_z$ (see [21, 35]).

To report the set of arcs intersected by a query line ℓ, we query the first-level structure with the half-plane ℓ^+. The arcs corresponding to each canonical node of the query output have their left endpoints lying above ℓ. For each canonical node, we query its second-level structure with ℓ^+. The second-level canonical nodes give the arcs whose right endpoints lie above ℓ. Thus, the first two levels together give the set of arcs whose both endpoints lie above ℓ. We now extract the arcs γ for which $circ(\gamma)$ intersects ℓ. To sift out these arcs, we map ℓ to the wedge $\chi_3(\ell)$ in $\mathbf{R}^3$ and query the third-level structure stored at every second-level canonical node. By Lemma 3.2, we obtain the set of arcs satisfying the third condition.

We now need to output those arcs γ of the third-level canonical nodes for which γ' lies above ℓ, and this is the case if the line parallel to ℓ and tangent to the lower half-circle of $circ(\gamma)$ touches γ, i.e., if $m \in \delta_\gamma$, where m is the slope of ℓ. Thus, we search in the interval tree $\mathcal{I}_z$ with m and report all arcs corresponding to the intervals containing the slope m.

Finally, we query in the symmetrically defined structure, but we now search with the half-plane ℓ^- in the first- and second-level structures, with the wedge $\chi_3(\ell)$ in the third-level structures, and with the slope m of ℓ in the interval tree defined for the upper tangents.

Since the time spent to report K_v arcs at a structure on the fourth level is $O(\log n + K_v)$, the total query time, by Theorems 2.1 and 2.3, is $O(n^{2/3+\epsilon} + K)$, for any $\epsilon > 0$. Using the technique of Section 2.3, we can reduce the overall space to $O(n)$.

Finally, we can change the structure into one that counts the number of arcs intersecting the query line. We do this by using the counting version of the interval tree. This leads to the same space and query time bounds as for the reporting version. Hence, we obtain

Theorem 5.2 *A set of n circular arcs in the plane can be stored in $O(n)$ space, such that all K arcs intersecting a query line can be reported in $O(n^{2/3+\epsilon}+K)$ time, for arbitrarily small positive ϵ. Furthermore, we can count the number of such arcs in time $O(n^{2/3+\epsilon})$.*

5.3 Line intersection queries in arcs: an alternative

Let $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ be a set of n arcs in $\mathbf{R}^2$. We first consider the case where the planar graph formed by $\mathcal{A}(\Gamma)$ — the arrangement of the n arcs in Γ — is connected. Our algorithm relies on the observation that if $\mathcal{A}(\Gamma)$ forms a connected graph, then a line ℓ intersects an arc of Γ if and only if ℓ intersects the convex hull $CH(\Gamma)$.

Lemma 5.3 *Let Γ be a set of n arcs in the plane with the property that $\mathcal{A}(\Gamma)$ is a connected graph. Then there exists a sequence $\mathcal{E} = \zeta_1, \zeta_2, \ldots, \zeta_m$, $m \leq 6n-4$, of (not necessarily distinct) subarcs of arcs of Γ, such that for any subsequence $\mathcal{E}'$ of $\mathcal{E}$, $\mathcal{A}(\mathcal{E}')$ is also a connected graph.*

As we will see below, each subarc appears in $\mathcal{E}$ exactly twice, although one arc may be split into as many as n subarcs. Once having computed $\mathcal{E}$, we construct a minimum height binary tree $\mathcal{T}$ on $\mathcal{E}$ whose leaves store the subarcs of $\mathcal{E}$ ordered in the leaves. Each internal node v of $\mathcal{T}$ stores CH_v, the convex hull of the arcs stored at the leaves of the subtree rooted at v. A line ℓ intersects CH_v if and only if it intersects one of the arcs stored in the subtree rooted at v (because the arrangement of these arcs is a connected graph). Therefore, to answer a query, we proceed in a top-down fashion. At each node v, we determine whether ℓ intersects CH_v; if the answer is positive, we visit both of its children. Otherwise, we do not continue in the subtree rooted at v. The number of nodes visited is $O((1+K)\log\frac{n}{1+K})$, where K is the number of subarcs of $\mathcal{E}$ intersecting ℓ. Since any line intersects any arc in at most a constant number of points, the number of arcs of Γ intersecting ℓ is also $\Theta(K)$. (If more than one subarc of the same arc γ intersects ℓ, then γ should be reported only once. It is, however, straightforward to take care of this.) The total time spent to find the subarcs is $O((1+K)\log\frac{n}{1+K})$ (see [11]). The space required by the data structure is obviously $O(n\log n)$.

Proof of Lemma 5.3: We now show how to compute $\mathcal{E}$. Let G be the intersection graph of Γ, that is, it has n vertices — one for each arc — and two of its vertices are connected by an edge if the corresponding arcs intersect. We compute a spanning tree T of G. For each edge (γ_i, γ_j) of T, we choose an intersection point of γ_i and γ_j (by construction, such a point does exist). If γ_i and γ_j intersect at more than one point, we choose one of them arbitrarily. Let P be the set of $n-1$ resulting points. Since $\mathcal{A}(\Gamma)$ is a connected graph, each arc of Γ contains at least one point of P. See Figure 3.

We now construct another graph G' whose vertices are the $n-1$ points of P and the $2n$ endpoints of arcs in Γ, and whose $3n-2$ edges are the maximal portions of arcs of Γ that do not contain any vertex of G'. The maximal portions of arcs of Γ that do not

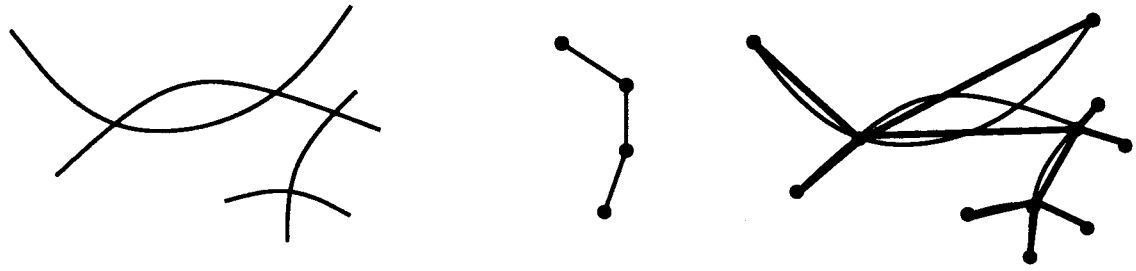

Figure 3: A set Γ of arcs, its intersection graph G, and the graph G'.

contain a point of P are the subarcs that will appear in $\mathcal{E}$. Since G' does not include all the vertices of $\mathcal{A}(\Gamma)$, its edges may intersect.

A cycle in G' would imply that T also has a cycle, which contradicts the fact that T is a spanning tree, and therefore G' is a tree. We now duplicate every edge in G', and construct an Eulerian tour Π in G'. The sequence of edges of G' ordered along Π gives the desired sequence $\mathcal{E}$. Note that each subarc of Γ appears exactly twice in $\mathcal{E}$, and that $\mathcal{E}$ has length $6n - 4$. Since a contiguous subsequence of $\mathcal{E}$ corresponds to a contiguous portion of Π, the arrangement formed by the arcs in this subsequence forms a connected graph. □

We have thus shown that

Lemma 5.4 *Given a set Γ of n arcs with the property that $\mathcal{A}(\Gamma)$ forms a connected graph, we can preprocess it into a data structure of size $O(n \log n)$, such that, for a query line ℓ, all K arcs of Γ intersecting ℓ can be reported in time $O((1+K)\log \frac{n}{1+K})$.*

Finally, consider the general case where $\mathcal{A}(\Gamma)$ does not form a connected graph. Let $\Gamma_1, \ldots, \Gamma_m$ be the partitioning of Γ into connected components. For each Γ_i, we construct an Eulerian path Π_i as above. Next, we choose the endpoint p_i of each Π_i (the point where the Eulerian tour begins and ends), and construct a spanning path Π' of stabbing number $O(\sqrt{m})$ on these endpoints[15, 30]. We next replace each vertex p_i of Π' by the corresponding path Π_i, giving an overall path Π. We construct a minimum height binary tree on Π and preprocess for line intersection queries as before. A query is answered in exactly the same way as before. Since the number of nodes visited is $O(\sqrt{m}\log n + K \log \frac{n}{1+K})$, we obtain

Theorem 5.5 *Given a set Γ of n arcs, such that $\mathcal{A}(\Gamma)$ has m connected components, we can preprocess Γ into a data structure of size $O(n \log n)$, such that, for a query line ℓ, all K arcs of Γ intersecting ℓ can be reported in time $O(\sqrt{m}\log n + K \log \frac{n}{1+K})$.*

6 Segment Intersection Queries

In this section we study the same problems as above, except that the query ranges are now segments instead of lines. We describe segment intersection queries for disks, circles, circular arcs and disjoint arcs. An efficient algorithm for general arcs is still lacking. To keep this paper to reasonable length, we often describe the main ideas and leave the details to the reader.

6.1 Segment intersection queries for disks

Let s be a query segment, and let p_s, q_s, ℓ_s, ℓ_p, ℓ_q, $strip_s$ be as defined in Section 3. For reporting intersections between a set $\mathcal{D}$ of disks and a query segment s, we rely on Lemma 3.3. We construct two data structures — one for reporting the disks satisfying (i) of the lemma, and the other to report the disks satisfying (ii) of the lemma.

The first data structure is used to report the disks that contain at least one of the endpoints of s. We compute the set $\varphi_2(\mathcal{D})$ of n points in $\mathbf{R}^3$ and preprocess it into the data structure of size $O(n \log n)$ for the half-space range searching (see [5] and Section 4.1). By Lemma 3.1, the set of disks containing an endpoint of s is the same as that subset of $\varphi_2(\mathcal{D})$ of points that lie below $\psi_2(p_s)$ or $\psi_2(q_s)$. Therefore, all K such disks can be reported in time $O(\log n + K)$. If we construct an s-tree on $\varphi_2(\mathcal{D})$, we can count the number of disks satisfying (i) of Lemma 3.3 in time $O(n^{2/3} \log^2 n)$, using $O(n \log n)$ space.

The second structure reports the disks that intersect ℓ_s and whose centers lie in the strip $strip_s$. This data structure is basically the same as the one in Section 5.1. Let P be the set of centers of disks of $\mathcal{D}$. We now construct an s-tree $\mathcal{T}$ on P as in Section 5.1, but at each node of the tree we now store the structure of Lemma 4.1 (instead of Theorem 4.4) as the secondary structure. To process a query with segment s, we first find $O(\sqrt{n} \log n)$ canonical nodes of $\mathcal{T}$ with respect to ℓ_s and $strip_s$. Let v be a canonical node for which $CH(P_v)$ lies in $strip_s$, and above (resp. below) ℓ_s. Then we report all disks of $\mathcal{D}_v$ that intersect the half-plane ℓ_s^- (resp. ℓ_s^+) by searching in the secondary structure at v. The correctness of the algorithm follows immediately from Lemma 3.3. Since the secondary search requires $O(\log n + K_v)$ time, the overall query time is $O(\sqrt{n} \log^2 n + K)$.

If we use partition trees for the first-level structure and s-trees for the second-level structure, we can count the number of arcs intersecting the query line and satisfying Lemma 3.3 in time $O(n^{2/3+\epsilon})$, for any $\epsilon > 0$, using linear space. We leave the details for the reader.

Theorem 6.1 *Given a set $\mathcal{D}$ of n disks in the plane, we can preprocess it into a data structure of size $O(n \log^2 n)$, such that for a query segment, all K disks of $\mathcal{D}$ intersecting it can be reported in time $O(\sqrt{n} \log^2 n + K)$. Moreover, the number of such disks can be counted in time $O(n^{2/3+\epsilon})$, for any $\epsilon > 0$, using a data structure of linear size.*

6.2 Segment intersection queries for circles

Next we consider segment intersection queries for a set $\mathcal{C}$ of circles. Our data structure follows Lemma 3.4 and consists of two trees, one for each of the cases.

For the first structure, we construct an s-tree on the set $\varphi_2(\mathcal{C})$ of points in $\mathbf{R}^3$. Let s be the query segment. By Lemma 3.1, a circle C contains exactly one endpoint of s if and only if $\varphi_2(C)$ lies above $\psi_2(p)$ and below $\psi_2(q)$, or vice versa. So the query can be answered by reporting the points of $\varphi_2(\mathcal{C})$ that lie in the double wedge bounded by $\psi_2(p)$ and $\psi_2(q)$, and not containing the vertical plane passing through $\psi_2(p) \cap \psi_2(q)$. By Theorem 2.3, this can be accomplished in time $O(n^{2/3}\log^2 n + K)$ using $O(n \log n)$ space.

For the second structure, we construct an s-tree $\mathcal{T}$ on the set $\varphi_3(\mathcal{C})$ of points in $\mathbf{R}^3$. For a node v of $\mathcal{T}$, let $\mathcal{C}_v$ be the circles corresponding to the points associated with it. We preprocess the set of points $\varphi_2(\mathcal{C}_v)$ for half-space range searching [5]. Since the secondary structure on a set of n_v points requires $O(n_v \log n_v)$ space, the overall space used is $O(n \log^2 n)$.

For a query segment s, we report all circles intersecting s as follows. We first query in the first structure as described above. To query in the second structure, recall that we want to report the circles that intersect ℓ_s, whose centers lie in the $strip_s$ (see Section 3 for the definition), and which do not contain both endpoints of s in their interior. We first define a prism $prism_s$ in $\mathbf{R}^3$ for s. Associate the plane in which $\mathcal{C}$ lies with the xy-plane. Let H_p (resp. H_q) be the vertical plane erected on the line ℓ_p (resp. ℓ_q), then $prism_s$ is the portion of the wedge $\chi_3(\ell_s)$ that lies between H_p and H_q (see Figure 4). It follows from Lemma 3.2 that a circle C whose center lies in $strip_s$ intersects ℓ_s if and only if $\varphi_3(C)$ lies in $prism_s$. Thus, to answer a query we find the set of canonical nodes of $\mathcal{T}$ with respect to $prism_s$. There are $O(n^{2/3}\log n)$ such nodes, and they can be determined in time $O(n^{2/3}\log^2 n)$. If the points of a canonical node v lie in $prism_s$, we use the secondary structure stored at v to report the circles corresponding to the points that lie above the plane $\psi_2(p_s)$ or $\psi_2(q_s)$. These are circles that do not contain the endpoint p_s or q_s of s. Since the secondary structure requires $O(\log n + K_v)$ time to report K_v circles, the overall time to report all K segments is $O(n^{2/3}\log^2 n + K)$. Hence, we have

Theorem 6.2 *Given a set of n circles in the plane, we can build a data structure of size $O(n \log^2 n)$, so that for a query segment, all K circles intersecting it can be reported in time $O(n^{2/3}\log^2 n + K)$.*

6.3 Segment intersection queries for circular arcs

Let Γ be a collection of circular arcs. Analogous to the previous subsections, we construct two data structures — one of them reports the arcs of Γ that satisfy the first condition of Lemma 3.6, and the other structure reports the arcs that satisfy the second condition.

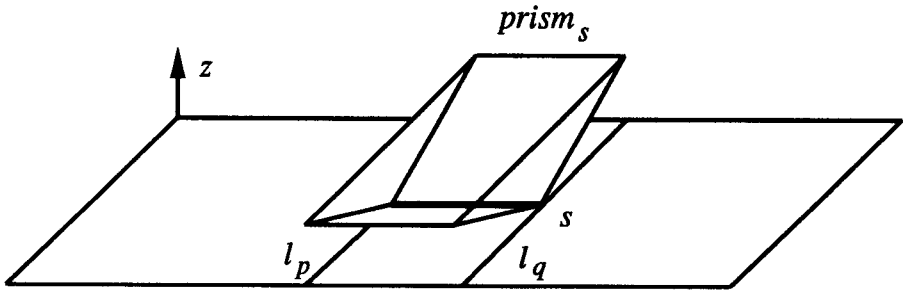

Figure 4: Segment s and $prism_s$.

To report the arcs that satisfy (i) of Lemma 3.6, we first construct a two-level partition tree, as in Section 5.2, to select those arcs of Γ whose endpoints are separated by ℓ_s. For each second-level node v, let Γ_v denote the subset of arcs that are associated with v. Let $\tilde{\Gamma}_v$ denote the set of lines passing through the endpoints of the arcs of Γ_v. Partition Γ_v in two subsets $\Gamma_{v,1}$ and $\Gamma_{v,2}$, such that every arc $\gamma \in \Gamma_{v,1}$ lies above the line through the endpoints of γ, and every arc $\gamma \in \Gamma_{v,2}$ lies below the line through its endpoints. We construct as third-level structures one partition tree on $\tilde{\Gamma}^*_{v,1}$ — the points dual to the lines in $\tilde{\Gamma}_{v,1}$ — and one partition tree on $\tilde{\Gamma}^*_{v,2}$. For a node w in any of the third-level structures, let C_w be the set of circles containing the arcs corresponding to the points associated with w. We now construct an s-tree on $\varphi_2(C_w)$.

Let s be a query segment. We query the first two levels of the structure with ℓ_s as in Section 5.2; this gives the subset of arcs of Γ whose endpoints are separated by s. To report the arcs that satisfy (i.a) of Lemma 3.6, we dualize the endpoint p_s to the line p^*_s, and search the third-level structure for $\tilde{\Gamma}^*_{v,1}$ with the half-plane p^{*+}_s, and also the third-level structure for $\tilde{\Gamma}^*_{v,2}$ with the half-plane p^{*-}_s. Finally, we search the fourth-level structure of each third-level canonical node determined by the query with the wedge formed by $\psi_2(p_s)^+ \cap \psi_2(q_s)^-$. We then repeat the whole process after exchanging the roles of p_s and q_s. It follows from Lemma 3.6 that the fourth-level canonical nodes of the query represent the desired arcs, which are reported by traversing their subtrees.

To report the arcs that satisfy (i.b) of the lemma, we process a query in the same way except that at the third level, we search with the wedge $p^{*+}_s \cap q^{*+}_s$ in $\tilde{\Gamma}^*_{v,1}$, and with the wedge $p^{*-}_s \cap q^{*-}_s$ in $\tilde{\Gamma}^*_{v,2}$. At the fourth level we search with the wedge $\psi_2(p_s)^+ \cap \psi_2(q_s)^+$. The overall query time is $O(n^{2/3+\epsilon} + K)$ to report K arcs, by Corollaries 2.5 and 2.6.

Similarly, one can construct another data structure to report all K arcs that satisfy (ii) of Lemma 3.6 in time $O(n^{2/3+\epsilon})$. The structure is a combination of the ones of Sections 5.2 and 6.2. Furthermore, we can count the number of arcs intersecting s in time $O(n^{6/7+\epsilon})$, using a related data structure. However, we use three-dimensional partition trees based on ϵ-nets rather than s-trees in some levels, which leads to the query time stated above.

The space reduction technique of Section 2.3 is applied to all partition trees and s-trees, which leads to

Theorem 6.3 *A set of n circular arcs in the plane can be stored in $O(n \log n)$ space, such that segment queries take $O(n^{2/3+\epsilon} + K)$ time. Furthermore, one can count the number of these arcs in time $O(n^{6/7+\epsilon})$, using a data structure of linear size.*

6.4 Segment intersection queries for disjoint arcs

In this subsection we consider the case where Γ is a set of non-intersecting Jordan arcs. We use the ray shooting algorithm, described in the next section, to report the arcs intersected by a query segment. A similar approach was used by Agarwal [1] to report the segments intersected by a query segment. The basic idea is as follows.

Let s be the query segment and let d be the direction of $p_s\vec{q}_s$. We shoot a ray ρ from p_s in direction d. If ρ does not intersect Γ, or the first intersection point μ lies beyond q_s, we stop, otherwise we report the arc containing μ, shoot again from μ in direction d, and repeat the same step. We will show in the next section that the first ray shooting query takes $O(\sqrt{n} \log n)$ time but each subsequent ray shooting query can be answered in time $O(\log n)$. If K arcs of Γ intersect s, we perform $O(K+1)$ ray shooting queries, because every ray shooting query except the last one returns a distinct intersection point of s and Γ, and an arc of Γ intersects s at $O(1)$ points. We thus have

Theorem 6.4 *A set Γ of n disjoint arcs in the plane can be stored in $O(n \log n)$ space, such that the K arcs of Γ intersecting a query segment can be reported in $O((\sqrt{n}+K) \log n)$ time.*

7 Ray Shooting Queries

In this section we study the *ray shooting* problem in arrangements of Jordan arcs: "Given a set Γ of n arcs in the plane, preprocess it, so that for any query ray ρ emanating from a point p in direction d, the first intersection point of Γ and ρ, denoted $\Phi(\Gamma, \rho)$, can be determined efficiently."

We propose two algorithms for this problem. The first algorithm works for disjoint arcs, while the second one works for possibly intersecting circular arcs.

7.1 Ray shooting among disjoint arcs

In this subsection we assume that the arcs in Γ are pairwise disjoint. We partition Γ into $j+1 \leq \sqrt{n}$ subsets $\Gamma_0, \Gamma_1, \ldots, \Gamma_j$ such that (i) any line intersects at most $\sqrt{n}$ arcs of Γ_0,

and (ii) for every $i \geq 1$, there is a line ℓ_i that intersects all arcs of Γ_i. How we obtain such a partitioning of Γ is indicated later. For the time being assume that such a partitioning of Γ together with the lines ℓ_i are available.

Preprocess Γ_0 for line intersection queries as in Section 5.3. For $i \geq 1$, Γ_i is preprocessed as follows. Observe that every face in the arrangement $\mathcal{A}(\Gamma_i \cup \ell_i)$ of $\Gamma_i \cup \ell_i$ is simply connected (a possibly unbounded polygon with curved edges). Therefore, each face can be preprocessed into a data structure of linear size for $O(\log n)$ time ray shooting queries by modifying the algorithm of Chazelle et al. [9, 38]. We also preprocess $\mathcal{A}(\Gamma_i \cup h_i)$ for planar point location [36], to determine the face in which the query ray starts. Since the number of edges in $\mathcal{A}(\Gamma_i \cup \ell_i)$ is $O(|\Gamma_i|)$, and the data structure for Γ_0 requires $O(|\Gamma_0| \log |\Gamma_0|)$ space, the total size of the data structure over all $i \geq 0$ is easily seen to be $O(n \log n)$.

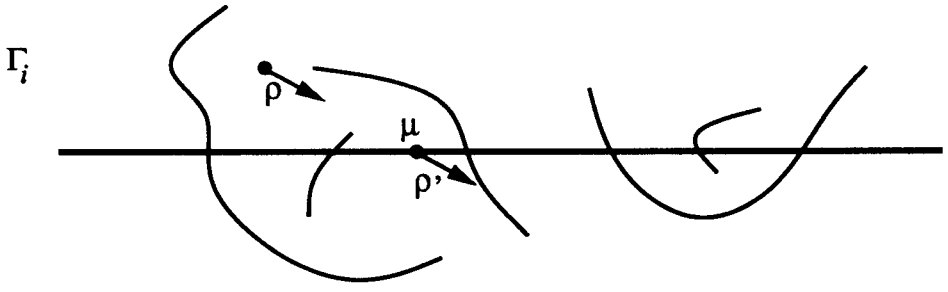

Figure 5: Ray shooting in arcs intersected by a common line.

As for computing $\Phi(\Gamma, \rho)$ for a query ray ρ, we first determine all arcs of Γ_0 intersecting ℓ_ρ, the line containing ρ, and then determine the first arc of Γ_0 hit by ρ. Since ℓ_ρ intersects at most $\sqrt{n}$ arcs of Γ, by Theorem 5.5, this step requires $O(\sqrt{n} \log n)$ time. Next, for each $i \geq 1$, we do the following. Locate the starting point p in $\mathcal{A}(\Gamma_i \cup \ell_i)$. Let f_i be the face containing p. Denote with μ_i the first intersection of ρ with $\mathcal{A}(\Gamma_i \cup \ell_i)$. Then $\mu_i = \Phi(\Gamma_i \cup \ell_i, \rho) = \Phi(f_i, \rho)$, and $\Phi(f_i, \rho)$ is computed by performing a ray shooting query in the simply connected region f_i. If μ_i is undefined or lies on an arc of Γ_i, we are done. Otherwise, μ_i lies on ℓ_i. Let ρ' be the ray starting from μ_i in the same direction as ρ. We now compute $\Phi(\Gamma_i \cup \ell_i, \rho')$ by ray shooting in another face f_i' of $\mathcal{A}(\Gamma_i \cup \ell_i)$. Since ρ cannot intersect ℓ_i more than once, shooting with ρ' in f_i' gives the correct answer in Γ_i, thus $\Phi(\Gamma_i, \rho) = \Phi(\Gamma_i \cup \ell_i, \rho')$ in this case. Among all $\Phi(\Gamma_i, \rho)$, we choose the one that is closest to s. Since we spend $O(\log n)$ time for each Γ_i, the total query time is $O(\sqrt{n} \log n)$.

Theorem 7.1 *Given a set Γ of n disjoint arcs in the plane, we can preprocess it into a data structure of size $O(n \log n)$, such that, for a query ray ρ, $\Phi(\Gamma, \rho)$ can be computed in $O(\sqrt{n} \log n)$ time.*

Remark. As in [1], we can compute the first k intersection points of ρ and Γ in time $O((\sqrt{n}+k)\log n)$ time. In particular, suppose we have already computed the first i intersection points, $\xi_1, \ldots, \xi_i$. To compute ξ_{i+1}, we shoot a ray from ξ_i along ρ. The first ray shooting query requires $O(\sqrt{n}\log n)$ time, but all subsequent queries require only $O(\log n)$ time; see [1] for details.

Remark. The method above also works for (intersecting) disks, but not for circles.

We sketch how the partitioning of Γ into subsets $\Gamma_0, \ldots, \Gamma_j$, can be obtained. We have the requirement that no line intersects more than $\sqrt{n}$ arcs of Γ_0. For the sets Γ_i, $1 \le i \le j$, we have the requirements that there is a line ℓ_i that intersects all arcs of Γ_i, and Γ_i has at least $\sqrt{n}$ arcs.

Define a *common tangent* of two arcs γ_1 and γ_2 to be a line ℓ containing a point of each arc, and such that no point on γ_1 lies strictly to some side of ℓ, and the same holds for γ_2 (the side may be the different or the same as for γ_1). A pair of disjoint arcs has at most four common tangents. One can show that the common tangents of Γ are the only lines to consider for $\ell_1, \ldots, \ell_j$. Compute these $O(n^2)$ common tangents, and sort the corresponding set of slopes.

Consider a horizontal line l, and project all arcs of Γ vertically onto this line. This gives a set of segments on l. The idea is to dynamically maintain a segment tree on these segments, when the line l rotates (and the projection remains normal to the line). The event points are now the slopes of the common tangents. When an event causes a point on l to be covered by $\sqrt{n}$ intervals, we have found a line (normal to l) intersecting $\sqrt{n}$ arcs. We delete the arcs of this subset to form Γ_i, we increase i, and continue with the rest. The arcs that remain in the segment tree after a full rotation form the set Γ_0. Straightforward analysis leads to an $O(n^2 \log n)$ time algorithm to compute the partitioning of Γ as required.

7.2 Ray shooting among circular arcs

In this subsection we describe the ray shooting structure for a set Γ of (possibly intersecting) circular arcs. It is based on the random sampling technique and constructs a multi-level partition tree on the arcs of Γ.

The primary data structure is a variant of the partition tree based on ϵ-nets. The root of the tree is associated with the set Γ itself. Let P be the set of endpoints of the arcs of Γ. We choose a random subset $R \subset P$ of size r, where r is some suitable constant. We compute the arrangement of the $\binom{r}{2}$ lines passing through each pair of points of R, and triangulate their arrangement. The resulting subdivision, denoted by $\mathcal{M}$, consists of $O(r^4)$ triangles. We create a child for each triangle τ of $\mathcal{M}$. For a triangle $\tau \in \mathcal{M}$, let Γ_τ denote the set of subarcs $\gamma \cap \tau$, where $\gamma \in \Gamma$. We call an arc $\gamma \in \Gamma_\tau$ *short* if at least one of its endpoints lies in the interior of τ, and *long* if both endpoints of lie on the boundary

of τ (see Figure 6, left). Let n_τ be the number of short arcs in τ, then we have

$$\sum_{\tau \in \mathcal{M}} n_\tau \leq 2n \tag{1}$$

and, by the theory of random sampling [26], with high probability we have for every line ℓ

$$\sum_{\tau \in \mathcal{M}, \tau \cap \ell \neq \emptyset} n_\tau \leq c \cdot \frac{n}{r} \log r, \tag{2}$$

where c is some fixed constant. For each triangle τ, the short arcs of Γ_τ will be preprocessed for ray shooting recursively. For long arcs, we associate the following secondary structure to find the first long arc of Γ_τ hit by the query ray ρ.

Let $L \subseteq \Gamma_\tau$ denote the set of long arcs of τ, and let $\mathcal{C}_L$ denote the set of circles that contain the arcs of L. We construct an s-tree $\mathcal{T}_\tau$ on the set $\varphi_2(\mathcal{C}_L)$ of points in $\mathbf{R}^3$. The s-tree allows us to obtain canonical sets of nodes that represent the common interior of the circles in $\mathcal{C}_L$, or the common exterior. Let $L_v \subseteq L$ be the set of arcs corresponding to the points associated with a node v of $\mathcal{T}_\tau$, and let $\mathcal{C}_v$ be the set of circles containing the arcs of L_v.

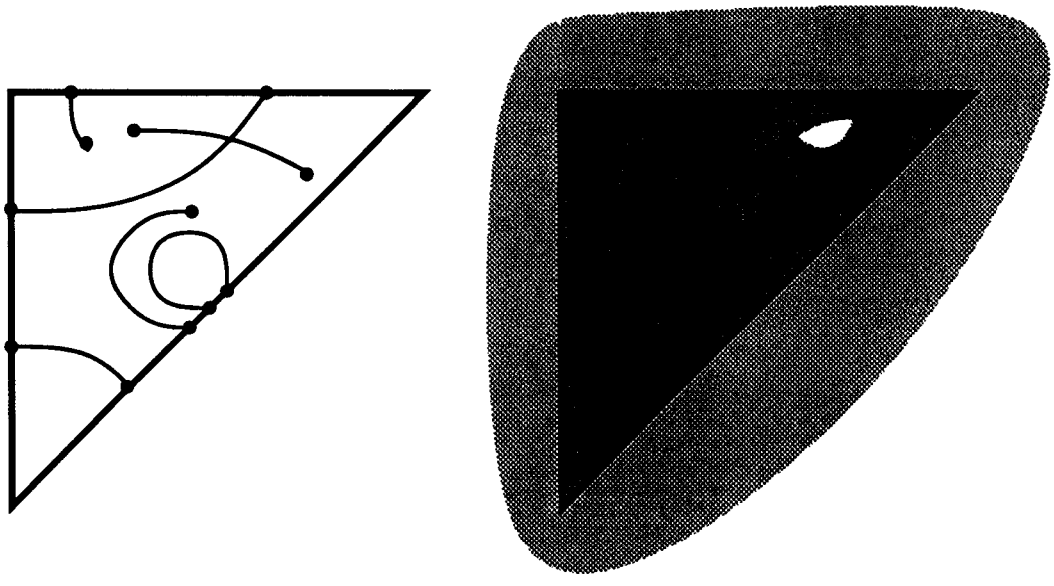

Figure 6: Ray shooting in circular arcs. Left, a triangle with long and short arcs intersecting it. Right, a triangle with long arcs, and the unbounded component of the common exterior shaded. The faces of F_E are shaded heavily, so the white region inside τ is the only region not in F_E. p_E is some point in the heavily shaded region and outside the circles. p_I does not exist.

We have a triangle τ, its boundary $\partial\tau$, a set of long arcs L_v, and the corresponding set $\mathcal{C}_v$ of circles. We will preprocess the common interior and the common exterior for ray shooting. Let p_I be a point inside τ and in the common interior of $\mathcal{C}_v$. If no such p_I exists, we are done with the interior. Otherwise, let f_I denote the face of the arrangement $\mathcal{A}(L_v \cup \partial\tau)$ that contains p_I. The face f_I is a simply connected region, and we preprocess it for efficient ray shooting by using the algorithm of Chazelle et al. [9, 38].

Let p_E be a point inside τ, and in the unbounded face of $\mathcal{A}(\mathcal{C}_v)$ (see Figure 6). Again, p_E may not exist, in which case we are done. Otherwise, denote the unbounded face that contains p_E *in the arrangement* $\mathcal{A}(L_v)$ by f_E. Consider the set F_E of faces, which consists of all simply connected regions of $f_E \cup \tau$. We preprocess F_E for efficient planar point location [36], so we can determine in which face of F_E a query point lies. Furthermore, every face of F_E is preprocessed for efficient ray shooting [9, 38]. This concludes the description of the data structure.

We query in the above structure with a ray ρ, starting at a point p. In order to determine $\Phi(\Gamma, \rho)$ we trace ρ through the convex subdivision $\mathcal{M}$, and visit the triangles in the order along ρ. If τ is the first triangle in $\mathcal{M}$ (the one that contains p), then we first determine whether the ray ρ hits any arc in τ. If the answer is yes, we find the first arc in τ intersecting ρ. Otherwise, we continue with the next triangle τ' and move the source point p to the first intersection point of ρ and τ'.

We determine the first short arc of τ that intersects ρ by recursively searching through the subtree rooted at τ. To determine the first long arc of τ hit by ρ, we shift to $\mathbf{R}^3$, and in such a way that the set Γ lies in the xy-plane. Let H_ρ be the vertical plane normal to ρ and passing through its starting point p. Let H_ρ^+ denote the half-space containing ρ. We query in the secondary s-tree stored at τ with the planes H_ρ and $\psi_2(p)$. For a canonical node v of the search, p lies either in the common interior of $\mathcal{C}_v$ or in the common exterior. In the former case, we can find the first intersection point of ρ and L by performing a ray shooting query in the face f_I stored at v. If p lies in the common exterior of $\mathcal{C}_v$ and $\varphi_2(\mathcal{C}_v) \in H_\rho^-$, then ρ cannot intersect any circle of $\mathcal{C}_v$, so we ignore v. Finally, if p lies in the common exterior of $\mathcal{C}_v$, and $\varphi_2(\mathcal{C}_v) \in H_\rho^+$, then p must lie in the unbounded face of $\mathcal{C}_v$. Furthermore, ρ intersects a circle $C \in \mathcal{C}_v$ if and only ℓ_ρ intersects C, which implies that $\Phi(L_v, \rho) \in f_E$. Therefore, we locate the face f of F_E that contains p and compute $\Phi(f, \rho)$. We perform this step for all second-level canonical subsets of the search, and then choose the intersection point that is closest to p to determine the first long arc of τ hit by ρ. As we already mentioned, if ρ hits any short or long arc of τ, we do not go further than τ. Otherwise, we visit the next triangle of $\mathcal{M}$ that is intersected by ρ.

Let $Q(n)$ denote the maximum query time. Since there are $O(n^{2/3}\log n)$ canonical nodes of $\mathcal{T}_\tau$ for $\psi_2(p)$, the total time spent at τ is $Q(n_\tau) + O(n^{2/3}\log^2 n)$. Hence, we get the following recurrence:

$$Q(n) = \sum_{i=1}^{O(r^2)} Q(n_i) + O(n^{2/3}\log^2 n),$$

where $\sum_i n_i \leq \frac{cn}{r} \log r$. The solution of this recurrence is known to be $O(n^{2/3+\epsilon})$, for any $\epsilon > 0$.

As to the storage requirements, one can prove — using Davenport-Schinzel arguments — that the size of the face f_I at a node v of the secondary s-tree is $O(m)$, and the total size of the faces of F_E is $O(m \cdot 2^{\alpha(m)})$, where $m = |L_v|$ (see [4, 25] and Section 2.4). With the space reduction technique of Section 2.3, the size of the whole structure is $O(n \cdot 2^{\alpha(n)})$.

Theorem 7.2 *Given a set Γ of n circular arcs, one can preprocess Γ into a data structure of size $O(n \cdot 2^{\alpha(n)})$ such that, for a query ray ρ, $\Phi(\Gamma, \rho)$ can be computed in time $O(n^{2/3+\epsilon})$, for any $\epsilon > 0$.*

8 Conclusions

In this paper we considered several intersection searching problems for a set of arcs. This is one of the first papers that deals with the query type problems for nonlinear objects. Most of the algorithms presented here are based on partition trees or spanning paths with low stabbing number, but they require a number of new techniques in order to handle nonlinear objects. Throughout this paper we generally ignored the issue of preprocessing time. In most cases the preprocessing time is dominated by the time required to construct a partition tree or a spanning path with low stabbing number. It is known that, for a set of n points in $\mathbf{R}^d$ ($d = 2, 3$), partition trees and s-trees can be constructed in time close to (within a polylogarithmic factor) n and $n^{2-1/d}$, respectively.

As mentioned in Section 2, more sophisticated partition trees have been developed recently for which the $\sigma_{\mathcal{T}}(n)$ is roughly $n^{1-1/d}$ in $\mathbf{R}^d$ (cf. [14, 31]). Unfortunately, they do not improve the query times of our algorithms, because in many of the data structures, we construct 2-dimensional partition trees and store a 3-dimensional s-tree at the bottom level. So, although we can reduce the number of canonical nodes at top levels to $n^{1/2}$ using the better partition trees, the overall number of canonical nodes still remains close to $n^{2/3}$. That is why we based our structures on Haussler-Welzl partition trees. However, s-trees in all data structures can be replaced by partition trees of Matoušek [31] or of Chazelle et al. [14] without affecting the query time significantly.

We conclude by mentioning some open problems:

(i) It seems that "counting" the number of intersecting arcs is harder than reporting the set of these arcs. It would be interesting if one could improve the query time of counting problems to $n^{1/2}$. Recall that Chazelle's lower bound implies that you cannot hope to do better than $n^{1/2}$ in the semigroup model [8].

(ii) Another interesting question would be to extend these algorithms to a more general case, where the query ranges are also nonlinear objects, e.g. report all arcs intersected by a query circle.

[12] Chazelle, B., L. J. Guibas, and D. T. Lee, The Power of Geometric Duality, *BIT* **25** (1985), pp. 76-90.

[13] Chazelle, B., and F. P. Preparata, Halfspace Range Search: An Algorithmic Application of k-Sets, *Discr. & Comp. Geom.* **1** (1985), pp. 83-93.

[14] Chazelle, B., M. Sharir, and E. Welzl, Quasi-Optimal Upper Bounds for Simplex Range Searching and New Zone Theorems, *Proc. 6th Ann. Symp. on Comp. Geometry* (1990), pp. 23-33.

[15] Chazelle, B., and E. Welzl, Quasi-Optimal Range Searching in Spaces of Finite VC-Dimension, *Discr. & Comp. Geom.* **4** (1989), pp. 467-489.

[16] Clarkson, K. L., and P. W. Shor, Applications of Random Sampling in Computational Geometry, II, *Discr. & Comp. Geom.* **4** (1989), pp. 387-421.

[17] Cole R. and C. Yap, Geometric Retrieval Problems, *Information and Control* **63** (1984), pp. 39-57.

[18] Dobkin, D. P. and H. Edelsbrunner, Space Searching for Intersecting Objects, *J. of Algorithms* **8** (1987), pp. 348-361.

[19] Driscoll, J. R., N. Sarnak, D. D. Sleator, and R. E. Tarjan, Making Data Structures Persistent, *J. of Comp. Sys. Sci.* **38** (1989), pp. 86-124.

[20] Edelsbrunner, H., *Algorithms in Combinatorial Geometry*, Springer-Verlag, Berlin, 1987.

[21] Edelsbrunner, H., *Dynamic Data Structures for Orthogonal Intersection Queries*, Rep. F59, Tech. Univ. Graz, 1980.

[22] Edelsbrunner, H., L. J. Guibas, J. Hershberger, R. Seidel, M. Sharir, J. Snoeyink, and E. Welzl, Implicitly Representing Arrangements of Lines or Segments, *Discr. & Comp. Geometry* **4** (1989), pp. 433-466.

[23] Edelsbrunner, H. and E. Welzl, Halfplanar Range Search in Linear Space and $O(n^{0.695})$ Query Time, *Inf. Proc. Lett.* **23** (1986), pp. 289-293.

[24] Guibas, L., M. Overmars, and M. Sharir, *Ray Shooting, Implicit Point Location, and Related Queries in Arrangements of Segments*, Techn. Rep. No. 433, New York University, 1989.

[25] Hart, S., and M. Sharir, Nonlinearity of Davenport-Schinzel Sequences and of Generalized Path Compression Schemes, *Combinatorica* **6** (1986), pp. 151-177.

[26] Haussler, D., and E. Welzl, ϵ-Nets and Simplex Range Queries, *Discr. & Comp. Geom.* **2** (1987), pp. 127-151.

[27] Kedem, K., R. Livne, J. Pach, and M. Sharir, On the Union of Jordan Regions and Collision-Free Translational Motion Amidst Polygonal Obstacles, *Discr. & Comp. Geom.* **1** (1986), pp. 59-71.

[28] van Kreveld, M., M. Overmars, and P. K. Agarwal, Intersection Queries in Sets of Disks, *SWAT 90*, Lect. Notes in Comp. Science 447, pp. 393-403.

[29] Matoušek, J., Construction of ϵ-Nets, *Discr. & Comp. Geom.* **5** (1990), pp. 427-448.

[30] Matoušek, J., Cutting Hyperplane Arrangements, *Proc. 6th Ann. Symp. on Comp. Geometry* (1990), pp. 1-9.

[31] Matoušek, J., Efficient Partition Trees, *Proc. 7th Ann. Symp. on Comp. Geometry* (1991), to appear.

[32] Matoušek, J., and E. Welzl, Good Splitters for Counting Points in Triangles, *Proc. 5th Ann. Symp. on Comp. Geometry* (1989), pp. 124-130.

[33] Overmars, M. H., H. Schipper, and M. Sharir, Storing Line Segments in Partition Trees, *BIT* **30** (1990), pp. 385-403.

[34] Overmars, M. H. and M. Sharir, Output Sensitive Hidden Surface Removal Algorithms, *Proc. 30th Symp. on the Found. of Comp. Science* (1989), pp. 598-603.

[35] Preparata, F. P. and M. I. Shamos, *Computational Geometry – an introduction*, Springer-Verlag, New York, 1985.

[36] Sarnak, N. and R. Tarjan, Planar Point Location Using Persistent Search Trees, *Communications of ACM* **29** (1986), pp. 609-679.

[37] Sharir, M., The k-Set Problem for Arrangements of Curves and Surfaces, to appear in *Discr. & Comp. Geom.*

[38] Snoeyink, J., private communication.

[39] Willard, D. E., Polygon Retrieval, *SIAM J. Comput.* **11** (1982), pp. 149-165.