Algorithms for Some Intersection Searching Problems Involving Circular Objects*

Prosenjit Gupta[†‡] Ravi Janardan[§†] Michiel Smid[¶]

Abstract

Two 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 circular objects, such as d-balls, d-spheres, circles, or circular arcs, and q is also a circular object. In the second class, the objects in S are circular 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 in the set of objects intersected by q.

Efficient algorithms are presented for several problems from these classes. The solution techniques are based on geometric transforms, on compositions of known solutions for simplex range searching, on the locus approach, and on persistent data structures.

Keywords: Computational geometry, data structures, intersection searching.

1 Introduction

In a generic instance of an *intersection searching problem* a set, S, of n geometric objects must be preprocessed, so that the k objects of S that are intersected by a query object q can be reported (or counted) efficiently. Examples of S and q are points, lines, line segments, rays, hyperplanes, and simplices. These problems arise in many applications and space- and query-time-efficient solutions are known for many of them [9, 20, 23].

Most previous work on these problems assumes that the input objects and the query object are linear or piecewise-linear (as above). To our knowledge, the case where the input and/or the query are curved has been investigated systematically only in [1, 2, 4, 25, 27, 29]. Yao and Yao [29], were one of the first to consider intersection searching involving curved objects. They used geometric

*A preliminary version of this paper appears in *Proc. Fourth Scandanavian Workshop on Algorithm Theory, 1994*, Springer Verlag Lecture Notes on Computer Science, Volume 824, pages 183–194.

†This research was supported in part by NSF grant CCR-92-00270.

‡Delsoft (India) Pvt. Ltd. E-mail: pgupta@delsoft.com . This work was done while the author was at the University of Minnesota, Minneapolis.

§Department of Computer Science & Engineering, University of Minnesota, Minneapolis, MN 55455, U.S.A. E-mail: janardan@cs.umn.edu.

¶Department of Computer Science, University of Magdeburg, D-39106 Magdeburg, Germany. E-mail: michiel@isg.cs.uni-magdeburg.de . This work was done while the author was at the Max-Planck-Institut für Informatik, Saarbrücken.

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

Table 1: *Summary of results for intersection reporting on circular objects with a circular query object.*

transforms to linearize nonlinear queries. Recently in [4], Agarwal *et al.* have considered searching on curved objects (e.g., circles, disks, circular arcs, Jordan arcs) with linear query objects (e.g., lines, line segments, halfspaces, rays). In [27] (resp. [25]), the problem of reporting the intersection of a query line or line segment (resp. a point) with a set of disks is considered. In [1], algorithms are given for range searching on point sets in $\mathbb{R}^d$ with ranges defined by a constant number of bounded-degree polynomials.

1.1 Overview of results

In this paper, we make further contributions to intersection searching in the curved setting by presenting efficient solutions to two broad classes of problems.[1]

In the first class of problems, we wish to preprocess a set S of n circular objects so that the ones that are intersected by a circular query object can be reported efficiently. This problem was left open in [4]. Table 1 summarizes our results.[2] 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. (Techniques for obtaining such intermediate trade-offs can be found in [6].)

Next, we consider a generalization of the standard intersection searching problems involving circular objects: Here S consists of n linear or circular objects and the objects come aggregated in

[1] In this paper, the term "circular" means objects such as circles, disks, circular arcs, annuli, d-balls, and d-spheres. (A *d-sphere* is the boundary of a closed d-ball.) Throughout it is assumed that the input objects can have different radii. Moreover, the radius of the query object is not known beforehand; it is specified as a part of the query.

[2] 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 $\epsilon \to 0$.

Input objects	*Query object*	*Space*	*Query time*
Disks	Point	$n \log n$	$\log n + i \log^2 n$
	Line	$n^2/\log^{1-\epsilon} n$	
	Disk	$n^2 \log n$	
Line Segments			
Lines			
Points	Annulus	$n^3/\log n$	

Table 2: *Summary of results for generalized reporting problems involving circular objects.*

Input objects	*Query object*	*Space*	*Query time*
Points	Annulus	$n^4 \log^2 n$	$\log^2 n$
Disk	Disk	$n^4 \log n$	$\log n$

Table 3: *Summary of results for generalized counting problems involving circular objects.*

disjoint groups. If we assign each object a color, according to the group it belongs to, then our goal is to report the distinct colors of the objects intersected by a query object (rather than reporting all the intersected objects as in the case of the standard problem). Such *generalized intersection searching* problems have been considered recently in [3, 11, 12, 13] in the context of linear input and query objects. The challenge in these problems is to obtain solutions whose query times are sensitive to the output size, namely the number, i, of distinct colors intersected (not the number, k, of intersected objects, which can be much larger). For the problems considered in this paper, we obtain query times of the form $O(f(n) + i \cdot g(n))$, where $f(n)$ and $g(n)$ are polylogarithmic (Table 2). We also consider the counting version, where we obtain polylogarithmic query time. Table 3 summarizes our result. Previously, efficient randomized algorithms have been given in [2] for counting intersections in sets of circles and circular arcs, with time bounds $O(n^{3/2+\epsilon})$ and $O(n^{5/3+\epsilon})$, respectively.

1.2 Overview of techniques

Our results are based on three main approaches. The first approach transforms the circular problem at hand to a simplex range searching problem and solves the latter by suitably composing together known techniques such as partition trees, cutting trees and geometric transforms. This approach is not new *per se*; for instance it has been used in [1, 29] for other problems. However, what makes this part of our work interesting is that the characterization of intersection and the appropriate transform(s) to use are not always apparent; indeed, in some cases, we need to apply successively

more than one transform (moreover, in the correct order). In addition to using known transforms we also introduce some new ones.

Our second approach, which we apply to the generalized reporting problems, is as follows: We start out by assuming that all colors in S are intersected and then progressively refine this estimate by identifying those colors that cannot possibly be intersected. To do this efficiently, we store the distinct colors of S in a suitable data structure and perform *standard intersection detection* tests on appropriately-defined subsets of S.

For the more difficult generalized counting problems, we use a different approach: We partition the plane into suitable regions within which the answer to a query is invariant, order the regions suitably, and apply the technique of persistence [8]. However the querying strategy is quite subtle because the answer to the query is not available explicitly but rather is embedded implicitly 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 counting problem, which can be solved efficiently.

Thus the contribution of the paper is a uniform framework to solve efficiently a variety of intersection searching problems involving circular objects.

The rest of the paper is organized as follows: In Section 2 we discuss the standard (i.e., uncolored) query problems and in Section 3 we consider the generalized problems. We conclude in Section 4.

2 Standard intersection searching with circular objects

2.1 Querying d-balls with a d-ball

Let $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$ be a collection of closed d-balls in $\mathbb{R}^d$, $d \geq 2$, and let Q be a query d-ball. Our goal is to report or count efficiently the d-balls in $\mathcal{B}$ that are intersected by Q. We use the coordinates $x_1, x_2, \ldots, x_d$ in $\mathbb{R}^d$. 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. 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 transform, τ, which maps B_i to a point in $\mathbb{R}^{d+2}$, 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 transform, α, which maps Q to a hyperplane in $\mathbb{R}^{d+2}$, as follows:

$$\alpha(Q) : x_{d+2} = 2q_1 x_1 + \cdots + 2q_d x_d + 2r_Q x_{d+1} - q_1^2 - \cdots - q_d^2 + r_Q^2. \tag{3}$$

We claim that Q intersects B_i iff $\tau(B_i) \in \alpha(Q)^-$, where $\alpha(Q)^-$ is the closed halfspace lying below $\alpha(Q)$. To see this, note that by Equations (1)–(3), it is sufficient to show that

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

This can be verified by straightforward algebraic manipulation.

Thus, we have transformed our d-ball problem in $\mathbb{R}^d$ to halfspace searching in $\mathbb{R}^{d+2}$. For the reporting problem, we can use the halfspace range searching structure of Matoušek [17] (resp. Clarkson and Shor [5]), which, in $\mathbb{R}^p$, uses $O(n \log\log n)$ (resp. $O(n^{\lfloor p/2 \rfloor+\epsilon})$) space and has a query time of $O(n^{1-1/\lfloor p/2 \rfloor}(\log n)^{O(1)}+k)$ (resp. $O(\log n+k)$), where k is the output size. For the counting problem, we can use the structure in [16] (resp. [18]), which uses $O(n)$ (resp. $O(n^p/\log^p n)$) space and has a query time of $O(n^{1-1/p}(\log\log n)^{O(1)})$ (resp. $O(\log n)$). Substituting $p = d+2$, we conclude:

Theorem 2.1 *Let $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$ be a collection of closed d-balls in $\mathbb{R}^d$, $d \geq 2$. $\mathcal{B}$ can be preprocessed into a data structure of size $O(n \log\log n)$ (resp. $O(n^{\lfloor d/2 \rfloor+1+\epsilon})$) such that the k d-balls that are intersected by a query d-ball, Q, can be reported in time $O(n^{1-1/(\lfloor d/2 \rfloor+1)}(\log n)^{O(1)}+k)$ (resp. $O(\log n+k)$). The counting problem can be solved in $O(n)$ (resp. $O(n^{d+2}/\log^{d+2} n)$) space and a query time of $O(n^{1-1/(d+2)}(\log\log n)^{O(1)})$ (resp. $O(\log n)$).* □

2.2 Review of a query composition technique

In Sections 2.3–2.5 below, we will express intersection conditions as the conjunction of $h > 1$ halfspace range queries, where $h = O(1)$. (This is unlike Section 2.1, where we used just one halfspace.) Towards this end, we review a useful query composition result due to van Kreveld [26], which we will use often in Sections 2.3–2.5. (This result is based on multi-level range searching structures [6, 7, 16].)[3]

Let $\mathcal{S}$ be a set of n geometric objects. Let D be a data structure for some query problem on $\mathcal{S}$, with space and query time bounds $O(f(n))$ and $O(g(n))$, respectively. Suppose that we now wish to answer queries not on the entire set $\mathcal{S}$ but on a subset $\mathcal{S}'$ of $\mathcal{S}$, where $\mathcal{S}'$ is specified by putting $\mathcal{S}$ in 1–1 correspondence with a set $\mathcal{P}$ of points in $\mathbb{R}^d$ and letting $\mathcal{S}'$ correspond to the subset $\mathcal{P}'$ of $\mathcal{P}$ lying in a query simplex. (In [26], this technique is called *simplex composition* on $\mathcal{P}$ to D) The following theorem states how fast the query problem on $\mathcal{S}'$ can be solved.

Theorem 2.2 [26] *Let $\mathcal{S}$, D, and $\mathcal{P}$ be as above. For an arbitrarily small constant $\epsilon > 0$, simplex composition on $\mathcal{P}$ to D yields a data structure (i) of size $O(n^\epsilon(n^d + f(n)))$ and query time*

[3] We note that slightly more efficient compositions than the one in [26] are known. See for instance [18]. We have chosen the method of [26] because it is relatively simpler and serves to illustrate the ideas behind our approach.

$O(\log n + g(n))$, or (ii) of size $O(n + f(n))$ and query time $O(n^\epsilon(n^{1-1/d} + g(n)))$, or (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 $f(n)/n$ is nondecreasing and $g(n)/n$ is nonincreasing. For the reporting problem, the output size, k, must be included in the query time as an additive term. $\square$

In our application, the simplex will always be a halfspace. As mentioned before, we express intersection conditions as the conjunction of h halfspace queries, for some constant h. Given the h halfspaces, we proceed as follows: We design an initial data structure D. Then we apply Theorem 2.2, with one of the h halfspaces. This gives a new structure D' to which we apply Theorem 2.2 using a second halfspace and so on. Since $h = O(1)$, the space and query time bounds of the resulting structure are asymptotically the same as the ones given in Theorem 2.2.

We illustrate the above idea with the following simple example: Let $\mathcal{S}$ be a set of n vertical line segments in $\mathbb{R}^2$. We wish to construct a data structure for the following query problem on $\mathcal{S}$: "Given a query line ℓ, 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. 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 compositions: In the first, we associate $\mathcal{S}$ with the set $\mathcal{P}$ of upper endpoints and use ℓ^+; in the second, we associate $\mathcal{S}$ with the set $\mathcal{P}$ of lower endpoints and use ℓ^-. We then apply these two compositions successively using Theorem 2.2. This gives $O(n^{2+\epsilon})$ space and $O(\log n)$ query time at one extreme and $O(n)$ space and $O(n^{1/2+\epsilon})$ query time at the other extreme for the counting problem. For the reporting problem, the query times are $O(\log n + k)$ and $O(n^{1/2+\epsilon} + k)$, respectively.

2.3 Querying d-spheres with a d-sphere

Let $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ be a collection of d-spheres, where S_i has radius r_i and center $C_i = (b_{i1}, b_{i2}, \ldots, b_{id})$, $1 \leq i \leq n$. Let $C_Q = (q_1, q_2, \ldots, q_d)$ be the center of a query d-sphere Q and r_Q its radius. Q intersects S_i iff the Euclidean distance between C_i and C_Q is at most $r_i + r_Q$ and at least $|r_i - r_Q|$, i.e.,

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

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

$$\beta(Q) : x_{d+2} = 2q_1x_1 + \cdots + 2q_dx_d - 2r_Qx_{d+1} - q_1^2 - \cdots - q_d^2 + r_Q^2. \tag{4}$$

In addition, consider the transforms τ and α of Section 2.1, 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 halfspace above $\beta(Q)$. The points $\tau(S_i)$ satisfying the above condition (which is

the conjunction of two halfspace queries) can be found via two halfspace compositions in $\mathbb{R}^{d+2}$. We now apply Theorem 2.2, taking the structure D to be a linked list of the points $\{\tau(S_i) : 1 \leq i \leq n\}$.

Theorem 2.3 *Let $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ be a collection of d-spheres in $\mathbb{R}^d$, $d \geq 2$. $\mathcal{S}$ can be preprocessed into a data structure of size $O(n)$ (resp. $O(n^{d+2+\epsilon})$) such that the k d-spheres that are intersected by a query d-sphere, Q, can be reported in time $O(n^{1-1/(d+2)+\epsilon} + k)$ (resp. $O(\log n + k)$). For the counting problem, the query times are $O(n^{1-1/(d+2)+\epsilon})$ and $O(\log n)$ respectively.* $\square$

Remark 2.1 We note that for $d = 2$, the $O(n^4/\log^4 n)$ bound of Theorem 2.1 and the $O(n^{4+\epsilon})$ space bound of Theorem 2.3 can be improved to $O(n^{3+\epsilon})$, without affecting the respective query times, by extending a result of Agarwal and Sharir [2].

2.4 Querying circular arcs with a circle

We show how to preprocess a set S of n circular arcs so that the arcs that are intersected by any query circle C can be reported quickly.

Let γ be a circular arc. Let $circ(\gamma)$ denote the circle that γ is a part of, and let, $center(\gamma)$ and $radius(\gamma)$ denote, respectively, the center and the radius of $circ(\gamma)$. Wlog assume that the arcs in S are x-monotone since any circular arc can be decomposed into at most three x-monotone pieces. Let $l(\gamma)$ and $r(\gamma)$ be γ's left and right endpoints, respectively. Let $disk(C)$ be the closed disk bounded by C. Let ℓ_l^{in} (resp. ℓ_l^{out}) 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 [22] and is illustrated in Figure 1.

Lemma 2.1 [22] *A circle C and an x-monotone arc γ intersect iff one of the following is true: (i) C separates the endpoints of γ or (ii) C intersects* $\mathrm{circ}(\gamma)$ *and either (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* $\mathrm{center}(C) \in \ell_l^{out} \cap \ell_r^{out}$. $\square$

An arc γ satisfies condition (i) if and only if $chord(\gamma)$ has one endpoint in the interior of $disk(C)$ and the other endpoint outside $disk(C)$. To report such arcs, we apply the well-known "lifting transform": Recall that this transform maps a point p in $\mathbb{R}^2$ to a point $\psi(p)$ in $\mathbb{R}^3$ and a circle C in $\mathbb{R}^2$ to a plane $\varphi(C)$ in $\mathbb{R}^3$ such that $\psi(p)$ lies below (resp. on, above) $\varphi(C)$ iff p lies inside (resp. on, outside) C. (See [23, pp. 236–238], [9, p. 304] or [4] for details.) Thus by applying ψ to the endpoints of γ and φ to C we convert the problem at hand to one of reporting those line segments $\overline{\psi(l(\gamma))\psi(r(\gamma))}$ in $\mathbb{R}^3$ 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 2.2).

Let us now consider how to report arcs that satisfy condition (ii). First consider the first part of condition (ii)(a), namely "$l(\gamma) \notin disk(C)$ and $r(\gamma) \notin disk(C)$". Using the transforms ψ and φ, this

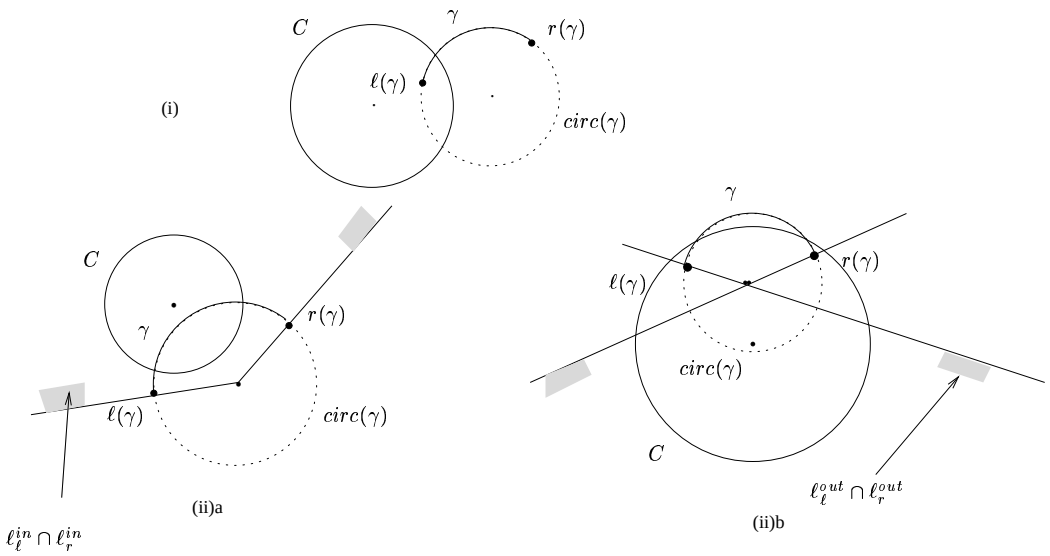

Figure 1: *Illustrating Lemma 2.1*

becomes "$\overline{\psi(l(\gamma))\psi(r(\gamma))}$ lies above $\varphi(C)$ in $\mathbb{R}^3$". This in turn can be expressed as two halfspace compositions in $\mathbb{R}^3$.

To handle the second part of condition (ii)(a), namely "$center(C) \in \ell_l^{in} \cap \ell_r^{in}$", we use the well-known duality transform $\mathcal{F}$: Recall that $\mathcal{F}$ maps a point p in $\mathbb{R}^2$ (resp. $\mathbb{R}^3$) to a non-vertical line (resp. plane) $\mathcal{F}(p)$ and vice-versa. (See [9, p. 13] for details.) Note that the wedge $\ell_l^{in} \cap \ell_r^{in}$ can be written as $t_1^+ \cap t_2^+$ or $t_1^- \cap t_2^-$ or $t_1^+ \cap t_2^-$ for some lines t_1 and t_2. Thus, by applying the transform $\mathcal{F}$ to $center(C)$, t_1, and t_2, the condition "$center(C) \in \ell_l^{in} \cap \ell_r^{in}$", becomes "$\mathcal{F}(center(C))$ 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 $\mathbb{R}^2$.

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

We are now ready to apply Theorem 2.2 to handle condition (ii). As the structure D of that theorem, we take the $O(n)$-space structure of Theorem 2.3, for $d = 2$, to handle the condition "C intersects $circ(\gamma)$". 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 2.1. 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 2.2.

Theorem 2.4 *A set S of n circular arcs in the plane 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 query circle C can be reported in time $O(n^{3/4+\epsilon}+k)$ (resp. $O(\log n + k)$).* $\Box$

Remark 2.2 We note that the $O(n^{3+\epsilon})$-space and $O(\log n + k)$-query time result of Theorem 2.4 can also be obtained by a different approach using a result in [2].

2.5 Querying circles with a circular arc

Here we consider the inverse problem where S consists of n circles and the query is a circular arc γ. Again, we may assume that γ is x-monotone. Lemma 2.1 still applies; however, the approach is somewhat different since the roles of arcs and circles are now reversed.

To report the circles that separate the endpoints of γ (condition (i) of Lemma 2.1), we map each circle C to the point $\mathcal{F}(\varphi(C))$ in $\mathbb{R}^3$ 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 condition (i) has been transformed to one of reporting among a set of points in $\mathbb{R}^3$ 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 $\mathbb{R}^3$.

Let us now consider condition (ii)(a) of Lemma 2.1. 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 $\mathbb{R}^3$. Moreover, the condition "$\mathit{center}(C) \in \ell_l^{in} \cap \ell_r^{in}$" can be expressed immediately as two halfplane compositions in $\mathbb{R}^2$. A similar discussion applies for condition (ii)(b).

We can now apply Theorem 2.2 to report the segments that satisfy condition (ii). We choose the structure D as in Section 2.4 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 2.5 *A set S of n circles in the plane 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 query circular arc γ can be reported in time $O(n^{3/4+\epsilon}+k)$ (resp. $O(\log n + k)$).* $\Box$

3 Generalized intersection searching with circular objects

We now turn to the second class of problems considered in this paper: Each object 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 can be counted or reported efficiently.

3.1 A general technique for reporting problems

Let S be a set of n colored geometric objects and let q be any query object. In preprocessing, we store the distinct colors in S at the leaves of a balanced binary tree CT (in no particular order). For any node v of CT, let $C(v)$ be the colors stored in the leaves of v's subtree and let $S(v)$ be the objects of S colored with the colors in $C(v)$. At v, we store a data structure $DET(v)$ to solve the following *standard detection* problem on $S(v)$: "Decide whether or not q intersects any object of $S(v)$." $DET(v)$ returns "true" iff there is an intersection.

To answer a generalized reporting query on S, we do a depth-first search of CT and query $DET(v)$ with q at each node v visited. If v is a non-leaf node then we continue searching below v iff the query returns "true"; if v is a leaf, then we output the color stored there iff the query returns "true".

Theorem 3.1 *Assume that a set of n geometric objects can be stored in a data structure of size $M(n)$ such that it can be detected in $f(n)$ time whether or not a query object intersects any of the n objects. Assume that M is convex*[4] *and that f is non-decreasing. Then a set S of n colored geometric objects can be preprocessed into a data structure of size $O(M(n)\log n)$ such that the i distinct colors of the objects in S that are intersected by a query object q can be reported in time $O(f(n) + i \cdot f(n)\log n)$.*

Proof We argue that a color c is reported iff there is a c-colored object in S intersecting q. Suppose that c is reported. This implies that a leaf v is reached in the search such that v stores c and the query on $DET(v)$ returns "true". Thus, some object in $S(v)$ intersects q. Since v is a leaf, all objects in $S(v)$ have the same color c and the claim follows.

For the converse, suppose that q intersects a c-colored object p. Let v be the leaf storing c. Thus, $p \in S(v')$ for every node v' on the root-to-v path in CT. Thus, for each v', the query on $DET(v')$ will return "true", which implies that v will be visited and c will be output.

If $v_1, v_2, \ldots, v_r$ are the nodes at any level, then the total space used by CT at that level is $\sum_{i=1}^{r} M(|S(v_i)|) = O(M(n))$, since $\sum_{i=1}^{r} |S(v_i)| = n$ and M is a convex function. Now since there are $O(\log n)$ levels, the overall space is $O(M(n)\log n)$. The query time can be upper-bounded as follows: If $i = 0$, then the query on $DET(\text{root})$ returns "false" and we abandon the search at the root itself; in this case, the query time is just $O(f(n))$. Suppose that $i \neq 0$. Call a visited node v *fruitful* if the query on $DET(v)$ returns "true" and *fruitless* otherwise. Each fruitful node can be charged to some color in its subtree that gets reported. Since the number of times any reported color can be charged is $O(\log n)$ (the height of CT) and since i colors are reported, the number

[4] i.e. We assume that for non-negative integers a and b, $M(a) + M(b) \leq M(a+b)$.

of fruitful nodes is $O(i \log n)$. Since each fruitless node has a fruitful parent and CT is a binary tree, it follows that there are only $O(i \log n)$ fruitless nodes. Hence the number of nodes visited by the search is $O(i \log n)$. At each such node, v, we spend time $f(|S(v)|)$, which is $O(f(n))$ since $|S(v)| \leq n$ and f is non-decreasing. Thus the total time spent in doing queries at the visited nodes is $O(i \cdot f(n) \log n)$. The claimed query time follows. $\square$

Theorem 3.1 generalizes a method used in [12] for the generalized halfspace range searching problem in $d \geq 4$ dimensions.

3.2 Some applications of the general technique

First we consider the problem where the input is a set of colored disks and the query is a point. We need to solve the following (standard) problem: "Preprocess a set of n disks such that it can be detected if a query point lies in the union of the disks". Using the transforms φ and ψ defined in Section 2.4, we map each disk D to the halfspace $\varphi(D)^-$ and the query q to the point $\psi(q)$ in $\mathbb{R}^3$. The complement of the union of the disks maps to an upper convex polytope. It is then sufficient to check if $\psi(q)$ lies in this polytope. This problem can be reduced to planar point location via stereographic projection of the polytope (see [15, page 605]) and hence can be solved in $O(n)$ space and $O(\log n)$ query time [9]. We now apply Theorem 3.1 to get:

Theorem 3.2 *A set S of n colored disks in $\mathbb{R}^2$ can be stored in a data structure of size $O(n \log n)$ so that the i distinct colors of the disks stabbed by any query point can be reported in time $O(\log n + i \log^2 n)$.* $\square$

Next we consider the problem where the input is a set of colored disks and the query is a line. Here our standard problem is "Preprocess a set of n disks such that it can be detected if a query line intersects any disk". In [1], this is reduced to a halfspace emptiness query in $\mathbb{R}^5$. Using the data structure of [21], this is solvable in $O(n^2/(\log n)^{2-\epsilon})$ space and a query time of $O(\log n)$. We now apply Theorem 3.1 and conclude:

Theorem 3.3 *A set S of n colored disks in $\mathbb{R}^2$ can be stored in a data structure occupying $O(n^2/(\log n)^{1-\epsilon})$ space so that the i distinct colors of the disks intersected by any query line can be reported in time $O(\log n + i \log^2 n)$.* $\square$

Suppose that the input set S is a set of colored lines (resp. line segments), and the query is a disk. The detection problem is: "Preprocess n lines (resp. line segments) such that it can be decided if a query disk intersects any line (resp. line segment)". To build the detection structure, we first compute the arrangement $\mathcal{A}$ of the input lines (resp. line segments). For any face f, we compute the Voronoi diagram of the edges of $\mathcal{A}$ on the boundary of f, using the sweepline algorithm of Fortune [10]. We also preprocess $\mathcal{A}$ for fast planar point location [9, 23]. Given a query disk Q with center C_Q and radius r_Q, we locate the face f of the arrangement that contains C_Q. Next,

we use f's Voronoi diagram to find the edge of f that is closest to C_Q. In this way, we know the minimal distance of C_Q to any line (resp. line segment) of S. The query disk intersects any line (resp. line segment) of S iff this distance is at most r_Q. This solution uses $O(n^2)$ space and has $O(\log n)$ query time. We now apply Theorem 3.1 to get a data structure of size $O(n^2 \log n)$ and query time $O(\log n + i \log^2 n)$. If S is a set of colored disks, we follow the same approach. The edges on the boundary of a face are circular arcs. We use the algorithm of Yap [28], to compute the Voronoi diagram in this case.

Theorem 3.4 *A set S of n colored lines (resp. line segments, disks) in $\mathbb{R}^2$ can be stored in a data structure of size $O(n^2 \log n)$, so that the i distinct colors of the lines (resp. line segments, disks) intersected by any query disk can be reported in time $O(\log n + i \log^2 n)$.* □

When S is a set of colored points and the query is an annulus, using transforms φ and ψ in Section 2.4, we get the following detection problem: "Given n points, is there a point in between two parallel query planes in $\mathbb{R}^3$?" By duality, this becomes "Does a vertical line segment in $\mathbb{R}^3$ intersect any of a set of n planes?"

In [19] it is shown that a set of n hyperplanes in $\mathbb{R}^d$ can be stored in $O(n^d / \log^{d-1} n)$ space so that a vertical ray shooting query can be answered in $O(\log n)$ time. We can use this to solve our detection problem in $M(n) = O(n^3 / \log^2 n)$ space and $O(\log n)$ time. We conclude:

Theorem 3.5 *A set S of n colored points in $\mathbb{R}^2$ can be stored in a data structure of size $O(n^3 / \log n)$, so that the i distinct colors of the points intersected by any query annulus can be reported in time $O(\log n + i \log^2 n)$.* □

3.3 Generalized intersection counting problems

We briefly review the technique of persistence which we will use in this section.

3.3.1 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 [8], 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". 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. (Here we assume that $m(n) = \Omega(n)$, which will be the case in our application below.)

3.3.2 Generalized annulus range counting

Let S be a set of n colored sites (points) in the plane. Let $Ann(q, r_1, r_2)$ be the query annulus, with radius r_1 and r_2 ($r_1 \leq r_2$) and center q. We wish to count the distinct colors of the sites contained in $Ann(q, r_1, r_2)$. We begin by partitioning the plane into regions that satisfy a certain distance invariance property, as specified in the following lemma:

Lemma 3.1 *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. $d(p, a) \geq d(p, b)$) iff $p \in P_a(a, b)$ (resp. $P_b(a, b)$), where $d(\cdot, \cdot)$ is the Euclidean distance function.

Face $f \in \mathcal{A}$ 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. Suppose that we sort the sites of S by non-decreasing distance from p_1 and, independently, sort them by non-decreasing distance from p_2. It is clear from the above discussion that the outcome of any distance comparison in the second sort is identical to that in the first sort (i.e. $d(p_2, a) \leq d(p_2, b)$ iff $d(p_1, a) \leq d(p_1, b)$ for any sites a and b). The claim follows. □

We preprocess $\mathcal{A}$ for fast planar point location [9]. Given $Ann(q, r_1, r_2)$ 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 3.1.

By definition of $\mathcal{E}(f)$, the sites at distance at least r_1 and at most r_2 from q (i.e., the ones 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, our problem becomes: *"Given a sequence of colored integers $1, 2, \ldots, n$ on the real line (namely, the indices of the sites in $\mathcal{E}(f)$), count the i distinct colors of the points lying in the interval $[j, k]$."* This is just an instance of the *generalized 1-dimensional range counting* problem considered in [11]. There a dynamic solution was given for this problem, which uses $O(n \log n)$ space, has a query time of $O(\log^2 n)$, and undergoes $O(\log^2 n)$ memory modifications per update. Let D_f be the structure for face f. Thus, given j and k, we can answer our problem in $O(\log^2 n)$ time using D_f.

But how do we find j and k efficiently? We find j (and symmetrically k) in $O(\log n)$ time using binary search as follows: Note that s_j is the site in $\mathcal{E}(f)$ with the smallest index j such that $d(q, s_j) \geq r_1$. Let T_f be a red-black tree 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.

Clearly, the total space is $O(n^5 \log n)$. We can reduce this to $O(n^4 \log^2 n)$ by applying persistence to the faces of $\mathcal{A}$. Lemma 3.2 below provides a suitable ordering of the faces for this purpose, based on an Eulerian walk of an appropriately defined dual graph of $\mathcal{A}$. (Such an ordering of faces in an arrangement has been used in the past, for instance in [2].)

Lemma 3.2 *Let P be any 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 includes each face of P at least once.* □

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. Using Lemma 3.2, we can prove:

Lemma 3.3 *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 3.2, 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 3.3 to store all the T_f's and D_f's persistently. 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 3.3, there are $O(n^4)$ updates. Moreover each update causes $O(\log^2 n)$ memory modifications ($O(1)$ for T_f and $O(\log^2 n)$ for D_f).

Theorem 3.6 *A set S of n colored points in the plane can be preprocessed into a data structure of size $O(n^4 \log^2 n)$ such that the distinct colors of the points lying inside any query annulus can be counted in $O(\log^2 n)$ time.* □

3.3.3 Generalized intersection counting on disks

Here $S = \{D_1, D_2, \ldots, D_n\}$ is a collection of closed disks in the plane. The query Q is a closed 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 $d(C_Q, C_i) \leq r_Q + r_i$. 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 [24] that L_{ij} is a curve of degree at most two. 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 (halflines, line segments, or subarcs of hyperbolic arcs).

Using a proof similar to Lemma 3.1 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. Wlog, let $\mathcal{E}(f) = C_1, C_2, \ldots, C_n$ denote this ordering for f. Assume that $C_Q \in f$. Then, the centers C_i of the disks D_i that are intersected by Q satisfy $0 \leq d(C_Q, C_i) - r_i \leq r_Q$. These C_i form a prefix $C_1, C_2, \ldots, C_k$ of $\mathcal{E}(f)$. As in the previous section, k can be found in $O(\log n)$ time by doing a binary search on $\mathcal{E}(f)$ using r_Q. Thus our generalized counting problem reduces to doing a generalized 1-dimensional range counting query on $1, 2, \ldots, n$, with the query interval $(-\infty, k]$. In [11], this problem is solved in $O(n)$ space, $O(\log n)$ query time, and $O(\log n)$ memory modifications per update. Together with the persistence-based approach for space reduction, this immediately gives us the following:

Theorem 3.7 *A set S of n disks in the plane can be preprocessed into a data structure of size $O(n^4 \log n)$ such that the distinct colors of the disks that are intersected by a query disk can be counted in $O(\log n)$ time.* □

4 Conclusion

We have investigated the problem of intersection searching involving circular objects and have presented efficient solutions for two broad classes of problems, namely (i) for circular input objects and circular query objects, and (ii) for a generalized version where the input objects are colored.

We close by mentioning an interesting direction for further research: 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?

References

[1] Agarwal, P.K. and Matoušek, J. (1994). On range searching with semialgebraic sets. *Discrete & Computational Geometry*, **11**, 393–418.

[2] Agarwal, P.K. and Sharir, M. (1991). Counting circular arc intersections. *Proceedings of the 7th Annual Symposium on Computational Geometry*, 10–20.

[3] Agarwal, P.K. and van Kreveld, M. (1996). Polygon and connected component intersection searching. *Algorithmica*, **15**, 626–660.

[4] Agarwal, P.K., van Kreveld, M. and Overmars, M. (1993). Intersection queries for curved objects. *Journal of Algorithms*, **15**, 229–266.

[5] Clarkson, K.L. and Shor, P.W. (1989). Applications of random sampling in computational geometry – II. *Discrete & Computational Geometry*, **4**, 387–421.

[6] Chazelle, B., Sharir, M. and Welzl, E. (1992). Quasi-optimal upper bounds for simplex range searching and new zone theorems. *Algorithmica*, **8**, 407–429.

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

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

[9] Edelsbrunner, H. (1987). *Algorithms in Combinatorial Geometry*. Springer, Berlin.

[10] Fortune, S. (1987). A sweepline algorithm for Voronoi diagrams. *Algorithmica*, **2**, 153–174.

[11] Gupta, P., Janardan, R. and Smid, M. (1995). Further results on generalized intersection searching problems: counting, reporting and dynamization. *Journal of Algorithms*, **19**, 282–317.

[12] Gupta, P., Janardan, R. and Smid, M. (1996). Algorithms for generalized halfspace range searching and other intersection searching problems. *Computational Geometry: Theory and Applications*, **5**, 321–340.

[13] Janardan, R. and Lopez, M. (1993). Generalized intersection searching problems. *International Journal on Computational Geometry & Applications*, **3**, 39–69.

[14] Kirkpatrick, D.G. (1983). Optimal search in planar subdivisions. *SIAM Journal on Computing*, **12**, 28–35.

[15] Lee, D.T. and Preparata, F.P. (1977). Location of a point in a planar subdivision and its applications. *SIAM Journal on Computing*, **6**, 594–606.

[16] Matoušek, J. (1992). Efficient partition trees. *Discrete & Computational Geometry*, **8**, 315–334.

[17] Matoušek, J. (1992). Reporting points in halfspaces. *Computational Geometry: Theory and Applications*, **2**, 169–186.

[18] Matoušek, J. (1993). Range searching with efficient hierarchical cuttings. *Discrete & Computational Geometry*, **10**, 157–182.

[19] Matoušek, J. (1993). On vertical ray-shooting in arrangements. *Computational Geometry: Theory and Applications*, **2**, 279–285.

[20] Matoušek, J. (1994). Geometric range searching. *ACM Computing Surveys*, **26**, 421–461.

[21] Matoušek, J. and Schwarzkopf, O. (1993). On ray shooting in convex polytopes. *Discrete & Computational Geometry*, **10**, 215–232.

[22] Pellegrini, M. (1992). A new algorithm for counting circular arc intersections. Technical Report 92-010, International Computer Science Institute, Berkeley.

[23] Preparata, F.P. and Shamos, M.I. (1988). *Computational geometry – an Introduction*. Springer, Berlin.

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

[25] Sharir, M. (1991). The k-set problem for arrangements of curves and surfaces. *Discrete and Computational Geometry*, **6**, 593–613.

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

[27] van Kreveld, M., Overmars, M. and Agarwal, P.K. (1992). Intersection queries in sets of disks. *BIT*, **32**, 268–279.

[28] Yap, C.K. (1987). The Voronoi diagram of a set of simple curve segments. *Discrete & Computational Geometry*, **2**, 365–393.

[29] Yao, A.C.and Yao, F.F. (1985). A general approach to d-dimensional geometric queries. *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, 163–174.