MAX-PLANCK-INSTITUT
FÜR
INFORMATIK

Efficient algorithms for generalized intersection searching on non-iso-oriented objects

Prosenjit Gupta   Ravi Janardan   Michiel Smid

MPI–I–93–166   December 1993

INFORMATIK

Im Stadtwald
66123 Saarbrücken
Germany

Efficient algorithms for generalized intersection searching on non-iso-oriented objects

Prosenjit Gupta Ravi Janardan Michiel Smid

MPI–I–93–166 December 1993

Efficient Algorithms for Generalized Intersection Searching on Non-Iso-Oriented Objects

Prosenjit Gupta* Ravi Janardan* Michiel Smid†

December 2, 1993

Abstract

In a generalized intersection searching problem, a set S of colored geometric objects is to be preprocessed so that, given a query object q, the distinct colors of the objects of S that are intersected by q can be reported or counted efficiently. These problems generalize the well-studied standard intersection searching problems and are rich in applications. Unfortunately, the solutions known for the standard problems do not yield efficient solutions to the generalized problems. Recently, efficient solutions have been given for generalized problems where the input and query objects are iso-oriented (i.e., axes-parallel) or where the color classes satisfy additional properties (e.g., connectedness). In this paper, efficient algorithms are given for several generalized problems involving non-iso-oriented objects. These problems include: generalized halfspace range searching in $\mathcal{R}^d$, for any fixed $d \geq 2$, and segment intersection searching, triangle stabbing, and triangle range searching in $\mathcal{R}^2$. The techniques used include: computing suitable sparse representations of the input, persistent data structures, and filtering search.

Keywords: Computational geometry, data structures, filtering search, geometric duality, intersection searching, persistence.

*Department of Computer Science, University of Minnesota, Minneapolis, MN 55455, U.S.A. Email: {pgupta, janardan}@cs.umn.edu. The research of these authors was supported in part by NSF grant CCR–92–00270. Portions of this work were done while RJ was visiting MS at the Max-Planck-Institut für Informatik, in Saarbrücken, Germany, in July 1993. RJ would like to thank the MPI for its generous support.

†Max-Planck-Institut für Informatik, D-66123 Saarbrücken, Germany. Email: michiel@mpi-sb.mpg.de. This author was supported by the ESPRIT Basic Research Actions Program, under contract No. 7141 (project ALCOM II).

1 Introduction

Consider the following generic searching problem: Assume that we are given a set S of n geometric objects in $\mathcal{R}^d$. Moreover, assume that the objects come aggregated in disjoint groups, where the grouping is dictated by the underlying application. (The number of groups can range from 1 to $\Theta(n)$.) Our goal is to preprocess S into a data structure so that given any query object q, we can report or count efficiently the groups that are intersected by q. (We say that q *intersects a group* iff q intersects some object in the group.) Notice that we are not interested in reporting or counting the individual objects intersected by q as is the case in a *standard intersection searching* problem. Indeed the standard problem is a special case of the above formulation, where each group has cardinality 1. For this reason, we call our version a *generalized intersection searching* problem.

For our purposes, it will be convenient to associate with each group a different color and imagine that all the objects in the group have that color. Suppose that q intersects i groups. Then, we can restate our problem as: "Preprocess a set S of n colored geometric objects so that for any query object q, the i distinct colors of the objects that are intersected by q can be reported or counted efficiently." This is the version that we will consider throughout the paper.

Before going further, let us illustrate the usefulness of our generalized formulation with some applications: **(1)** In designing a VLSI chip, the wires (line segments) can be grouped naturally according to the circuits they belong to. A problem of interest to the designer is determining which circuits (rather than wires) become connected when a new wire is to be added. This is an instance of the *generalized segment intersection searching* problem. **(2)** Consider a collection of supply points (e.g., warehouses) of different types in $\mathcal{R}^2$. We would like to preprocess these points so that given a demand point q and a radius r, we can determine the types of supply points that are within distance r from q. By using a well-known "lifting" transformation, this problem can be transformed to an instance of *generalized halfspace range searching* in $\mathcal{R}^3$.

One approach to solving a generalized problem is to take advantage of known solutions for the corresponding standard problem. For example, to solve a generalized reporting problem, we can determine all the objects intersected by q (a standard problem) and then read off the distinct colors among these. (To avoid reporting a color more than once, we can use an array of colors to track already-reported colors. After the query, we can reset this in time proportional to i—the output size.) However, with this approach the query time can be very high since q could intersect $\Omega(n)$ objects but only $O(1)$ distinct colors. Thus, the challenge in the generalized reporting problem is to attain a query time that is sensitive to the output size. Typically we seek query times of the form $O(f(n) + i)$ or $O(f(n) + i \cdot \text{polylog}(n))$, where $f(n)$ is "small" (e.g., polylog(n) or n^p, where $0 < p < 1$). For a generalized counting problem, it is not even clear how one can use the solution for the corresponding standard problem (a mere count) to determine how many distinct colors are intersected. Nevertheless, we seek here query times of the form $O(f(n))$. Of course, for both the counting and the reporting problems, we seek solutions that are also as space-efficient as possible.

1.1 Previous work

While the standard problems have been investigated extensively (see, for example, [Cha86, CJ90, CJ92, CW89, AvKO93, Mat91a, Mat91b, Mat92a]), their generalized counterparts have been less studied. The generalized problems were first considered in [JL93], where efficient solutions were given for several problems defined on iso-oriented objects (i.e., the input and the query objects are axes-parallel). These include cases where the colored objects in S are, respectively, intervals, points in $\mathcal{R}^2$, orthogonal line segments in $\mathcal{R}^2$, and iso-oriented rectangles in $\mathcal{R}^d$, $d \geq 1$, and the query is, respectively, an interval, an iso-oriented rectangle, an orthogonal line segment, and a point in $\mathcal{R}^d$. A solution was also given for searching on arbitrary (non-intersecting) colored line segments with an arbitrary line segment. In [GJS93a], efficient solutions were given for the counting, reporting, and dynamic versions of some of the iso-oriented problems mentioned above. In [GJS93b], solutions were given for generalized problems involving circular and circle-like objects (among other results). In [AvK93], Agarwal and van Kreveld consider the problem of reporting the intersections of a query line segment with color classes consisting of line segments and satisfying the property that each color class is a simple polygon or a connected component.

1.2 Summary of results

In this paper, we present efficient solutions to several generalized intersection searching problems that are defined on non-iso-oriented objects. Specifically, we consider the following problems: generalized halfspace range searching in $\mathcal{R}^d$, for any fixed $d \geq 2$, and generalized segment intersection searching, triangle stabbing, and triangle range searching in $\mathcal{R}^2$. Our main results are summarized in Table 1. No results were known previously for any of these problems, with the exception of generalized segment intersection searching: For this problem, the following results were known: (a) $O(n^{1+\epsilon})$ space and $O(n^{1/2+\epsilon} + i)$ query time when each color class is a simple polygon or a connected component [AvK93]; (b) $O((n+\chi)^2 \log n)$ (resp.

#	*Input objects*	*Query object*	*Space*	*Query time*
1	Points in $\mathcal{R}^d$	Halfspace in $\mathcal{R}^d$	$d = 2$ $n \log n$	$\log^2 n + i$ $n^{1/2}$
			$d = 3$ $n \log^2 n$ $n^{2+\epsilon}$ $n \log n$	$n^{1/2+\epsilon} + i$ $\log^2 n + i$ $n^{2/3+\epsilon}$
			$d > 3$ $n^{\lfloor d/2 \rfloor + \epsilon}$	$\log n + i \log^2 n$
2	Line segs. in $\mathcal{R}^2$	Halfplane	$n \log n$	$\log^2 n + i$ $n^{1/2}$
		Vertical ray	$n\alpha(n) \log n$	$\log^2 n + i$ $(n\alpha(n))^{1/2}$
3	Lines in $\mathcal{R}^2$	Vertical line segment	$n^{2.5-\mu} \log n$ $n^2 \log n$	$n^\mu + i$ $\log n + i$
4	Line segs. of length $\geq$ a constant in unit square	Line	$n \log n$	$\log^2 n + i$
5	Line segs. in $\mathcal{R}^2$	Vertical line segment	$(n + \chi) \log n$	$\log n + i$
		Arbitrary line segment		$\log^2 n + i \log n$
6	Triangles	Point	$n^{3/2} \log n$	$\log^2 n + i$
	Fat-Wedges		$n \log n$	$\log^2 n + i$
7	Points	Fat-Triangle	$n \log^3 n$	$\log^4 n + i \log^2 n$

Table 1: Summary of main results for generalized intersection searching problems; additional results can be found in the text. Wherever the output size, i, is missing in a query time bound, it is a counting problem. A *fat-wedge* or *fat-triangle* is one where each interior angle is greater than or equal to a fixed constant. The meanings of the different symbols are as follows:

n	:	input size
i	:	output size (number of distinct colors intersected)
ϵ	:	arbitrarily small constant > 0
μ	:	tunable parameter, $0.5 < \mu < 1$
χ	:	number of pairwise-intersecting segments, $0 \leq \chi \leq \binom{n}{2}$
$\alpha(n)$	:	slow-growing inverse of Ackermann's function

$O((n+\chi)^2))$ space with a query time of $O(\log n + i)$ (resp. $O(\log^2 n + i)$) for general color classes, which follows from a result in [JL93]; and (c) $O(n^{1+\epsilon})$ space and $O((i+1)\sqrt{n}\log^{O(1)} n)$ query time for general classes, which is mentioned in [AvK93].[1]

Our results are based on a combination of several techniques: (1) Computing for each color class a sparse representation which captures essential information about the color class and allows us to reduce the generalized problem at hand to a standard problem. (2) The persistence-addition technique of Driscoll et al. [DSST89], which allows us to reduce a generalized query problem to a generalized query problem one dimension lower. (3) A version of filtering search which, in combination with persistence, yields a space-query time tradeoff. Moreover, when the input objects or query objects satisfy certain reasonable conditions (e.g., fatness), then we use further ideas to obtain very efficient solutions.

The rest of the paper is organized as follows: We consider Problems 1 and 2 of Table 1 in Section 2, Problems 3–5 in Section 3, Problem 6 in Section 4, and Problem 7 in Section 5. We conclude in Section 6 with some open problems.

2 Generalized halfspace range searching in $\mathcal{R}^d$

Let S be a set of n colored points in $\mathcal{R}^d$, for any fixed $d \geq 2$. We show how to preprocess S so that for any query hyperplane Q, the i distinct colors of the points lying in the halfspace Q^- (i.e., below Q) can be reported or counted efficiently.[2] For now assume that Q is non-vertical; we consider the case where Q is vertical in Section 2.3.

We denote the coordinate directions by $x_1, x_2, \ldots, x_d$. (For convenience, when discussing the solution in $\mathcal{R}^2$ specifically, we identify x_1 and x_2 with x and y; similarly, in $\mathcal{R}^3$, we use x, y, and z.) Let $\mathcal{F}$ denote the well-known point-hyperplane duality transform: If $p = (p_1, \ldots, p_d)$ is a point in $\mathcal{R}^d$, then $\mathcal{F}(p)$ is the hyperplane $x_d = p_1x_1 + \cdots + p_{d-1}x_{d-1} - p_d$. If $H : x_d = a_1x_1 + \cdots + a_{d-1}x_{d-1} + a_d$ is a (non-vertical) hyperplane in $\mathcal{R}^d$, then $\mathcal{F}(H)$ is the point $(a_1, \ldots, a_{d-1}, -a_d)$. It is easily verified that p is above (resp. on, below) H, in the x_d direction, iff $\mathcal{F}(p)$ is below (resp. on, above) $\mathcal{F}(H)$. Note also that $\mathcal{F}(\mathcal{F}(p)) = p$ and $\mathcal{F}(\mathcal{F}(H)) = H$.

Using $\mathcal{F}$ we map S to a set S' of hyperplanes and map Q to the point $q = \mathcal{F}(Q)$, both

[1] In Table 1, whenever ϵ appears in a query time or space bound, the corresponding space or query time bound contains a multiplicative factor which goes to ∞ as $\epsilon \longrightarrow 0$. Also, the constants implied in the definition of "long" segments (Problem 4) and "fat" wedges and triangles (Problems 6 and 7) are present in the space and query time bounds for these problems.

[2] In general, if h is a non-vertical hyperplane, then h^+ (resp. h^-) is the halfspace above (resp. below) h; unless specified otherwise, these halfspaces are closed.

in $\mathcal{R}^d$. Our problem is now equivalent to: "Report or count the i distinct colors of the hyperplanes lying on or above q, i.e., the hyperplanes that are intersected by the vertical ray r emanating upwards from q."

Let S_c be the set of hyperplanes of color c. For each color c, we compute the *upper envelope* E_c of the hyperplanes in S_c. E_c is the locus of the points of S_c of maximum x_d-coordinate for each point on the plane $x_d = 0$. E_c is a d-dimensional convex polytope which is unbounded in the positive x_d direction. Formally, E_c's boundary is composed of j-faces, $0 \leq j \leq d-1$, where each *j-face* is a j-dimensional convex polytope. Of particular interest to us are the $(d-1)$-faces of E_c; we call each such face a *facet*. For instance, in $\mathcal{R}^2$, E_c is an unbounded convex chain and its facets are line segments; in $\mathcal{R}^3$, E_c is an unbounded convex polytope whose facets are convex polygons.

For now, let us assume that r is well-behaved in the sense that for no color c does r intersect two or more facets of E_c at a common boundary—for instance, a vertex in $\mathcal{R}^2$ and an edge or a vertex in $\mathcal{R}^3$. (In Section 2.3 we show how to remove this assumption.) Then, by definition of the upper envelope, it follows that (i) r intersects a c-colored hyperplane iff r intersects E_c and, moreover, (ii) if r intersects E_c, then r intersects a unique facet of E_c (in the interior of the facet). Let $\mathcal{E}$ be the collection of the envelopes of the different colors. By the above discussion, our problem is equivalent to: "Report or count the facets of $\mathcal{E}$ that are intersected by r," which is a standard intersection searching problem! In Sections 2.1 and 2.2, we show how to solve efficiently this ray-envelope intersection problem in $\mathcal{R}^2$ and in $\mathcal{R}^3$. This approach does not give an efficient solution to the generalized halfspace searching problem in $\mathcal{R}^d$ for $d > 3$; for this case, we give a different solution in Section 2.4.

2.1 Solving the ray–envelope intersection problem in $\mathcal{R}^2$

We project the endpoints of the line segments of $\mathcal{E}$ onto the x-axis, thus partitioning it into $2n+1$ *elementary intervals* (some of which may be empty). We build a *segment tree* T which stores these elementary intervals at the leaves. Let v be any node of T. Associated with v is an x-interval $I(v)$, which is the union of the elementary intervals stored at the leaves in v's subtree. Let *Strip*(v) be the vertical strip defined by $I(v)$. We say that a segment $s \in \mathcal{E}$ is *allocated* to a node $v \in T$ iff $I(v) \neq \emptyset$ and s crosses *Strip*(v) but not *Strip*$(parent(v))$. Let $\mathcal{E}(v)$ be the set of segments allocated to v. Within *Strip*(v), the segments of $\mathcal{E}(v)$ can be viewed as lines since they cross *Strip*(v) completely. Let $\mathcal{E}'(v)$ be the set of points dual to these lines. We store $\mathcal{E}'(v)$ in an instance $H(v)$ of the standard halfplane reporting (resp. counting) structure for $\mathcal{R}^2$ given in [CGL85] (resp. [Mat92b]). This structure uses $O(m)$ space and has a query time of $O(\log m + k_v)$ (resp. $O(m^{1/2})$), where $m = |\mathcal{E}(v)|$ and k_v is the output size at v.

To answer a query, we search in T using q's x-coordinate. At each node v visited, we need to report or count the lines intersected by r. But, by duality, this is equivalent to answering, in $\mathcal{R}^2$, a halfplane query at v using the query $\mathcal{F}(q)^- = Q^-$, which we do using $H(v)$. For the reporting problem, we simply output what is returned by the query at each visited node; for the counting problem, we return the sum of the counts obtained at the visited nodes.

Theorem 2.1 *A set S of n colored points in $\mathcal{R}^2$ can be stored in a data structure of size $O(n \log n)$ so that the i distinct colors of the points lying in any query halfplane can be reported (resp. counted) in time $O(\log^2 n + i)$ (resp. $O(n^{1/2})$).*

Proof Correctness follows from the preceding discussion. As noted earlier, there are $O(|S_c|)$ line segments (facets) in E_c; thus $|\mathcal{E}| = O(\sum_c |S_c|) = O(n)$ and so $|T| = O(n)$. Hence each segment of $\mathcal{E}$ can get allocated to $O(\log n)$ nodes of T. Since the structure $H(v)$ has size linear in $m = |\mathcal{E}(v)|$, the total space used is $O(n \log n)$. For the reporting problem, the query time at a node v is $O(\log m + k_v) = O(\log n + k_v)$. When summed over the $O(\log n)$ nodes visited, this gives $O(\log^2 n + i)$. To see this, recall that the ray r can intersect at most one envelope segment of any color; thus the terms k_v, taken over all nodes v visited, sum to i.

For the counting problem, the query time at v is $O(m^{1/2})$. It can be shown that if v has depth j in T, then $m = |\mathcal{E}(v)| = O(n/2^j)$. (See, for instance, [CJ92, page 675].) Thus, the overall query time is $O(\sum_{j=0}^{O(\log n)} (n/2^j)^{1/2})$, which is $O(n^{1/2})$. $\square$

2.2 Solving the ray–envelope intersection problem in $\mathcal{R}^3$

For each color c, we triangulate the facets of E_c. Let T_c denote the set of resulting triangles and let $\mathcal{T} = \bigcup_c T_c$. For any triangle $t \in \mathcal{T}$, let $h(t)$ be the supporting plane of t and let t' be the projection of t (also a triangle) onto the xy–plane. Let $\mathcal{T}'$ be the set of such projected triangles. Let q' be the projection of q (the origin of the vertical query ray r) on the xy–plane. Clearly, t is intersected by r iff (a) t''s interior contains q' and (b) $h(t)$ is on or above q.

Let us now consider how to find the triangles satisfying condition (a). We first divide each triangle $t' \in \mathcal{T}'$ that does not have a vertical side into at most two such triangles by drawing a vertical line through its vertex of median x-coordinate. We store the resulting set of triangles (which we continue to call $\mathcal{T}'$) in a segment tree T according to their x-spans. Let v be any node of T and let $A(v)$ be the set of triangles of $\mathcal{T}'$ allocated to v. Let $m = |A(v)|$. Note that if $t' \in A(v)$ then both its non-vertical sides, call the upper one t'_u and the lower one t'_l, cross $Strip(v)$. If $q' \in Strip(v)$, then q' is in t''s interior iff q' is above t'_l and below t'_u. Since t'_l and t'_u behave like lines within $Strip(v)$, by duality we have that $q' \in t'$ iff in $\mathcal{R}^2$ the line $\mathcal{F}(q')$ intersects the segment $\overline{\mathcal{F}(t'_l)\mathcal{F}(t'_u)}$, i.e., iff one endpoint of $\overline{\mathcal{F}(t'_l)\mathcal{F}(t'_u)}$ lies in the open halfplane $\mathcal{F}(q')^-$ and the other lies in the open halfplane $\mathcal{F}(q')^+$. Let us denote these halfplane queries by J_1 and J_2, respectively.

Next, consider condition (b). By duality, $h(t)$ is on or above q iff in $\mathcal{R}^3$ the point $\mathcal{F}(h(t))$ is in the halfspace $\mathcal{F}(q)^-$. Denote this halfspace query by J_3.

So, our problem at v is to report or count the triangles of $A(v)$ that satisfy J_1, J_2, and J_3. We can do this by augmenting v with a 3-level data structure based on partition trees [Mat92a], as follows: Let E be the set of endpoints $\{\mathcal{F}(t'_l), \mathcal{F}(t'_u) \mid t' \in \mathcal{T}'\}$ in $\mathcal{R}^2$. We build a partition tree D_1 on E. Using the space-reduction strategy of Dobkin and Edelsbrunner [DE87] (see also [AvKO93, vK92]), we consider a constant number of levels of D_1 and augment each node w at these levels with a partition tree $D_2(w)$, which is built on the subset of E associated with w. Finally for each such w, again using the above-mentioned space-reduction strategy, we consider a constant number of levels of $D_2(w)$ and augment each node u at these levels with an instance $D_3(u)$ of a data structure for halfspace range reporting or counting, which is built on the subset of E associated with u. Let us denote this 3-level structure at v by $\mathcal{D}(v)$.

To report or count the triangles of $A(v)$ satisfying the queries J_1, J_2, and J_3, we perform J_1 on D_1, then perform J_2 on $D_2(w)$, for each canonical node w of D_1 identified by J_1, and finally perform J_3 on $D_3(u)$ for each canonical node u of $D_2(w)$ identified by J_2.

Let us analyse the space and query time of the structure $\mathcal{D}(v)$. We discuss the reporting version first. Let $f_3(p)$ be the space used by an instance of the reporting version of D_3 built on p points and let $g_3(p) + k$ be its query time, where k is the output size. (Throughout, we will use the generic symbol k to denote the output size of a query on some data structure.) If $f_3(p)/p$ is non-decreasing and $g_3(p)/p$ is non-increasing, then it can be shown (see [vK92, Theorem 5.8(ii), page 69]) that $\mathcal{D}(v)$ uses $O(m + f_3(m))$ space and has a query time of $O(m^\epsilon(m^{1/2} + g_3(m)) + k)$, where $\epsilon > 0$ is an arbitrarily small constant. (Recall that $m = |A(v)|$.) We can use for D_3 the structure given in [AHL90] for which $f_3(p) = O(p \log p)$ and $g_3(p) = O(\log p + k)$. Then $\mathcal{D}(v)$ has size $O(m \log m)$ and query time $O(m^{1/2+\epsilon} + k)$.

Instead of partition trees, we can also use 2-dimensional cutting trees [Mat91a] for the two outer levels of $\mathcal{D}(v)$. Then $\mathcal{D}(v)$ uses space $O(m^\epsilon(m^2 + f_3(m))$ and has query time $O(\log m + g_3(m) + k)$ (see [vK92, Theorem 5.8(i), page 69]). For D_3, we use the above-mentioned structure of [AHL90]. Then $\mathcal{D}(v)$ has size $O(m^{2+\epsilon})$ and query time $O(\log m + k)$.

Now consider the counting problem. Let $f_3(p)$ be the space and $g_3(p)$ be the query time for the counting version of D_3 built on p points. If we use partition trees and use for D_3 the structure given in [Mat92a], for which $f_3(p) = O(p)$ and $g_3(p) = O(p^{2/3})$, then $\mathcal{D}(v)$ uses $O(m)$ space and has query time $O(m^{2/3+\epsilon})$.

So, the overall data structure for the ray-envelope problem in $\mathcal{R}^3$ consists of the segment tree T where each node v is augmented with the above structure $\mathcal{D}(v)$, which is built on $A(v)$. To answer a query, we search down T and query $\mathcal{D}(v)$ at each node v visited.

Theorem 2.2 *The reporting version of the generalized halfspace range searching problem for a set of n colored points in $\mathcal{R}^3$ can be solved in $O(n \log^2 n)$ (resp. $O(n^{2+\epsilon})$) space and $O(n^{1/2+\epsilon} + i)$ (resp. $O(\log^2 n + i)$) query time, where i is the output size and $\epsilon > 0$ is an arbitrarily small constant. The counting version is solvable in $O(n \log n)$ space and $O(n^{2/3+\epsilon})$ query time.*

Proof Correctness follows from the preceding discussion. Consider the space and query time for the reporting problem. It is well-known (see [Mul93, page 271] for instance) that E_c has $O(|S_c|)$ facets. This implies that T_c contains $O(|S_c|)$ triangles. Thus $|\mathcal{T}| = O(\sum_c |S_c|) = O(n)$ and so the segment tree T has size $O(n)$.

A triangle $t' \in \mathcal{T}'$ can get allocated to $O(\log n)$ nodes of T. Since the auxiliary structure $\mathcal{D}(v)$ at a node $v \in T$ has size $O(m \log m)$ (resp. $O(m^{2+\epsilon})$), where $m = |A(v)|$, it follows that the overall space is $O(n \log^2 n)$ (resp. $O(n^{2+\epsilon'})$), for some $\epsilon' > 0$. The query time at v is $O(m^{1/2+\epsilon} + k_v)$ (resp. $O(\log m + k_v)$), where k_v is the output size at v. As in the proof of Theorem 2.1, taken over all nodes v visited, this sums to $O(n^{1/2+\epsilon} + i)$ (resp. $O(\log^2 n + i)$).

The analysis for the counting problem is similar and hence omitted. $\square$

An application: Consider the *generalized disk range searching* problem mentioned in the Introduction: "Given a set S of n colored points in $\mathcal{R}^2$, count or report the i distinct colors of the points lying inside a variable-radius query disk q." Using the well-known lifting transformation [Ede87], this problem can be transformed to the generalized halfspace range searching problem in $\mathcal{R}^3$ and hence can be solved within the bounds of Theorem 2.2. In [GJS93b], the reporting (resp. counting) version of this generalized disk range searching problem is solved in $O(n^4)$ (resp. $O(n^4 \log n)$) space and $O(\log n + i)$ (resp. $O(\log n)$) query time using a different approach.

Remark 2.1 In addition to the bounds given in Theorem 2.2, other bounds are also possible. For instance, the reporting problem is solvable using partition trees and a version of D_3 from [Mat92b]; the bounds are $O(n \log n)$ space and $O(n^{2/3+\epsilon} + i)$ query time. Also, by using a combination of partition trees and cutting trees [vK92, Theorem 5.8(iii), page 69]), we can obtain a space-query time tradeoff. We have omitted a detailed discussion of these results since they can be derived in the same way as the bounds in Theorem 2.2.

We also note that for the counting problem, there is a solution, based on cutting trees, which uses $O(n^{3+\epsilon})$ space and has $O(\log^2 n)$ query time. However, this can be improved as we will see in Section 2.4.

2.3 Handling special cases in $\mathcal{R}^2$ and $\mathcal{R}^3$

Vertical queries

So far we have assumed that the query halfspace Q is non-vertical, i.e., not parallel to the x_d direction, which allowed us to apply the transformation $\mathcal{F}$. If Q is vertical, then we proceed as follows:

In $\mathcal{R}^2$, Q is a vertical line, say $Q : x = x_0$. We simply project all the points on the x-axis and solve an instance of the *generalized 1-dimensional range searching* problem with the query interval $[x_0, \infty)$, i.e., report or count the distinct colors lying in the interval $[x_0, \infty)$. As shown in [GJS93a], for a semi-infinite query interval, the problem is solvable in $O(n)$ space and $\Theta(i)$ (resp. $O(\log n)$) query time for the reporting (resp. counting) case. Thus the bounds of Theorem 2.1 are unaffected.

Similarly, in $\mathcal{R}^3$, if Q is the vertical hyperplane $Q : a_1x + a_2y + a_3 = 0$, then we project the points onto the plane $z = 0$ and solve an instance of the generalized halfplane range searching problem in $\mathcal{R}^2$, using as query the projection of Q onto the plane $z = 0$. Thus the bounds in Theorem 2.2 are unaffected.

Query rays that are not well-behaved

So far we have assumed that the vertical ray r is well-behaved in the sense that it does not meet two or more facets of E_c at a common boundary, for any color c. Let us consider what happens if this is not true in $\mathcal{R}^2$. Thus, r meets E_c at a vertex p. Let a and b be the line segments of E_c having p as endpoint. At first sight it would appear that the solution returned to the standard counting problem is not valid for the generalized counting problem since the count would include c twice—and we would not be aware of this. (Of course, this is not an issue for the reporting problem.) Fortunately, however, the solution given in Section 2.1 carries over unchanged even if r is not well-behaved. The argument is as follows:

Let v be the node of T, with left child u and right child w, such that p lies on the common boundary of *Strip*(u) and *Strip*(w). Thus, during the search down T with q (r's origin), when q falls on this common boundary, we are faced with the question of which child of v to visit. (Clearly, this situation will not arise at any other node.) The answer is that we can arbitrarily pick either u or w to visit without affecting the correctness of the query. To see why, notice that since E_c is an unbounded convex chain, in both *Strip*(u) and in *Strip*(w) there will be exactly one c-colored line segment on or above q (namely, a and b, respectively, assuming that a is to the left of p and b is to the right). Thus, regardless of whether u or w is visited, color c will be reported or counted when the auxiliary structure of the visited node is queried.

In $\mathcal{R}^3$ the situation is more subtle. Here r can intersect an edge shared by triangles s and t of T_c or the common vertex of several triangles $s, t, \ldots$ of T_c. (The latter case is bad even for the reporting problem since the query time will be high if the common vertex has many triangles incident with it.) Consider the first case. It follows that q' lies on the common edge of triangles s' and t' of $\mathcal{T}'$. This implies that one endpoint of $\overline{\mathcal{F}(s'_l)\mathcal{F}(s'_u)}$ touches $\mathcal{F}(q')$ while the other is (say) below $\mathcal{F}(q')$ and a symmetric situation exists w.r.t. $\overline{\mathcal{F}(t'_l)\mathcal{F}(t'_u)}$. If we use open halfplanes in doing the queries J_1 and J_2 (as we do in Section 2.2), then c will be missed. If we use closed halfplanes then c will be found twice, which is unacceptable for the counting problem. The solution is to use an open halfplane in doing (say) J_1 and a closed halfplane in doing J_2; with this approach c will be found exactly once.

Suppose instead that r meets the common vertex of several c-colored triangles $s, t, \ldots$. From the properties of $\mathcal{F}$, it follows that the line segments $\overline{\mathcal{F}(s'_l)\mathcal{F}(s'_u)}, \overline{\mathcal{F}(t'_l)\mathcal{F}(t'_u)}, \ldots$ are all contained in $\mathcal{F}(q')$. Thus, with the modified queries J_1 and J_2, c will be missed. The solution is to identify such colors c separately, as follows: In preprocessing, we project all envelope vertices onto the plane $z = 0$. Let p' be the projection of vertex p. We check if q' coincides with any p' and, if it does, then we report or count its color if p is on or above q. (If several differently-colored vertices $p_1, p_2, \ldots$ all project to the same point p', then we store them with p' in sorted z-order and do a binary search on them to determine the ones that are on or above q.) Clearly, all this can be done in $O(n)$ space and $O(\log n + i)$ (resp. $O(\log n)$) query time for the reporting (resp. counting) case and so the bounds of Theorem 2.2 are unaffected.

2.4 Generalized halfspace range searching in $d > 3$ dimensions

The approach of Section 2.2 can be extended, with some modifications, to any fixed $d > 3$ also. However, the space requirements are high—the scheme uses $O(n^{d\lfloor d/2\rfloor+\gamma})$ space and has query time $O(\log n + i)$ (resp. $O(\log n)$) for reporting (resp. counting), where $\gamma > 0$ is an arbitrarily small constant. This approach uses cutting trees. By using a combination of cutting trees and partition trees the space can be lowered slightly at the expense of a higher query time. If one tries to use partition trees alone to obtain a close-to-linear-space solution, the query time becomes superlinear! (Intuitively, this is because the input to the ray-envelope problem is large—it is a collection of $O(n^{\lfloor d/2 \rfloor})$ $(d-1)$-dimensional simplices.) We leave open the problem of obtaining a near-linear-space solution with a query time of the form $O(n^p + i \cdot \mathrm{polylog}(n))$, $0 < p < 1$.

In the remainder of this section, we describe two different approaches whose space requirements are much lower than the bound given above (but still nowhere close to linear). The first solution works for the reporting problem only, while the second works for both reporting and counting. (In fact, these methods work for $d = 2, 3$ also.)

The first solution is as follows: In preprocessing, we store the distinct colors in the input point-set 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 points of S colored with the colors in $C(v)$. At v, we store a data structure $HSE(v)$ to solve the *halfspace emptiness* problem on $S(v)$, i.e., "Does a query halfspace contain any points of $S(v)$?" $HSE(v)$ returns "true" if the query halfspace is empty and "false" otherwise. If $|S_v| = n_v$, then $HSE(v)$ uses $O(n_v^{\lfloor d/2 \rfloor + \epsilon})$ space and has query time $O(\log n_v)$ [Mul93, page 290], for some arbitrarily small constant $\epsilon > 0$.

We answer a generalized halfspace reporting query for a halfspace Q^- as follows: We do a depth-first search in CT and query $HSE(v)$ at each node v visited. If v is a non-leaf then we continue searching below v iff the query returns "false"; if v is a leaf, then we output the color stored there iff the query returns "false".

Theorem 2.3 *For any fixed $d \geq 2$, a set S of n colored points in $\mathcal{R}^d$ can be stored in a data structure of size $O(n^{\lfloor d/2 \rfloor + \epsilon})$ such that the i distinct colors of the points contained in a query halfspace Q^- can be reported in time $O(\log n + i \log^2 n)$. Here $\epsilon > 0$ is an arbitrarily small constant.*

Proof We argue that a color c is reported iff there is a c-colored point in 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 $HSE(v)$ returns "false". Thus, some point in $S(v)$ is in Q^-. Since v is a leaf, all points in $S(v)$ have the same color c and the claim follows.

For the converse, suppose that Q^- contains a c-colored point 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 $HSE(v')$ will return "false", which implies that v will be visited and c will be output.

CT uses $O(n^{\lfloor d/2 \rfloor + \epsilon})$ space per level and there are $O(\log n)$ levels. Thus, the overall space is $O(n^{\lfloor d/2 \rfloor + \epsilon'})$, for an arbitrarily-small constant $\epsilon' > 0$. The query time can be upper-bounded as follows: If $i = 0$, then the query on $HSE(\mathrm{root})$ returns "true" and we abandon the search at the root itself; in this case, the query time is just $O(\log n)$. Suppose that $i \neq 0$. Call a visited node v *fruitful* if the query on $HSE(v)$ returns "false" 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 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 $O(i \log n)$, which implies that the total time spent at these nodes is $O(i \log^2 n)$. The claimed query time follows. $\square$

The second solution is based on cutting trees. Since the idea is similar to that for the standard problem (see, for instance, [vK92]), we will just sketch the idea. We dualize S to a set S' of hyperplanes and Q to a point q all in $\mathcal{R}^d$. Then we compute a cutting tree T for S', based on a $\frac{1}{r}$-cutting, where $r = n^\epsilon$ for some arbitrarily small constant $\epsilon > 0$. T will have constant depth. For any node $v \in T$, let H_v be the hyperplanes of S' that are above the simplex of the cutting associated with v. We store with v a list of the distinct colors of the hyperplanes in H_v. To answer a query we search along a path in T and at each visited node we output the colors stored there. The space and query time can be shown to be $O(n^{d+\epsilon})$ and $O(\log n + i)$, respectively (note that any color is reported only a constant number of times).

This approach will not work for the counting problem since a color can be reported more than once. With a small modification, we can solve the counting problem in $O(n^d)$ space and $O(\log n)$ query time. We compute for S' a cutting tree T' based on a $\frac{1}{r}$-cutting for some constant r. Thus T' has depth $O(\log n)$. For any leaf l, we consider the union of the sets H_v for all nodes v on the root-to-l path in T', determine the distinct colors in this union, and store a count of these at l. To answer a counting query, we search down T to a leaf and output its count. It is easy to see that the method is correct.

Theorem 2.4 *For any fixed $d \geq 2$, the generalized halfspace range reporting (resp. counting) problem for n colored points in $\mathcal{R}^d$ can be solved in $O(n^{d+\epsilon})$ (resp. $O(n^d)$) space and $O(\log n + i)$ (resp. $O(\log n)$) query time. Here $\epsilon > 0$ is an arbitrarily small constant.* $\square$

2.5 Extensions

We show briefly how to extend the approach of Section 2.1 to solve two other problems in $\mathcal{R}^2$. In the first problem, we wish to preprocess a set $S = \{s_1, s_2, \ldots, s_n\}$ of colored line segments in $\mathcal{R}^2$ so that for any query halfspace Q^-, the i distinct colors of the line segments that lie completely in Q^- can be reported or counted efficiently.

We dualize each s_i to a doublewedge $w_i = \mathcal{F}(s_i)$. Our problem is now equivalent to: "Count or report the distinct colors of the w_i that lie above the point $\mathcal{F}(Q)$." Let $\wedge_i$ be the lower envelope of w_i; $\wedge_i$ consists of two rays emanating from a common point. Let r be the upward-directed vertical ray emanating from $\mathcal{F}(Q)$. Clearly, w_i is above $\mathcal{F}(Q)$ iff r intersects $\wedge_i$. Let E_c be the upper envelope of the $\wedge_i$'s that have color c. E_c consists of line segments (some of which are rays) and it is well-known that $|E_c| = O(n_c)$, where n_c is the number of c-colored $\wedge_i$'s (see [Ede87, page 377, Problem 15.6]). Moreover, for any color c, we have: (i) r intersects a c-colored $\wedge_i$ iff r intersects E_c and (ii) if r intersects E_c, then it intersects a unique line segment of E_c. (Again, assume that r is well-behaved; otherwise, the discussion of Section 2.3 for $d = 2$ will apply.) Let $\mathcal{E}$ be the set of upper envelopes of all colors. Our problem now is: "Count or report the segments of $\mathcal{E}$ that are intersected by r," which we can solve as in Section 2.1 by storing $\mathcal{E}$ in a segment tree. We conclude directly:

Theorem 2.5 *A set S of n colored line segments in $\mathcal{R}^2$ can be stored in a data structure of size $O(n \log n)$ so that the i distinct colors of the segments that are contained completely in a query halfplane can be reported (resp. counted) in $O(\log^2 n + i)$ (resp. $O(n^{1/2})$) time.* □

In the second problem, we wish to preprocess a set $S = \{s_1, s_2, \ldots, s_n\}$ of n colored line segments in $\mathcal{R}^2$ so that the distinct colors of the ones that are intersected by an upward-directed vertical query ray r can be reported efficiently.

We stay in the primal space. Let S_c be the set of c-colored segments of S. We compute the upper envelope E_c of S_c. If $|S_c| = m_c$, then $|E_c| = O(\lambda_3(m_c)) = O(m_c\alpha(m_c))$, where $\lambda_3(m_c)$ is the maximum length of a Davenport–Schinzel sequence of order $k = 3$ on m_c symbols and $\alpha(\cdot)$ is the slow-growing functional inverse of Ackermann's function [Sha88].

As before, we have the properties that for any c (i) r intersects a c-colored segment iff it intersects E_c and (ii) if r intersects E_c then it intersects a unique segment of E_c. We can now solve the problem using a segment tree as before.

Theorem 2.6 *A set S of n colored line segments in $\mathcal{R}^2$ can be stored in a data structure of size $O(n\alpha(n) \log n)$ so that the i distinct colors of the line segments that are intersected by a vertical query ray can be reported (resp. counted) in time $O(\log^2 n + i)$ (resp. $O((n\alpha(n))^{1/2})$).* □

Proof Since $|E_c| = O(m_c\alpha(m_c))$ and $\sum_c m_c = n$, the total size of the envelopes of all colors is $O(n\alpha(n))$. From this the $O(n\alpha(n) \log n)$ space bound follows. The query time for the reporting problem is clear. For the counting problem, the query time at a node v is $O(|S(v)|^{1/2})$. Since $|S(v)| = O(|T|/2^j) = O(n\alpha(n)/2^j)$ for a level j node v, the query time when summed over all nodes v visited is $O((n\alpha(n))^{1/2})$. $\square$

3 Generalized intersection searching on lines and line segments

We consider several versions of the generalized intersection searching problem for a set S of n colored lines or line segments; the query is either a line or a line segment. Specifically, in Section 3.1, we consider the case where S consists of colored lines and q is a vertical line segment. Even this restricted problem turns out to be quite nontrivial and leads to an interesting solution which exhibits a space-query time tradeoff. Note that, by definition of $\mathcal{F}$ for $\mathcal{R}^2$, this problem is equivalent to searching on a set of colored points with a query strip (i.e., the region of the plane that is enclosed by two parallel lines). In Section 3.2, we consider the case where S consists of colored line segments and q is a line. Finally, in Section 3.3, we consider the case where S consists of colored line segments and q is also a line segment.

3.1 Querying colored lines with a vertical line segment

For simplicity, assume that the lines of S are in general position, i.e., no three lines intersect at a common point. This assumption can be removed easily. The following simple approach is the starting point for our solution.

We construct the arrangement $\mathcal{A}$ of the lines in S and draw a vertical line through each of the $t = O(n^2)$ intersection points, thereby dividing the plane into $t+1$ strips $V_1, V_2, \ldots, V_{t+1}$, ordered from left to right. Within any strip V_k, the lines are non-intersecting and so can be totally ordered, say, from top to bottom, as $\ell_1, \ell_2, \ldots, \ell_n$. Denote this total order by E_k. We store E_k in a balanced binary search tree T_k. We also store the sequence $1, 2, \ldots, n$ of colored integers, where integer j gets the color of ℓ_j, in an instance D_k of the data structure given in [GJS93a] for generalized 1-dimensional range reporting. Given a query interval $[a, b]$, D_k reports in $O(\log n + i)$ time the i distinct colors of the integers contained in $[a, b]$. The structure uses $O(n)$ space and, moreover, supports updates in $O(\log n)$ time with $O(\log n)$ memory modifications per update. Finally, we store the x-coordinates of the strip boundaries in sorted order in an array A.

Let the endpoints of the query segment be $(\bar{x}, y_1)$ and $(\bar{x}, y_2)$, where $y_1 \geq y_2$. We determine the strip V_k containing $\bar{x}$ by binary search on A. We then search in T_k and find lines ℓ_a and ℓ_b such that ℓ_a (resp. ℓ_b) is the highest (resp. lowest) line in E_k that is on or below (resp. on or above) $(\bar{x}, y_1)$ (resp. $(\bar{x}, y_2)$). We then query D_k with the interval $[a, b]$ and output the colors reported. To see the correctness of the method note that the lines in E_k are totally ordered and, moreover, they all cross V_k; thus the ones intersected by q have their indices in $[a, b]$. Clearly, the total space is $O(n^3)$ and the query time is $O(\log n + i)$.

We can reduce the space to $O(n^2 \log n)$, using persistence [DSST89], as follows: We sweep over the strips from left to right. If V_k is the current strip, then we update D_{k-1} in a partially-persistent way to get D_k. This update involves interchanging in D_{k-1} the colors of the two integers corresponding to the two lines l' and l'' that intersect on the boundary between V_{k-1} and V_k. This can be done via two insertions and two deletions. Similarly, we also update T_{k-1} by interchanging l' and l''; this gives T_k. The space used per update is $O(\log n)$ for a D-structure and $O(1)$ for a T-structure [DSST89], which implies the claimed space bound. Let $\mathcal{D}$ and $\mathcal{T}$ denote the D- and T-structures, respectively. Given q, we locate V_k, then query the kth version of $\mathcal{T}$ to find $[a, b]$, and then query the kth version of $\mathcal{D}$ with this.

Lemma 3.1 *A set S of n colored lines in $\mathcal{R}^2$ can be stored in a data structure of size $O(n^2 \log n)$ such that the i distinct colors of the lines that are intersected by a vertical query segment q can be reported in $O(\log n + i)$ time.* □

3.1.1 A space-query time tradeoff

Let μ be a tunable parameter in the range $0.5 < \mu < 1$. We give a solution which uses $O(n^{2.5-\mu} \log n)$ space and has a query time of $O(n^\mu + i)$. Our approach builds upon a technique based on filtering search which was given in [AvKO93] for a different (standard) problem. (However, due to the generalized nature of the problem we consider, our approach differs from the one in [AvKO93] in some key respects—see Remark 3.2 later.) In a nutshell, our approach is as follows: We extract from the sequence $\mathcal{E} = (E_1, \ldots, E_{t+1})$ a subsequence $\mathcal{E}' = (E'_1, \ldots, E'_m)$ such that (i) E'_j and E'_{j+1} differ by a swap and (ii) for each $E_i \in \mathcal{E}$ there is an $E'_j \in \mathcal{E}'$ which approximates E_i in a sense that we will formalize later. Here $m = O(n^{2.5-\mu})$. We store $\mathcal{E}'$ (rather than $\mathcal{E}$) in a partially-persistent structure and perform queries on this.

Constructing $\mathcal{E}'$

We begin by defining a sequence of distinguished list positions (integers) called *borders*. The borders are $b_i = (i-1)\lfloor n^\mu + 1 \rfloor + 1$ for $1 \le i \le B$, where $B = \lceil n / \lfloor n^\mu + 1 \rfloor \rceil = \Theta(n^{1-\mu})$ is the number of borders. As we construct $\mathcal{E}'$, for each E'_j we will record in a variable δ_j the index of the first list of $\mathcal{E}$ that E'_j approximates. Thus, for $1 \le j \le m$, E'_j approximates the lists E_i such that $\delta_j \le i \le \delta_{j+1} - 1$.

We begin the construction by setting $E'_1 = E_1$ and $\delta_1 = 1$. Assume that we have scanned $\mathcal{E}$ upto E_i and have constructed $E'_1, \ldots, E'_j$. Suppose that E_{i+1} is obtained from E_i by swapping the b_kth line α with either the $(b_k - 1)$th line β or the $(b_k + 1)$th line γ for some border b_k; we call this a *border-swap* or a *b_k-swap*. We then construct E'_{j+1} by swapping, in E'_j, the line α with the line β or the line γ, as appropriate, set $\delta_{j+1} = i + 1$, and move on to E_{i+1}. (As we will see, this is a crucial change from [AvKO93] where only a swap of the b_kth and (b_k+1)th lines in E_i triggers the construction of E'_{j+1}.) Note that the swapped lines need not be adjacent in E'_j. If the transition from E_i to E_{i+1} does not involve a border-swap, then we simply move on to E_{i+1}. Once $\mathcal{E}$ has been scanned, we set $\delta_{m+1} = t + 2$ (recall that $t + 1 = |\mathcal{E}|$).

The following lemma establishes some key properties of $\mathcal{E}'$.

Lemma 3.2 *Suppose that $\mathcal{E}' = (E'_1, \ldots, E'_m)$ has been constructed as above from $\mathcal{E} = (E_1, \ldots, E_{t+1})$. Then*

(a) *E'_j and E'_{j+1} differ by a swap, $1 \le j < m$.*

(b) *For any j such that $1 \le j \le m$ and any i, k such that $\delta_j \le i < k \le \delta_{j+1} - 1$, E_i and E_k agree at all borders. That is, for each border b_l the b_lth line in E_i and E_k is the same. Thus, at any x-coordinate within the superstrip $\mathcal{V}_j = V_{\delta_j} \cup V_{\delta_j+1} \cup \cdots \cup V_{\delta_{j+1}-1}$ the b_1th, b_2th, $\ldots$, b_Bth lines are the same and hence they can be totally ordered within $\mathcal{V}_j$.*

(c) *For any j such that $1 \le j \le m$ and any i such that $\delta_j \le i \le \delta_{j+1} - 1$, E'_j and E_i agree at all borders. Thus the border-lines of E'_j, i.e., the lines at positions $b_1, \ldots, b_B$ in E'_j, respect the total order of part (b).*

(d) *For any j such that $1 \le j \le m$ consider any i such that $\delta_j \le i \le \delta_{j+1} - 1$. Let b_l and b_{l+1} be any two successive borders. Then E'_j approximates E_i in the following sense: The (unordered) set of lines at positions $b_l + 1, \ldots, b_{l+1}$ in E'_j is the same as the (unordered) set of lines in E_i at these same positions.*

Proof

(a) Immediate from the way in which E'_{j+1} is constructed from E'_j.

(b) By construction, the transition from E_i to E_{i+1} does not involve swapping a border-line (which is unlike [AvKO93]). Thus E_i and E_{i+1} agree at all borders. Similarly for E_{i+1}, E_{i+2} and so on up to E_{k-1}, E_k. The claim follows.

(c) By part (b), it suffices to show that E_{δ_j} and E'_j agree at all borders. We use induction on j. The claim is true for $j = 1$ since we have $E'_1 = E_1 = E_{\delta_1}$. For $j > 1$ assume that E'_{j-1} and $E_{\delta_{j-1}}$ agree at all borders. By part (b), this implies that E'_{j-1} and $E_{\delta_j - 1}$ also agree at all borders.

Suppose that the transition from $E_{\delta_j - 1}$ to E_{δ_j} involves the swap of the b_kth and (say) the $(b_k - 1)$th lines of $E_{\delta_j - 1}$, for some border b_k. Let these lines be α and β, respectively. Since $E_{\delta_j - 1}$ and E_{δ_j} differ in only the positions b_k and $b_k - 1$, it follows that E'_{j-1} and E_{δ_j} agree at all border positions except b_k—at b_k E_{δ_j} contains β while E'_{j-1} contains α. Now, in constructing E'_j, we swap α and β in E'_{j-1}. Thus, β ends up in position b_k in E'_j and so E'_j and E_{δ_j} agree at all borders. (Note that because of the way we define borders, successive borders differ by $\lfloor n^\mu + 1 \rfloor \geq 2$. Thus, $b_k - 1$ is not a border. This is not required, but it shortens the proof a little.)

We note that properties (b) and (c) above, which are crucial for our solution, are not guaranteed by [AvKO93].

(d) The proof is similar to that in [AvKO93] and hence omitted. In [AvKO93], only swaps involving lines at positions b_k and $b_k + 1$ are considered. We need to also consider swaps of lines at positions b_k and $b_k - 1$, which can be done in the same way.

□

The following lemma provides an upper bound on the length of $\mathcal{E}'$.

Lemma 3.3 *The sequence* $\mathcal{E}' = (E'_1, \ldots, E'_m)$ *has length* $m = O(n^{2.5-\mu})$, *where* $0.5 < \mu < 1$ *is a tunable parameter.*

Proof We add a new list to $\mathcal{E}'$ whenever a b_k-swap occurs for some b_k, i.e., whenever we encounter in the arrangement $\mathcal{A}$ a vertex v such that there are $b_k - 2$ or $b_k - 1$ lines above v (depending on whether v is the intersection of the b_kth line with the $(b_k - 1)$th line or with the $(b_k + 1)$th line.) Thus, v is either a *level-*$(b_k - 1)$ or a *level-*b_k vertex in $\mathcal{A}$

[Ede87]. Let $N(j,n)$ be the maximum number of level-j vertices in an arrangement of n lines in the plane. The total number of b_k-swaps that occur during the construction of $\mathcal{E}'$ is thus $O(N(b_k-1,n)+N(b_k,n))$. Thus, $m=|\mathcal{E}'|=O(\sum_{k=1}^{B}N(b_k-1,n)+N(b_k,n))$.

We have $\sum_{k=1}^{B}N(b_k,n) = \sum_{k=1}^{B/2}N(b_k,n)+\sum_{k=B/2+1}^{B}N(n-b_k+1,n)$, since, by symmetry, we have $N(b_k,n)=N(n-b_k+1,n)$. From [Ede87] (Theorem 3.3 on page 49 and Corollary 3.18 on page 60), it follows that $N(b_k,n)=O(n\sqrt{b_k})$ for $1\leq b_k\leq n/2$. It is easy to verify that for $k\leq B/2$, we have $b_k\leq n/2$ and for $k\geq B/2+1$, we have $b_k\geq n/2+1$; the latter implies that $n-b_k+1\leq n/2$. It follows now that $\sum_{k=1}^{B}N(b_k,n)=O(\sum_{k=1}^{B/2}n\sqrt{b_k}+\sum_{k=B/2+1}^{B}n\sqrt{n-b_k+1})$.

Now, $\sum_{k=1}^{B/2}n\sqrt{b_k}=O(n\sum_{k=1}^{B/2}\sqrt{(k-1)n^{\mu}})=O(n^{1+\mu/2}\int_1^{n^{1-\mu}/2}\sqrt{x}dx)=O(n^{2.5-\mu})$. Likewise, $\sum_{k=B/2+1}^{B}n\sqrt{n-b_k+1}=O(n^{2.5-\mu})$. Similarly, we can show that $\sum_{k=1}^{B}N(b_k-1,n)=O(n^{2.5-\mu})$. The lemma follows. □

The data structure

The overall structure consists of four parts, as follows:

(1) An array A': For $j=1,\ldots,m-1$, $A'[j]$ contains the x-coordinate of the boundary shared by the strips $V_{\delta_{j+1}-1}$ and $V_{\delta_{j+1}}$.

(2) A structure $\mathcal{T}'$: For $j=1,\ldots,m$, let T'_j be a red-black tree storing the border-lines of E'_j according to the total order specified by Lemma 3.2(c). $\mathcal{T}'$ is obtained by making the T'_j's partially persistent using the technique given in [DSST89].

(3) A structure $\mathcal{D}'$: For $j=1,\ldots,m$, let D'_j be an instance of the generalized 1-dimensional range reporting structure given in [GJS93a]. D'_j is built on a sequence $I_j=(1,2,\ldots,n)$ of colored integers, where integer p gets the color of the pth line in E'_j. The structure $\mathcal{D}'$ is obtained by making the D'_j's partially persistent.

(4) A structure $\mathcal{I}'$: Let I'_j be a red-black tree storing the sequence I_j. $\mathcal{I}'$ is obtained by making the I'_j's partially persistent.

Remark 3.1 When constructing $\mathcal{T}'$, we do not make the red-black color fields of the different T'_j's persistent since they are not needed for searching. These fields are required only in the current T'_j to keep it balanced during updates. Similarly for $\mathcal{I}'$.

Lemma 3.4 *The above 4-part data structure occupies $O(n^{2.5-\mu}\log n)$ space.*

Proof By Lemma 3.3, $m = O(n^{2.5-\mu})$, Thus, A' uses $O(n^{2.5-\mu})$ space. From [DSST89], each memory modification in an ephemeral structure (T'_j, D'_j, or I'_j) causes the corresponding persistent structure to use an additional $O(1)$ amortized space.

By Lemma 3.2(a), successive E'_j differ by a swap. Thus, starting with T'_1 we need to perform a total of $O(m)$ updates to create $\mathcal{T}'$. Since any T'_j undergoes $O(1)$ memory modifications per update (excluding color flips) [CLR90], and since T'_1 has size $\Theta(B)$, the total space used by $\mathcal{T}'$ is $O(m + B) = O(n^{2.5-\mu})$.

A similar argument shows that $\mathcal{I}'$ also uses $O(n^{2.5-\mu})$ space.

Finally, consider $\mathcal{D}'$. As shown in [GJS93a], each D'_j undergoes $O(\log n)$ memory modifications per update. Also, D'_1 uses $O(n)$ space. It follows that the space used by $\mathcal{D}'$ is $O(n + m \log n) = O(n^{2.5-\mu} \log n)$. □

The query algorithm

Recall that the vertical query segment has endpoints $(\bar{x}, y_1)$ and $(\bar{x}, y_2)$, where $y_1 \geq y_2$. We search with $\bar{x}$ in A' and find a j such that q is in $\mathcal{V}_j = V_{\delta_j} \cup V_{\delta_j+1} \cup \cdots \cup V_{\delta_{j+1}-1}$. Next we search in the jth version of $\mathcal{T}'$ and identify the smallest (resp. greatest) border b_s (resp. b_g) in E'_j such that $(\bar{x}, y_1)$ (resp. $(\bar{x}, y_2)$) is on or above (resp. on or below) the line at position b_s (resp. b_g). We then do the following:

(i) Query the jth version of $\mathcal{D}'$ with the interval $[b_s + 1, b_g]$.

(ii) Search in the jth version of $\mathcal{I}'$ for b_s, scan the sequence of positions $b_s, b_s - 1, \ldots, b_{s-1} + 1$ and report the distinct colors of the lines at these positions that are intersected by q.

(iii) If $(\bar{x}, y_2)$ is below the b_gth line of E'_j, then search in the jth version of $\mathcal{I}'$ for b_{g+1}, scan the sequence of positions $b_g + 1, b_g + 2, \ldots, b_{g+1}$ and report the distinct colors of the lines at these positions that are intersected by q.

Lemma 3.5 *The above query algorithm is correct and runs in $O(n^\mu + i)$ time, where i is the output size.*

Proof We first prove the correctness. Let $V_k \in \mathcal{V}_j$ be the substrip containing q. By Lemma 3.2(c), the borders b_s and b_g returned by the search on E'_j are correct w.r.t. E_k also in the sense that the b_sth (resp. b_gth) line of E_k is the highest (resp. lowest) line that is on or below (resp. on or above) $(\bar{x}, y_1)$ (resp. $(\bar{x}, y_2)$). Also, by Lemma 3.2(d), for any borders b_l and b_{l+1} the unordered set of lines at positions $b_l + 1, \ldots, b_{l+1}$ is the same as the unordered set of lines at these same positions in E_k. It follows now that the unordered set of lines at positions $b_s+1,\ldots,b_g$ in E'_j is the same as the unordered set of lines at these same positions in E_k. Thus it suffices to consider E'_j rather than E_k. Since q spans the positions b_s+1 through b_g, the lines of E'_j that are intersected by it have their indices in the interval $[b_s+1, b_g]$ and so their distinct colors are reported correctly by the query on the jth version of $\mathcal{D}'$.

Again by Lemma 3.2(d), the unordered set of lines at positions $b_{s-1}+1,\ldots,b_s$ of E'_j is the same as the unordered set of lines at these same positions in E_k. Thus the scan of these lines in E'_j reports correctly the distinct colors of the lines intersected by the portion of q which lies on or above the b_sth line of E'_j. Symmetrically for the portion of q which is below the b_gth line of E'_j. This establishes the correctness.

The query on A' takes $O(\log m) = O(\log n)$ time. The time to access and query the jth version of $\mathcal{T}'$ is $O(1)+O(\log B) = O(\log n)$. Similarly, the time to access and query the jth version of $\mathcal{D}'$ is $O(1)+O(\log n + i)$. The time to access and query the jth version of $\mathcal{I}'$ is $O(1)+O(n^\mu)$ since the gap between successive borders is $O(n^\mu)$. Thus the total query time is $O(n^\mu + i)$. □

From Lemmas 3.1, 3.4, and 3.5 we get our main result:

Theorem 3.1 *A set S of n colored lines in $\mathcal{R}^2$ can be stored in a data structure of size $O(n^{2.5-\mu}\log n)$ such that the i distinct colors of the lines that are intersected by a vertical query line segment can be reported in $O(n^\mu + i)$ time. Here μ is a tunable parameter in the range $0.5 < \mu < 1$. The problem is also solvable in $O(n^2 \log n)$ space and $O(\log n + i)$ query time.* □

Remark 3.2 Our choice of borders is different from [AvKO93]. In [AvKO93], borders are not spaced equally; instead, each border is between two and three times larger than the preceding one and thus there are only $O(\log n)$ borders. These borders have the nice property that for any E_i the first n' elements of E_i are within the first $3n'$ elements of the corresponding E'_j. This choice of borders works well in [AvKO93] because the problem considered there is a standard intersection searching problem (on curves) which essentially boils down to listing the elements intersected by a downward-directed ray. Thus, because of the above-mentioned property, the time taken to scan between the b_sth and the b_{s-1}th elements is of the same order as the number of elements below the b_sth element—all of which are intersected since the query is a ray. Thus, in true filtering search fashion, the time for the scanning can be charged to the output size.

Unfortunately, this does not work in our case because (i) the query is a finite line segment and (ii) we are solving a generalized problem. Choosing borders as in [AvKO93] would result in a linear query time in the worst case.

3.2 Querying colored line segments with a line

Using $\mathcal{F}$ we map the colored line segments of S to a set S' of colored doublewedges and map the query line q to a point q'. Thus, our problem reduces to reporting the i distinct colors of the doublewedges that are stabbed by q'. As we will see in Section 4, this problem can be solved in $O(n^{3/2} \log n)$ space with a query time of $O(\log^2 n + i)$.

In the remainder of this section, we consider the special case where the segments of S all lie in the unit square $\mathcal{U}$ and each segment has length at least λ, where $\lambda > 0$ is a constant. These assumptions are reasonable for practical applications and they allow a very efficient solution.

We first give a solution for the case where all the segments intersect the y-axis Y. (For this problem, S need not satisfy the above-mentioned assumptions.) For now assume that each segment $s \in S$ truly intersects Y rather than merely touching it or being contained in it. (We discuss these special cases later.) Thus, one endpoint of s has negative x-coordinate and the other has positive x-coordinate. By the definition of $\mathcal{F}$ in $\mathcal{R}^2$, this implies that in the corresponding dual doublewedge one of the bounding lines has positive slope and the other has negative slope.

We split each doublewedge into a *left-facing wedge* (or *left-wedge* for short) and a *right-facing wedge* (or *right-wedge*) in the obvious way. Note that each wedge is y-monotone. Let us consider how to store the right-wedges. (The left-wedges can be handled symmetrically.) Because of y-monotonicity, the query point q' is contained in a right-wedge w iff the horizontal, leftward-directed ray r emanating from q' intersects the boundary of w. This suggests the following approach: For each color c, we compute the *left-envelope* of the boundaries of all c-colored right wedges, i.e., the portions of the boundaries visible from $(-\infty, 0)$. This left-envelope is a y-monotone chain of line segments; we give each segment the color c. If there are n_c c-colored right wedges, then the c-colored left-envelope has size $O(n_c)$ (see [Ede87, page 357, Problem 15.6]).

In this way, we obtain a collection S'' of colored line segments in the plane. Note that (i) r intersects the boundary of a c-colored right-wedge iff r intersects a c-colored left-envelope and (ii) if r intersects a c-colored left-envelope then it intersects a unique line segment of this envelope. Thus we have transformed our generalized problem into a standard one and we can solve the latter by storing S'' in a segment tree as in Section 2.1. We conclude directly:

Lemma 3.6 *A set S of n colored line segments in the plane, where all the segments intersect the y-axis Y, can be stored in a data structure of size $O(n \log n)$ such that the i distinct colors of the segments that are intersected by a query line can be reported in time $O(\log^2 n + i)$.* $\square$

We now discuss the two special cases mentioned before. If a segment $s \in S$ merely touches Y, then in the dual doublewedge one of the bounding lines is parallel to the x-axis. Consider the right-wedge w of this doublewedge. The claim that q' is in w iff r intersects the boundary of w is still true. Moreover, when we compute the left-envelope, properties (i) and (ii) above still hold. Thus the given algorithm applies unchanged.

We can handle segments that are completely contained in Y as follows. Let $\bar{S}$ be the set of such segments (intervals on Y) and let p be the point where the query line q intersects Y. Clearly, q intersects a segment of $\bar{S}$ iff p is contained in the corresponding interval on Y. Thus our problem reduces to a generalized 1-dimensional point enclosure searching problem. This problem has been solved in [JL93] in $O(n)$ space and $O(\log n + i)$ query time. Thus the bounds of Lemma 3.6 are unaffected.

What if the segments of S do not all intersect Y? Suppose that there is a constant K such that each segment intersects one of K fixed lines $Y_1, \ldots, Y_K$. We extend the above approach as follows:

Let $S_i \subseteq S$ be the set of segments intersecting Y_i, $1 \leq i \leq K$. If a segment intersects more than one Y_i, we put it in any one of the S_i's; thus the S_i's partition S. For $1 \leq i \leq K$, we create a coordinate system C_i, where Y_i is the y-axis and any line perpendicular to Y_i is taken as the x-axis. We give the segments of S_i coordinates in C_i and store them in an instance of the data structure of Lemma 3.6. To answer a query, we query each of the K structures separately. Since K is a constant, each intersected color is reported only $O(1)$ times and so the query time remains $O(\log^2 n + i)$. Similarly, the space remains $O(n \log n)$.

We are now ready to solve the problem where S consists of colored line segments each of length at least λ and all lying in $\mathcal{U}$. Wlog assume that the origin is at the bottom-left corner of $\mathcal{U}$ (otherwise re-position the origin). Consider the $K = 2 + 2\lceil\sqrt{2}/\lambda\rceil$ lines $x = i \cdot \lambda/\sqrt{2}$ and $y = i \cdot \lambda/\sqrt{2}$, where $0 \leq i \leq \lceil\sqrt{2}/\lambda\rceil$. Since each segment has length at least λ, either its x-span or its y-span is at least $\lambda/\sqrt{2}$. Thus each segment intersects one of the K lines. We now use the structure discussed earlier. We conclude:

Theorem 3.2 *Let $\lambda > 0$ be a constant and let $\mathcal{U}$ be the unit square. A set S of n colored line segments in $\mathcal{R}^2$, where each segment has length at least λ and all segments lie in $\mathcal{U}$, can be stored in a structure of size $O(n \log n)$ such that the i distinct colors of the segments that* *are intersected by a query line q can be reported in time $O(\log^2 n + i)$.* $\square$

3.3 Querying colored line segments with a line segment

3.3.1 Vertical query line segments

Our approach is similar to the one described at the beginning of Section 3.1. However, there is a subtle problem that must be overcome now since we are dealing with line segments rather than lines.

We draw vertical lines through the endpoints and the intersection points of the segments of S and obtain $O(n + \chi)$ vertical strips, where χ, $0 \leq \chi \leq \binom{n}{2}$, is the number of pairwise intersections. Within any strip, the segments that cross it can be totally ordered. We sweep over the strips starting at the leftmost non-empty strip. Let $s_1, s_2, \ldots, s_m$ be the segments that cross this strip, sorted from bottom to top. Note that not all segments of S are present in $s_1, s_2, \ldots, s_m$, which is unlike the case for lines—this is where the problem alluded to above arises. For $1 \leq i \leq m$, we give s_i a label $l(s_i) = i$ and give this label the color of s_i. We store the segments $s_1, \ldots, s_m$ in this order in a partially persistent red-black tree T_S. (Again as in Remark 3.1, we do not make the red-black color information persistent.) With each segment, we store its label. We also store the colored labels $l(s_i)$, $1 \leq i \leq m$, in a partially persistent version T_l of the data structure of [GJS93a] for the generalized 1-dimensional range reporting problem.

Suppose we sweep from the ith to the $(i + 1)$th strip. There are three possible cases:
Case 1: We encounter the left endpoint of segment s. In the current version of T_S, we locate the segment s. Let t and u be the segments that are immediately below and above s in the $(i + 1)$th strip, respectively. Then we insert s into the current version of T_S and store with it the label $l(s) = (l(t) + l(u))/2$. (If t does not exist, then $l(s) = l(u) - 1$. If u does not exist, then $l(s) = l(t) + 1$.) Moreover, we give the label $l(s)$ the same color as s and insert this colored number into the current version of the structure T_l.
Case 2: We encounter the right endpoint of segment s. We delete s from the current version of T_S and delete the colored label $l(s)$ from the current version of T_l.
Case 3: We encounter the intersection point of the segments s and t. In the current version of T_S, we interchange the order of s and t and also interchange their labels. In the current version of T_l we interchange the colors of s and t. (Each interchange operation can be simulated by two deletions and two insertions.)

It follows from the given algorithm that the labels of the segments that cross any strip increase if we visit these segments from bottom to top within the strip.

Now let q be a vertical query segment. We locate the strip containing q and then search in the version of T_S corresponding to this strip for the lowest and highest segments s and t that intersect q. Finally, we search in the version of T_l corresponding to this strip for the distinct colors of all labels that are contained in the interval $[l(s), l(t)]$.

It is easy to see that the query algorithm is correct. The ephemeral versions of T_S (resp. T_l) have size $O(n)$, a query time of $O(\log n)$ (resp. $O(\log n + i)$) and undergo $O(1)$ (resp. $O(\log n)$) memory modifications per update. Moreover, both structures are of bounded in-degree. Therefore, since we perform $O(n + \chi)$ updates to build our data structure, the final structure has size $O((n + \chi) \log n)$. A query takes $O(\log n + i)$ time.

Unfortunately, this method has a major drawback. Because of the way we label the segments we may end up getting labels consisting of $\Theta(n)$ bits. To overcome this, we use a labeling scheme due to Dietz and Sleator [DS87]. Using their approach we take integer labels in the range $[0..O(n^2)]$, i.e., labels consisting of only $O(\log n)$ bits. Consider Case 1 again. We need to give segment s a label that lies in between $l(t)$ and $l(u)$. Using the scheme of [DS87], this may result in the relabeling of other segments. Dietz and Sleator show how to choose the labels such that only $O(1)$ amortized relabelings are necessary per update. (In fact, they even give an $O(1)$ worst-case scheme. For our application, an amortized number of relabelings is good enough.)

If we relabel segment s from $l(s)$ to $l'(s)$, then we just delete the colored number $l(s)$ from T_l and insert the number $l'(s)$, having the same color as $l(s)$, into it.

It follows that the total number of updates in T_S and T_l to build the complete data structure is still $O(n + \chi)$. As a result, the structure still has size $O((n + \chi) \log n)$ and a query time of $O(\log n + i)$.

Theorem 3.3 *A set S of n colored line segments in the plane can be preprocessed into a data structure of size $O((n + \chi) \log n)$ such that the i distinct colors of the segments intersected by a vertical query line segment q can be reported in $O(\log n + i)$ time. Here χ, $0 \leq \chi \leq \binom{n}{2}$, is the number of pairwise intersections among the segments in S.*

3.3.2 Arbitrary query line segments

If q is non-vertical, then we use a different approach which yields the same space bound as Theorem 3.3 but has a slightly higher query time. In preprocessing, we break up the segments at their χ intersection points to obtain a set of $O(n + \chi)$ segments that are non-intersecting (but possibly touching). For convenience, we continue to call the resulting set S.

We store the x-projections of the segments of S in a segment tree T. For any $v \in T$, define $I(v)$ and *Strip*(v) as in Section 2.1. Let $S(v)$ be the segments of S allocated to v. Since the segments of $S(v)$ are non-intersecting, within *Strip*(v) they can be totally ordered, say, from bottom to top, as $s_1, s_2, \ldots, s_m$, where $m = |S(v)| = O(n + \chi)$. We store $S(v)$ according to this total order in a balanced search tree $B(v)$.

Let us first consider how to answer queries with a segment $l = \overline{ab}$ such that a is below b and $a, b \in$ *Strip*(v) for some node v. Let s_u (resp. s_w) be the lowest (resp. highest) segment of $S(v)$ that is above (resp. below) a (resp. b). Then, since the segments of $S(v)$ all cross *Strip*(v), the ones that are intersected by l are precisely $s_u, s_{u+1}, \ldots, s_w$. Thus to report the distinct colors of the segments of $S(v)$ that are intersected by l, we merely need to solve the generalized 1-dimensional range reporting problem for a sequence $1, 2, \ldots, m$ of colored integers, where integer p gets the color of s_p, using the query interval $[u, w]$. Thus we can answer the query for l in $S(v)$ in $O(m)$ space and $O(\log m + i)$ query time. At v, we store an auxiliary structure to answer this query.

Now suppose that we are given any query segment q and wish to report the i distinct colors of the segments of S intersected by q. We can determine in $O(\log n)$ time a set V of $O(\log n)$ nodes in T such that $\bigcup_{v \in V} S(v)$ includes all the segments of S that q intersects and, moreover, for each $v \in V$ one of the following is true: (i) q is contained completely in *Strip*(v), (ii) q is contained partially in *Strip*(v), or (iii) q crosses *Strip*(v). (The set V can be found as follows: Let $q' = [l', r']$ be the x-projection of q. We locate the leaf v_1 (resp. v_2) of T such that $l' \in$ *Strip*(v_1) (resp. $r' \in$ *Strip*(v_2)). We clip from q' the parts (if any) that partially overlap *Strip*(v_1) and *Strip*(v_2). Let q'' be the part of q' that is left. We mimic the insertion algorithm for segment trees and determine the set V' of nodes to which q'' gets allocated. The set V is $V' \cup \{v_1, v_2\}$. Its size is $O(\log n)$ since T has height $O(\log(n + \chi)) = O(\log n)$.) For any $v \in V$ we let l_v be the subsegment of q that is contained in *Strip*(v). Since the endpoints of l_v are contained in *Strip*(v), l_v behaves like the segment l above and so we query the auxiliary structure stored at v with l_v.

Theorem 3.4 *A set S of n colored line segments in $\mathcal{R}^2$ can be preprocessed into a data structure of size $O((n + \chi) \log n)$ so that the i distinct colors of the segments that are intersected by a query segment q can be reported in $O(\log^2 n + i \log n)$ time. Here χ, $0 \leq \chi \leq \binom{n}{2}$ is the number of pairwise intersections between segments in S.*

Proof At any $v \in V$, the query time is $O(\log |S(v)| + i) = O(\log n + i)$, since $|S(v)| = O(n + \chi)$. This implies the claimed bound for the query time. Each segment of S can get stored in $O(\log n)$ nodes of T. Since the supporting structures at each node v of T (i.e., $B(v)$

and the structure for generalized 1-dimensional range searching) have size linear in $|S(v)|$, it follows that the total space is $O((n+\chi)\log n)$. $\square$

4 Generalized triangle stabbing

In this section we consider the following problem: "Preprocess a set S of n colored triangles in $\mathcal{R}^2$, so that the i distinct colors of the triangles stabbed by any query point q can be reported efficiently."

We first divide each triangle $t \in S$ that does not have a horizontal side into two such triangles by drawing a horizontal line through its second highest vertex. We then group the vertices of the triangles, from left to right, into $\Theta(n^{1/2})$ vertical strips each of size $\Theta(n^{1/2})$ (except possibly for the last strip, which may contain fewer vertices).

Let V_i be any strip. The triangles that intersect V_i form two disjoint subsets T_i and T_i', where T_i (resp. T_i') consists of triangles having no (resp. at least one) vertex inside V_i. We further subdivide V_i into vertical strips by taking each triangle in T_i' and drawing a vertical line through each of its vertices that lies in V_i. Let W_{ij} be any such substrip within V_i and let T_{ij} consist of the triangles of T_i' that cross W_{ij}. Note that, by construction, no triangle of T_{ij} can have a vertex inside W_{ij}. (This partitioning technique is reminiscent of a method used in [OY88] for computing the measure of the union of iso-oriented boxes in $\mathcal{R}^d$.)

Given the query point q, suppose that $q \in V_i$ and $q \in W_{ij}$. Then we need to only report the distinct colors of the triangles of T_i and of T_{ij} that are stabbed by q. We discuss how to do this for V_i and T_i. (The discussion for W_{ij} and T_{ij} is similar.)

Let $t \in T_i$ be any triangle. Let $h(t)$ be t's horizontal side and let $s(t)$ be the slanted side of t which crosses V_i. Let $p(t)$ be the vertex shared by $h(t)$ and $s(t)$ and let $\bar{h}(t)$ and $\bar{s}(t)$ be the rays that emanate from $p(t)$ and contain $h(t)$ and $s(t)$, respectively. Let $w(t)$ be the wedge defined by $\bar{h}(t)$ and $\bar{s}(t)$. T_i can be partitioned into two sets L_i and R_i, where L_i (resp. R_i) consists of triangles t such that $p(t)$ is on or to the left of (resp. on or to the right of) the left (resp. right) boundary of V_i. Let l_q (resp. r_q) be the horizontal leftward (resp. rightward) ray emanating from q. It is easy to see that q stabs $t \in L_i$ (resp. $t \in R_i$) iff l_q (resp. r_q) intersects $w(t)$. Thus, we need to report the distinct colors of the wedges $w(t)$, $t \in L_i$, that are intersected by l_q and symmetrically for R_i and r_q. Each of these problems can be solved by computing left- and right-envelopes and using a segment tree, as in Section 3.2.

Theorem 4.1 *A set S of n colored triangles in $\mathcal{R}^2$ can be stored in a data structure of size $O(n^{3/2}\log n)$ such that the i distinct colors of the triangles that are stabbed by a query point q can be reported in $O(\log^2 n + i)$ time.*

Proof The correctness of the method is clear. We now consider the space bound. We have $|T_i| = O(n)$ for all i. Thus, $|L_i|$ and $|R_i|$ are each $O(n)$. It now follows that the segment-tree-based data structure for L_i and R_i (hence for T_i) uses $O(n \log n)$ space. Thus the space for all the T_i is $O(n^{3/2} \log n)$. Consider the structure for T_{ij}. We have $|T_i'| = \Theta(n^{1/2})$, since each V_i has size $\Theta(n^{1/2})$. Thus, $|T_{ij}| = O(|T_i'|) = O(n^{1/2})$ and so the structure for $|T_{ij}|$ uses $O(n^{1/2} \log n)$ space. There are $\Theta(n^{1/2})$ strips within each W_{ij} (since $|V_i| = \Theta(n^{1/2})$) and a total of $\Theta(n)$ such strips taken over all V_i. Thus the total space for all the T_{ij}'s is $O(n^{3/2} \log n)$.

The query algorithm consists of four ray–envelope intersection queries, each of which takes $O(\log^2 n + i)$ time. □

Using Theorem 4.1 we can now solve the following problem, which was mentioned at the beginning of Section 3.2: "Preprocess a set S of n colored line segments in $\mathcal{R}^2$ so that the i distinct colors of the segments that are intersected by a query line q can be reported efficiently."

Using $\mathcal{F}$ we dualize each segment $s \in S$ to a doublewedge and then break each doublewedge into two wedges (i.e., infinite triangles) in the obvious way. Our problem is now equivalent to reporting the distinct colors of the wedges that are stabbed by $\mathcal{F}(q)$.

We enclose the wedges in a sufficiently large box B such that there are no intersections of wedges outside B. (For example, the box defined by the leftmost, rightmost, topmost, and bottommost intersections will do.) Inside B we have $2n$ finite triangles, which we process as in Theorem 4.1. Also, if we delete the parts of the wedges lying inside B, then B and the parts of the wedges lying outside B define a planar subdivision $\mathcal{P}$. We preprocess $\mathcal{P}$ for fast planar point location [ST86]. Note that each face of $\mathcal{P}$ that is outside B is either covered by no wedge or is covered by exactly one wedge, since there are no intersections outside B. With each face of the latter type we associate the color of the corresponding wedge.

Given q, we determine if $\mathcal{F}(q)$ is inside or outside B. If inside, then we apply Theorem 4.1; if outside, then we locate the face of $\mathcal{P}$ that contains q and read off the color (if any) associated with it.

Corollary 4.1 *A set S of n colored line segments in $\mathcal{R}^2$ can be stored in a data structure of size $O(n^{3/2} \log n)$ such that the i distinctly-colored segments that are intersected by a query line can be reported in time $O(\log^2 n + i)$.*

Proof Correctness is clear. $\mathcal{P}$ has size $O(n)$ and so the point location structure uses $O(n)$ space and answers queries in $O(\log n)$ time. This, together with Theorem 4.1, implies the claimed bounds. □

4.1 Generalized stabbing queries on fat-wedges

Let $\gamma > 0$ be a constant. A wedge is a *fat-wedge* if the angle between its bounding lines is at least γ. We show how to preprocess a set S of n colored fat-wedges so that point-stabbing queries can be answered efficiently.

In preprocessing, we select $t = \lceil 2\pi/\gamma \rceil$ coordinate systems $\mathcal{C}_i = (x_i y_i)$, where all the $\mathcal{C}_i$ share the same origin and $\mathcal{C}_{i+1}$ is offset from $\mathcal{C}_i$ by an angle γ, $0 \leq i \leq t-1$ (indices are taken modulo t). Each fat-wedge $w \in S$ is y_i-monotone for at least one i. Specifically, if the bounding rays r' and r'' of w make angles α' and α'' with the positive x_0-axis (our frame of reference is $\mathcal{C}_0$), where $\alpha'' < \alpha'$ and $\alpha' - \alpha'' \geq \gamma$, then w is y_i-monotone for $i = \lceil \alpha''/\gamma \rceil$.

Let S_i be the fat-wedges of S that are y_i-monotone. If a fat-wedge is y_i-monotone for more than one i, then we put it in only one of the S_i; thus the S_i's partition S. Suppose that $w \in S_i$ is a right-wedge. A query point q is contained in w iff the ray r which emanates from q in the negative x_i-direction intersects w's boundary. Symmetrically if w is a left-wedge. For each S_i, we build two instances of the data structure of Section 3.2 for y-monotone wedges, with $y = y_i$, one for the right-wedges of S_i and the other for the left-wedges of S_i. Given a query point q, we simply query the $2t$ data structures and output the distinct colors returned.

Theorem 4.2 *Let $\gamma > 0$ be a constant. A set S of n colored wedges, where the angle between the bounding lines of each wedge is at least γ, can be stored in a data structure of size $O(n \log n)$ such that the i distinct colors of the wedges that are stabbed by a query point can be reported in $O(\log^2 n + i)$ time.* $\square$

5 Generalized triangle range searching

We wish to preprocess a set S of n colored points in $\mathcal{R}^2$ so that given any query triangle q, the i distinct colors of the points lying inside q can be reported efficiently. With minor modifications, the reporting structure of Theorem 2.4, for $d = 2$, can be used to solve our problem in $O(n^{2+\epsilon})$ space and $O(\log n + i)$ query time. We leave open the question of whether a near-linear-space solution with good output-sensitive query time exists. In the rest of this section, we show how to solve the problem efficiently for fat-triangles. A *fat-triangle* is a triangle in which each internal angle is at least γ for some positive constant γ.

We begin with a solution for a query fat-wedge q, i.e., a wedge where the angle between the bounding lines is at least γ. Let v_q be the vertex of q. For now assume that q is

y-monotone, i.e., any horizontal line intersects q exactly once. (Later we will remove this assumption.) We store the points of S at the leaves of a balanced binary search tree T by non-decreasing y-coordinates from left to right. We augment each node v of T with an instance $HP(v)$ of the structure of Theorem 2.1 for generalized halfplane range reporting; $HP(v)$ is built on the points in v's descendant leaves.

Given q, we divide it into two wedges q_a and q_b, each with a horizontal side, by drawing a horizontal line L through v_q. This is always possible because q is y-monotone. Here q_a (resp. q_b) lies above (resp. below) L. Let r_a (resp. r_b) be the ray of q_a (resp. q_b) that also belongs to q and let l_a (resp. l_b) be the line supporting r_a (resp. r_b). (See Figure 1.) We search in T using the y-coordinate of v_q and determine sets V_a and V_b of nodes, where V_a (resp. V_b) consists of the nodes of T that are right (resp. left) children of nodes on the search path but are not themselves on the search path. We query $HP(v)$ at each $v \in V_a$ (resp. $v \in V_b$) with the halfplane l_a^- (resp. l_b^+).

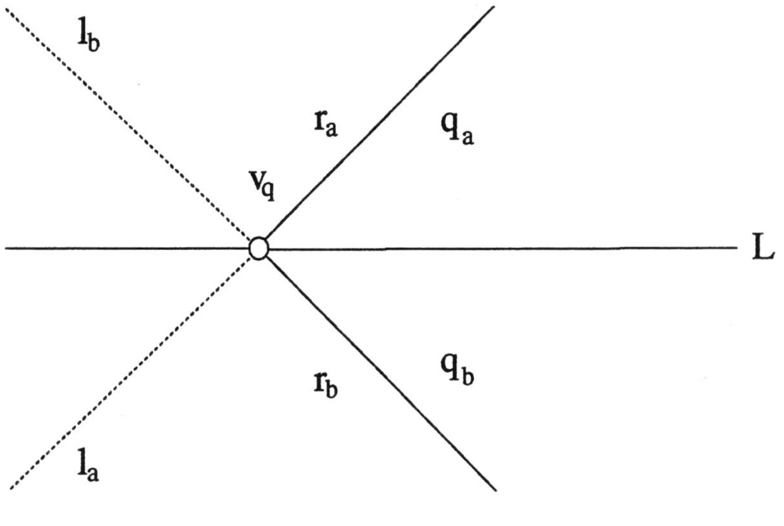

Figure 1: Range searching with a fat-wedge

Lemma 5.1 *A set S of n colored points in $\mathcal{R}^2$ can be stored in a data structure of size $O(n \log^2 n)$ such that the i distinct colors of the points that are contained in a query fat-wedge q that is y-monotone can be reported in time $O(\log^3 n + i \log n)$.*

Proof For each $v \in V_a$, the points in the descendant leaves of v are all above L. Moreover, each point of S that is above L is stored in a leaf of the subtree of exactly one node $v \in V_a$.

Of these points, the ones in q_a (hence in q) are those lying in l_a^-. By Theorem 2.1, the query on $HP(v)$ with l_a^- returns the colors of these points. Symmetrically for q_b.

Each level of T uses $O(n \log n)$ space by Theorem 2.1 and so the total space is $O(n \log^2 n)$. The query time at each node visited is $O(\log^2 n + i)$, from which the claimed time bound follows. $\square$

What if q is not y-monotone? In preprocessing, we select $t = \lceil 2\pi/\gamma \rceil$ coordinate systems $\mathcal{C}_i = (x_i y_i)$, where all the $\mathcal{C}_i$ share the same origin and $\mathcal{C}_{i+1}$ is offset from $\mathcal{C}_i$ by an angle γ, $0 \leq i \leq t-1$. Within each $\mathcal{C}_i$ we build an instance of the data structure of Lemma 5.1 for y_i-monotone fat-wedges. Given a query fat-wedge q, we locate a $\mathcal{C}_i$ such that q is y_i-monotone and then query the associated structure.

Lemma 5.2 *Let $\gamma > 0$ be a constant. A set S of n colored points in $\mathcal{R}^2$ can be stored in a data structure of size $O(n \log^2 n)$ such that the i distinct colors of the points that are contained in a query wedge q whose internal angle is at least γ can be reported in time $O(\log^3 n + i \log n)$.* $\square$

We are now ready to describe the algorithm for a query fat-triangle. We store the points by non-decreasing x-coordinates from left to right in a balanced search tree T' and augment each node v with an instance $FW(v)$ of the structure of Lemma 5.2 for fat-wedges. $FW(v)$ is built on the points in v's descendant leaves.

Given q, we divide it into at most two triangles q_l and q_r, each with a vertical side s, with q_l to the left of s and q_r to the right. We search in T' with the x-coordinate of s and identify sets V_l and V_r of nodes that lie to the left and to the right of the search path, respectively. For each node $v \in V_l$ (resp. $v \in V_r$), we query $FW(v)$ with the wedge supporting q_l (resp. q_r), which is a fat-wedge. We conclude directly:

Theorem 5.1 *Let $\gamma > 0$ be a constant. A set S of n colored points in $\mathcal{R}^2$ can be stored in a data structure of size $O(n \log^3 n)$ such that the i distinct colors of the points that are contained in a query triangle q each of whose internal angles is at least γ can be reported in time $O(\log^4 n + i \log^2 n)$.* $\square$

6 Conclusions and further work

We have presented efficient solutions to several generalized intersection searching problems involving non-iso-oriented objects. Our methods have included sparse representations, persistence, and filtering search.

Besides improving upon our bounds, three problems are of particular interest: (i) obtaining dynamic data structures for the generalized problems considered here, (ii) obtaining linear-space or near-linear space solutions with output-sensitive query times (of the form $O(n^p + i)$ or $O(n^p + i \cdot \text{polylog}(n))$, $0 < p < 1$) for the generalized halfspace range searching problem in $d \geq 4$ dimensions and for the generalized simplex range searching problem in $d \geq 2$ dimensions, and (iii) solving the counting versions of Problems 3–7 in Table 1.

References

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

[AvK93] P.K. Agarwal and M. van Kreveld. Connected component and simple polygon intersection searching. In *Proceedings of the 1993 Workshop on Algorithms and Data Structures*, pages 36–47, August 1993.

[AvKO93] P.K. Agarwal, M. van Kreveld, and M. Overmars. Intersection queries for curved objects. *Journal of Algorithms*, 15:229–266, 1993.

[CGL85] B.M. Chazelle, L.J. Guibas, and D.T. Lee. The power of geometric duality. *BIT*, 25:76–90, 1985.

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

[CJ90] S.W. Cheng and R. Janardan. Efficient dynamic algorithms for some geometric intersection problems. *Information Processing Letters*, 36:251–258, 1990.

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

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

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

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

[DS87] P.F. Dietz and D.D. Sleator. Two algorithms for maintaining order in a list. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 365–372, 1987.

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

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

[GJS93a] P. Gupta, R. Janardan, and M. Smid. Further results on generalized intersection searching problems: counting, reporting, and dynamization. In *Proceedings of the 1993 Workshop on Algorithms and Data Structures*, pages 361–372, August 1993.

[GJS93b] P. Gupta, R. Janardan, and M. Smid. On intersection searching problems involving curved objects. Technical Report TR–93–42, Dept. of Computer Science, University of Minnesota, 1993. Submitted.

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

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

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

[Mat92a] J. Matoušek. Efficient partition trees. *Discrete & Computational Geometry*, 8:315–334, 1992.

[Mat92b] J. Matoušek. Range searching with efficient hierarchical cuttings. In *Proceedings of the 8th Annual ACM Symposium on Computational Geometry*, pages 276–285, 1992.

[Mul93] K. Mulmuley. *Computational Geometry: An introduction through randomized algorithms.* Prentice–Hall, 1993.

[OY88] M.H. Overmars and C.K. Yap. New upper bounds in Klee's measure problem. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, pages 550–556, 1988.

[Sha88] M. Sharir. Davenport-schinzel sequences and their geometric applications. In R. A. Earnshaw, editor, *Theoretical Foundations of Computer Graphics and CAD*, pages 253–278. Springer Verlag, 1988.

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

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