# Algorithms for generalized halfspace range searching and other intersection searching problems ⋆

Prosenjit Gupta [1]

*Dept. of Computer Science, University of Minnesota, Minneapolis, MN 55455, U.S.A.*

Ravi Janardan [2]

*Dept. of Computer Science, University of Minnesota, Minneapolis, MN 55455, U.S.A.*

Michiel Smid [3]

*Max-Planck-Institut für Informatik, D-66123 Saarbrücken, Germany.*

**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 have many 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 objects that are not necessarily iso-oriented. 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 $\mathcal{R}^2$ for certain classes of line segments and triangles. 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.

## 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 $n$.) 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. These generalized problems have many applications—see [0,0,0,0] for some examples.

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 in the paper.

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. However, 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

[⋆] A preliminary version appears in *Proc. 10th ACM Symp. on Computational Geometry (1994)* under the title "Efficient algorithms for generalized intersection searching on non-iso-oriented objects."

[1] Email: `pgupta@cs.umn.edu`. Research supported in part by NSF grant CCR–92–00270.

[2] Email: `janardan@cs.umn.edu`. Research 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.

[3] Email: `michiel@mpi-sb.mpg.de`. This author was supported by the ESPRIT Basic Research Actions Program, under contract No. 7141 (project ALCOM II).

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, in both cases, the solution should also be space-efficient.

## *1.1 Previous work*

While the standard problems have been investigated extensively (see, for example, [0,0,0,0,0]), their generalized counterparts have been less studied. The generalized problems were first considered in [0], where efficient solutions were given for several problems defined on iso-oriented objects (i.e., the input and the query objects are axes-parallel), including intervals, points, orthogonal line segments, and iso-oriented rectangles in two or more dimensions. A solution was also given for searching on arbitrary (non-intersecting) colored line segments with an arbitrary line segment. In [0], efficient solutions were given for the counting, reporting, and dynamic versions of some of the iso-oriented problems mentioned above. In [0], solutions were given for generalized problems involving circular and circle-like objects (among other results). In [0], Agarwal and van Kreveld considered the colored line segment intersection searching problem for color classes consisting of line segments and satisfying the property that each color class is a simple polygon or a connected component. The above results from [0] can also be found in van Kreveld's Ph.D. thesis [0]. In [0], van Kreveld also gives a general approach for such colored searching problems and illustrates this approach by deriving an $O(n^{2+\epsilon})$-space and $O(\log n + i)$-query time solution for searching on arbitrary colored line segments with a query line segment. (Here $\epsilon > 0$ is an arbitrarily small constant.) The same general approach can also be used to solve the generalized halfspace (or even simplex) range reporting problem for $n$ points in $\mathcal{R}^d$ in $O(\log n + i)$ time using $O(n^{d+\epsilon})$ space.

## *1.2 Summary of results*

In this paper, we present efficient solutions to several generalized intersection searching problems that are defined on objects that are not necessarily iso-oriented. 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 $\mathcal{R}^2$ for certain classes of line segments and triangles. Our main results are summarized in Table 1.

We note the following about the table: 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

Table 1
Summary of results for generalized problems. The information in the table is discussed in more detail in Section 1.2.

| # | *Input objects* | *Query object* | *Space* | *Query time* |
|---|---|---|---|---|
| 1 | Points in $\mathcal{R}^d$ | Halfspace in $\mathcal{R}^d$ | $d = 2$ $\quad n \log n$ | $\log^2 n + i$ |
| | | | | $n^{1/2}$ |
| | | | $n \log^2 n$ | $n^{1/2+\epsilon} + i$ |
| | | | $d = 3$ $\quad n^{2+\epsilon}$ | $\log^2 n + i$ |
| | | | $n \log n$ | $n^{2/3+\epsilon}$ |
| | | | $d \geq 4$ $\quad n^{\lfloor d/2 \rfloor}/(\log n)^{\lfloor d/2 \rfloor - 1 - \epsilon}$ | $\log n + i \log^2 n$ |
| 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 $\mathcal{R}^2$ (or points in $\mathcal{R}^2$) | Vertical line seg. (or non-vert. strip) | $n^2 \log n$ | $\log n + i$ |
| | | | $n^{2-\mu/2} \log n$ | $n^\mu + i$ |
| 4 | Line segs. in $\mathcal{R}^2$ | Vertical line segment | $(n + \chi) \log n$ | $\log n + i$ |
| 5 | Line segs. of length $\geq$ a constant in unit square | Line | $n \log n$ | $\log^2 n + i$ |
| 6 | Fat-Wedges | Point | $n \log n$ | $\log^2 n + i$ |
| | Fat-Triangles | | $n^{1+1/c} \log^2 n$ | $\log^2 n + i$ |
| | | | $n \log^2 n$ | $\log^3 n + i \log n$ |
| 7 | Points | Fat-Triangle | $n \log^3 n$ | $\log^4 n + i \log^2 n$ |

meanings of the different symbols in the table are as follows: $n$: input size; $i$ : output size (number of distinct colors intersected); $\epsilon$: arbitrarily small constant $> 0$; $\mu$: adjustable parameter, $0 < \mu < 1$; $\chi$: number of pairwise-intersecting segments, $0 \leq \chi \leq \binom{n}{2}$; $\alpha(n)$: slow-growing inverse of Ackermann's function; $c$: constant $> 1$.

Also, wherever $\epsilon$ appears in a query time or space bound in the table, the corresponding space or query time bound contains a multiplicative factor which goes to $\infty$ as $\epsilon \to 0$. Moreover, the constants implied in the definition of "long" segments (Problem 5) and "fat" wedges and triangles (Problems 6 and 7) are present in the space and query time bounds for these problems.

Finally, we note that some additional results that follow immediately from the same techniques are not shown in the table; these are mentioned in the text—see Remarks 3, 7, and 12.

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. [0], which allows us to reduce a generalized query problem to a generalized query problem one dimension lower; and (3) A version of filtering search which, in combination with persistence, yields a space-query time tradeoff.

## 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.[4] Without loss of generality, we may assume that $Q$ is non-vertical since vertical queries are easy to handle.

We denote the coordinate directions by $x_1, x_2, \ldots, x_d$. (For convenience, when discussing our solution in $\mathcal{R}^2$, 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 $x_d = p_1 x_1 + \cdots + p_{d-1}x_{d-1} - p_d$. If $H : x_d = a_1 x_1 + \cdots + a_{d-1}x_{d-1} + a_d$ 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 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

[4] 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.

*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. Its 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 $E_c$, called *facets*. 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 $\mathcal{E}$ on 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 $T$. We associate with $v$ 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*$(\textit{parent}(v))$. 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 [0] (resp. [0]). 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 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, [0, 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 $T_c$ be 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) on 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 [0], 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 [0] (see also [0,0]), 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 analyze 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 [0, 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 [0] 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 [0] 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 [0, Theorem 5.8(i), page 69]). For $D_3$, we use the above-mentioned structure of [0]. 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 [0], 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** *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 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. $O(m^{2+\epsilon})$), 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 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. $\square$

**An application:** Consider the *generalized disk range searching* problem: "Given a set $S$ of $n$ colored points in $\mathcal{R}^2$, report or count the $i$ distinct colors of the points lying inside a variable-radius query disk $q$." Using the well-known lifting transformation [0], 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.

**Remark 3** In addition to the bounds given in Theorem 2, other bounds are also possible. For instance, the reporting problem is solvable using partition trees and a version of $D_3$ from [0]; 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 [0, 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.

## 2.3 Handling 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 $\mathit{Strip}(u)$ and $\mathit{Strip}(w)$. 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 $\mathit{Strip}(u)$ and in $\mathit{Strip}(w)$ 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 are unaffected.

## *2.4 Generalized halfspace range searching in $d \geq 4$ dimensions*

The approach of Section 2.2 can be extended, with some modifications, to any fixed $d \geq 4$ also. However, the bounds are not satisfactory—$O(n^{d\lfloor d/2\rfloor+\epsilon})$ space and logarithmic query time (using cutting trees) or near-linear space and superlinear query time (using partition trees). Intuitively, the latter 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 now describe a different approach for the reporting problem in $\mathcal{R}^d$, $d \geq 4$. 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 $O(n_v^{\lfloor d/2\rfloor}/(\log n_v)^{\lfloor d/2\rfloor-\epsilon})$ space and has query time $O(\log n_v)$ [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 4** *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 $O(n^{\lfloor d/2\rfloor}/(\log n)^{\lfloor d/2\rfloor-1-\epsilon})$ 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 $O(n^{\lfloor d/2 \rfloor}/(\log n)^{\lfloor d/2 \rfloor - \epsilon})$ space per level and there are $O(\log n)$ levels, which gives the stated space bound. The query time can be upper-bounded as follows: If $i = 0$, then the query on $HSE(\text{root})$ 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. □

## *2.5 Extensions*

We can extend the approach of Section 2.1 to also obtain the following result:

**Theorem 5** *Let $S = \{s_1, s_2, \ldots, s_n\}$ be $n$ colored line segments in the plane. The $i$ distinct colors of the segments that are contained completely in a query halfplane $Q^-$ be reported (resp. counted) in $O(\log^2 n + i)$ (resp. $O(n^{1/2})$) time, using $O(n \log n)$ space. Similarly, the $i$ distinct colors of the segments that are intersected by a vertical query ray $r$ can be reported (resp. counted) in time $O(\log^2 n + i)$ (resp. $O((n\alpha(n))^{1/2})$), using $O(n\alpha(n) \log n)$ space.*

Briefly, the approach is as follows: By duality, the first problem becomes: "Count or report the distinct colors of the double-wedges $w_i = \mathcal{F}(s_i)$ 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 [0, 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 $E_c$. 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.

As for the second problem, 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 $|E_c| = O(m_c \alpha(m_c))$ [0]. Also, properties analogous to (i) and (ii) above hold and so we can proceed as in Section 2.1.

## 3 Generalized intersection searching on lines and line segments

### *3.1 Querying colored lines with a vertical line segment*[5]

We give a simple approach based on persistence [0], which has query time $O(\log n + i)$ and uses $O(n^2 \log n)$ space. In Section 3.1.1, we show how to modify this solution to get a query time of $O(n^\mu + i)$ using $O(n^{2-\mu/2} \log n)$ space, for any $\mu$ in the range $0 < \mu < 1$, thus getting a (nearly) continuous trade-off between query time and space.

Let $\mathcal{A}$ be the arrangement of the $n$ lines of $S$ (the input set). We divide the plane into $t+1$ strips, $V_1, V_2, \ldots, V_{t+1}$, by drawing vertical lines through the $t = O(n^2)$ vertices of $\mathcal{A}$. Within any strip, $V_k$, the lines can be totally ordered from top to bottom, as $E_k : \ell_1, \ell_2, \ldots, \ell_n$. We store $E_k$ in a balanced search tree $T_k$. Suppose that the vertical query segment $q$ is in $V_k$ and let $\ell_a$ and $\ell_b$ be the highest and lowest lines of $E_k$ intersected by $q$. Our problem is now equivalent to the following *generalized 1-dimensional range searching* problem: Given colored integers $1, 2, \ldots, n$ (where integer $j$ gets the color of $\ell_j$), report the distinct colors in the query interval $[a, b]$. In [0], this problem is solved using a structure $D_k$ of size $O(n)$ and a query time $O(\log n + i)$. $D_k$ supports updates in $O(\log n)$ time with $O(\log n)$ memory modifications. It is now clear that we can report the $i$ distinct colors of the lines intersected by $q$ in $O(\log n + i)$ time using $O(n^3)$ space.

We can reduce the space to $O(n^2 \log n)$ by sweeping over the strips and applying persistence. Specifically, 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 $V_k$, which can be done via two insertions and two deletions. Similarly, we also update $T_{k-1}$ by interchanging $l'$ and $l''$ to get $T_k$. Since both the $D$-structure and the $T$-structure have con-

[5] Note that, by duality, this problem is equivalent to searching in a planar set of colored points for the distinct colors contained in the non-vertical strip enclosed by two parallel straight lines. Indeed, this was the original motivation for studying the problem since it is a natural generalization of the colored halfplane range searching problem considered earlier.

stant in-degree (see [0] for the definition of "in-degree"), the space used per update is $O(\log n)$ for a $D$-structure and $O(1)$ for a $T$-structure, 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 $k$th version of $\mathcal{T}$ to find $[a, b]$, and then query the $k$th version of $\mathcal{D}$ with this.

**Theorem 6** *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.*

**Remark 7** Similarly, we can solve the counting problem in $O(n^2 \log^2 n)$ space and $O(\log^2 n)$ query time by using the counting version of $D_k$ from [0].

### *3.1.1 A space-query time trade-off*

In a nutshell, the idea is as follows: We extract from the sequence $\mathcal{E} = (E_1, \ldots, E_{t+1})$ a smaller subsequence $\mathcal{E}' = (E'_1, \ldots, E'_m)$ such that (i) $E'_j$ and $E'_{j+1}$ differ in just two (not necessarily adjacent) positions, (ii) for each $E_k \in \mathcal{E}$ there is an $E'_j \in \mathcal{E}'$ which "approximates" $E_i$ in a sense that we will elaborate upon later, and (iii) $m = O(n^{2-\mu/2})$. Properties (i) and (iii) suggest that we can apply persistence to $\mathcal{E}'$ and get a scheme with space bound $o(n^2)$; property (ii) suggests that instead of querying $E_k$, as we might in the simple scheme, we can query $E'_j$ in the new scheme and still be assured of correctness.

We remark that except for two key differences, this approach is similar to one used in [0,0] for a different (standard) problem. (These differences have to do with the "swap criterion" we use to construct $\mathcal{E}'$ and with our choice of so-called "borders" (see below).) Therefore, we will describe our solution only briefly here. The interested reader can find a full discussion in [0].

Formally, we define a sequence $b_1, b_2, \ldots, b_B$ of $B = \Theta(n^{1-\mu})$ distinguished list positions called *borders*, where $b_1 = 1$, $b_B = n$, and, for $i = 2, \ldots, B-1$, $b_i = b_{i-1} + \lfloor n^\mu + 1 \rfloor$. To construct $\mathcal{E}'$ we scan $\mathcal{E}$ from left to right. Let $E_i$ be the currently-scanned list of $\mathcal{E}$. Let $E'_j$ be the most recently constructed list of $\mathcal{E}'$ and suppose that we constructed $E'_j$ when we reached $E_{\delta_j} \in \mathcal{E}$. If $E_{i+1}$ is obtained from $E_i$ by swapping lines across some border $b_k$, i.e. by swapping the $b_k$th line $\alpha$ with either the $(b_k - 1)$th line $\beta$ or the $(b_k + 1)$th line $\gamma$, then we create $E'_{j+1}$ from $E'_j$ by swapping $\alpha$ with $\beta$ or $\gamma$, as appropriate, set $\delta_{j+1} = i + 1$, and move on to $E_{i+1}$. The following properties of $\mathcal{E}'$ are easily shown:

**Lemma 8** *Suppose that $\mathcal{E}' = (E'_1, \ldots, E'_m)$ has been constructed as described above from $\mathcal{E} = (E_1, \ldots, E_{t+1})$. Then (i) $E'_j$ approximates any list $E_k \in \{E_{\delta_j}, E_{\delta_j+1}, \ldots, E_{\delta_{j+1}-1}\}$ in the following sense: for any two successive borders $b_l$ and $b_{l+1}$, 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 $E_k$ at these same positions. And (ii) The border-lines in $E'_j$ and $E_k$ are the same and this implies that the border-lines in $\{E_{\delta_j}, E_{\delta_j+1}, \ldots, E_{\delta_{j+1}-1}\}$ are the same and can be totally ordered.*

**Lemma 9** *$m = |\mathcal{E}'| = O(n^{2-\mu/2})$.*

**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 $b_k - 1$ or $b_k - 2$ lines above $v$. Let $S'$ be the set of $n$ planar points that are dual to the lines of $S$. By duality, the $b_k - 1$ (or $b_k - 2$) lines above $v$ correspond to a $(b_k - 1)$-*set* (or a $(b_k - 2)$-*set*) [0] of $S'$. Let $f_j(n)$ be the maximum number of $j$-sets realized by a set of $n$ points in the plane, $0 \le j \le n$. Then the number of $b_k$-swaps that occur during the construction of $\mathcal{E}'$ is $O(f_{b_k-1}(n) + f_{b_k-2}(n))$. Thus, $m = O(\sum_{k=1}^{B} f_{b_k-1}(n) + f_{b_k-2}(n))$.

We will show that $\sum_{k=1}^{B} f_{b_k-1}(n) = O(n^{2-\mu/2})$. (A similar proof applies to $\sum_{k=1}^{B} f_{b_k-2}(n)$.) Let $T = \bigcup_{k=1}^{B}\{b_k - 1\} = \{0, t, 2t, \ldots, (B-1)t, n\}$, where $t = \lfloor n^{\mu} + 1 \rfloor$. Let $T_1 = \{t, 2t, \ldots, rt\}$ and $T_2 = \{(r+1)t, \ldots, n\}$, where $rt$ is the largest integer in $T$ that is no more than $n/2$. Thus, $\sum_{k=1}^{B} f_{b_k-1}(n) = 1 + \sum_{j\in T_1} f_j(n) + \sum_{j\in T_2} f_{n-j}(n)$ (since $f_0(n) = 1$ and $f_j(n) = f_{n-j}(n)$). By a result of Welzl [0, Theorem 1], we have $\sum_{j\in T_1} f_j(n) = O(n(\sum_{j\in T_1} j)^{1/2}) = O(n^2/t^{1/2}) = O(n^{2-\mu/2})$. Similarly, $\sum_{j\in T_2} f_{n-j}(n) = O(n^{2-\mu/2})$, which completes the proof. □

The data structure consists of (1) a red-black tree $T'_j$ storing the total order of the border lines of $E'_j$, (2) an instance $D'_j$ of the generalized 1-dimensional range searching structure built on the colored integers $I_j : (1, 2, \ldots, n)$, where integer $p$ gets the color of the $p$th line of $E'_j$, and (3) a red-black tree $I'_j$ storing $I_j$. We make all these structures partially-persistent using the method of [0]. The space used by the overall structure can be shown to be $O(n^{2-\mu/2}\log n)$.

Given $q$, suppose we need to query $E'_j$. We search in $T'_j$ and find the smallest border $b_s$ on or above $q$'s upper endpoint and, symmetrically the greatest border $b_g$. We query $D'_j$ with $[b_s + 1, b_g]$. Then we scan all the integers $b_s, b_s - 1, \ldots, b_{s-1} + 1$ in $I'_j$ and report the distinct colors of the lines of $E'_j$ at these positions that are intersected by $q$; an almost symmetric discussion applies to $b_g$. The correctness of the query follows from Lemma 8 and the correctness of $D'_j$ [0]. The query time is upper-bounded by the time to search in $D'_j$ (which is $O(\log n + i)$) and the time to scan $I'_j$ (which is $O(n^{\mu})$).

**Theorem 10** *A set $S$ of $n$ colored lines in $\mathcal{R}^2$ can be stored in a data structure of size $O(n^{2-\mu/2}\log n)$ 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 $\mu$ is an adjustable parameter in the range $0 < \mu < 1$.*

### *3.2 Querying colored line segments with a vertical line segment*

We can use an approach similar to the one underlying Theorem 6. However, since we are now dealing with line segments rather than lines, we must overcome a subtle problem that can arise. We discuss this later. We draw vertical lines through the endpoints and the $\chi \leq \binom{n}{2}$ intersection points of the segments of $S$. 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. 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 $T_S$. 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 [0] for the generalized 1-dimensional range reporting problem.

Suppose we sweep from the $i$th to the $(i+1)$th strip. There are three cases: (i) We encounter the left endpoint of segment $s$. In the current version of $T_S$, we locate $s$. Let $t$ and $u$ be the segments that are immediately below and above $s$ in the $(i+1)$th strip. We insert $s$ into the current version of $T_S$ and store with it a label $l(s)$ that lies between $l(t)$ and $l(u)$. Moreover, we give the label $l(s)$ the same color as $s$ and insert this colored number into the current version of $T_l$. (ii) 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 $T_l$. (iii) 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 $t$.

In this scheme, we need to assign special labels to the segments as we encounter them because all the segments are not present in each strip. However, we must be careful in choosing the labels since otherwise we may end up getting labels consisting of $\Theta(n)$ bits. Towards this end, we use a labeling scheme due to Dietz and Sleator [0]. Using their approach we take integer labels in the range $[0..O(n^2)]$, i.e., labels consisting of only $O(\log n)$ bits. We need to give segment $s$ a label that lies in between $l(t)$ and $l(u)$. Using the scheme of [0], 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. 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 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. Moreover, the total number of updates done in $T_S$ and $T_l$ is $O(n+\chi)$, which implies that the total space is $O((n+\chi)\log n)$.

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 $[l(s), l(t)]$. The query time is clearly $O(\log n + i)$.

**Theorem 11** *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 $\chi$, $0 \leq \chi \leq \binom{n}{2}$, is the number of pairwise intersections among the segments in $S$.*

**Remark 12** Similarly, by using the counting version of $T_l$ from [0], we can solve the counting problem in $O((n+\chi)\log^2 n)$ space and $O(\log^2 n)$ query time.

## *3.3 Querying "long" colored line segments in the unit square with a line*

In this section, we consider the case where the segments of $S$ all lie in the unit square $\mathcal{U}$ and each segment is "long" in the sense that it has length at least a constant $\lambda > 0$. The query object, $q$, is a line. 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. 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 double-wedge one of the bounding lines has positive slope and the other has negative slope.

We split each double-wedge 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. (Left-wedges are symmetric.) Because of $y$-monotonicity, the query point $q' = \mathcal{F}(q)$ 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 [0, page 377, 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.

**Lemma 13** *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 $O(\log^2 n + i)$.*

We now discuss the two special cases mentioned before. If a segment $s \in S$ merely touches $Y$, then in the dual double-wedge one of the bounding lines is parallel to the $x$-axis. Consider the right-wedge $w$ of this double-wedge. 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 [0] in $O(n)$ space and $O(\log n + i)$ query time. Thus the bounds of Lemma 13 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 $\mathcal{C}_i$, 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 $\mathcal{C}_i$ and store them in an instance of the data structure of Lemma 13. 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 $\lambda$ and all lying in $\mathcal{U}$. Without loss of generality, 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 $\lambda$, 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.

**Theorem 14** *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 $\lambda$ 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 $O(\log^2 n + i)$.*

## 4 Generalized fat-wedge and fat-triangle stabbing

Call a wedge a *fat-wedge* if the internal angle at its vertex is at least a constant $\gamma > 0$. Similarly, call a triangle a *fat-triangle* if each internal angle is at least $\gamma$. In this section we consider the following problem: "Preprocess a set $S$ of $n$ colored fat-wedges (resp. fat-triangles) in $\mathcal{R}^2$, so that the $i$ distinct colors of the fat-wedges (resp. fat-triangles) stabbed by any query point $q$ can be reported efficiently."

### *4.1 Querying fat-wedges*

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 $\gamma$, $0 \leq i \leq t-1$ (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 $\alpha'$ and $\alpha''$ 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 3.3 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.

**Lemma 15** *A set $S$ of $n$ colored fat-wedges in $\mathcal{R}^2$ can be stored in a data structure of size $O(n\log n)$ such that the $i$ distinct colors of the fat-wedges* *that are stabbed by a query point can be reported in* $O(\log^2 n + i)$ *time.*

## *4.2 Querying fat-triangles*

We split each fat-triangle that does not have a vertical side into two such triangles. We then store these triangles in a segment tree $T$ according to their $x$-spans. Let $C(v)$ be the triangles allocated to $v$. Each triangle $t \in C(v)$ has exactly two sides, $s_1(t)$ and $s_2(t)$, that cross *Strip*$(v)$. The wedge $w(t)$ supporting $s_1(t)$ and $s_2(t)$ is a fat-wedge. Call it a *left-wedge* (resp. *right-wedge*) if the common vertex of $s_1(t)$ and $s_2(t)$ is on or to the left (resp. on or to the right) of *Strip*$(v)$.

At $v$, we build two structures $L(v)$ and $R(v)$: $L(v)$ is the fat-wedge stabbing structure of Lemma 15 and is built on the left-wedges determined by $C(v)$. $R(v)$ is symmetric. Given $q$, we search down $T$ with $q$'s $x$-coordinate and query $L(v)$ and $R(v)$ at each node visited.

It is easy to see that for any triangle $t \in C(v)$, where $v$ is a node visited by the search, $q \in t \cap \textit{Strip}(v)$ iff $q \in w(t)$. This shows that the method is correct. Since each triangle is stored in $O(\log n)$ nodes of $T$, from Lemma 15 it follows that the space is $O(n \log^2 n)$. Since $O(\log n)$ nodes $v$ are queried, the query time is $O(\log^3 n + i \log n)$.

We now show how to reduce the query time to $O(\log^2 n + i)$ while increasing the space to $O(n^{1+1/c} \log^2 n)$, where $c > 1$ is a constant. [6] Let $T$ be the segment tree in the previous solution. Assume that $T$ is a complete binary tree and that the height, $h$, of $T$ is a multiple of $c$. (This simplifies the discussion; the extension to the general case is not difficult.) Let $k = h/c$. The idea is to store all the triangles only at nodes of height $jk$, where $0 \leq j \leq c$. Specifically, let $t$ be a triangle which is stored at a node $v$ in the previous solution. If $v$'s height lies between $jk + 1$ and $(j + 1)k - 1$, where $0 \leq j < c$, then instead of storing $t$ at $v$, we store $t$ at all descendants of $v$ of height $jk$. If $v$'s height is $jk$, $0 \leq j \leq c$, then we store $t$ at $v$ itself. At each node $v$ where triangles get stored, we build the structures $L(v)$ and $R(v)$ described above on these triangles.

What is the space used by this structure? A node $v$ whose height lies between $jk+1$ and $(j+1)k-1$ has $O(2^k) = O(n^{1/c})$ descendants of height $jk$. Therefore, each triangle $t$ gets stored at $O(n^{1/c} \log n)$ nodes and it follows that the structure uses $O(n^{1+1/c} \log^2 n)$ space.

To answer a stabbing query, we search down $T$ as before and query the aux-

[6] We thank one of the referees for suggesting this approach.

iliary structures stored at the height $jk$ nodes on the search path, where $0 \leq j \leq c$. The correctness of the query algorithm follows from the fact that the set of triangles stored at a node $v$ of height $jk$ is exactly the set of triangles that "belong" to nodes of height $jk$ through $(j+1)k-1$ on the search path to $v$.

We can now conclude:

**Theorem 16** *A set $S$ of $n$ colored fat-triangles in $\mathcal{R}^2$ can be stored in a data structure of size $O(n^{1+1/c}\log^2 n)$ (resp. $O(n\log^2 n)$) such that the $i$ distinct colors of the fat-triangles that are stabbed by a query point can be reported in $O(\log^2 n + i)$ (resp. $O(\log^3 n + i\log n)$) time. Here $c > 1$ is a constant.*

## 5 Generalized fat-wedge and fat-triangle range searching

We consider the following problem: "Preprocess a set $S$ of $n$ colored points in $\mathcal{R}^2$ so that given any query fat-wedge or fat-triangle $q$, the $i$ distinct colors of the points lying inside $q$ can be reported efficiently."

*Querying with a fat-wedge*

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. 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 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. $r_b$). 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. $l_b^+$).

We now discuss correctness. 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 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 1 and so the total space is $O(n \log^2 n)$. The query time at each node visited is $O(\log^2 n + i)$, which implies an overall query time of $O(\log^3 n + i \log n)$.

What if $q$ is not $y$-monotone? In preprocessing, we select $t = \lceil 2\pi/\gamma \rceil$ coordinate systems $\mathcal{C}_i = (x_i y_i)$ as in Section 4.1. Within each $\mathcal{C}_i$ we build an instance of the above data structure 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 17** *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 $q$ can be reported in time $O(\log^3 n + i \log n)$.*

*Querying with a 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 17 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.

**Theorem 18** *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 fat-triangle $q$ can be reported in time $O(\log^4 n + i \log^2 n)$.*

## 6 Conclusions and further work

We have presented efficient solutions to several generalized intersection searching problems involving objects that are not necessarily iso-oriented. Our methods have included sparse representations, persistence, and filtering search.

Besides improving upon our bounds, two other problems are of particular interest, namely: (i) 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, for the generalized simplex range searching problem in $d \geq 2$ dimensions, and for triangle range searching, triangle stabbing, and segment intersection searching in the plane; and (ii) obtaining dynamic data structures for the generalized problems considered here.

**Acknowledgement**

The authors thank the three referees for many helpful suggestions that strengthened the paper and for pointing out errors in an earlier version.

## References

[1] P.K. Agarwal and M. van Kreveld. Connected component and simple polygon intersection searching. In *Proceedings of the 3rd Workshop on Algorithms and Data Structures*, pages 36–47, August 1993.

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

[3] A. Aggarwal, M. Hansen, and T. Leighton. Solving query-retrieval problems by compacting Voronoi diagrams. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 331–340, 1990.

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

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

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

[7] 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.

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

[9] 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.

[10] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer–Verlag, New York, 1987.

[11] P. Gupta, R. Janardan, and M. Smid. Efficient algorithms for generalized intersection searching on non-iso-oriented objects. Technical Report TR–93–73, Dept. of Computer Science, University of Minnesota, 1993.

[12] P. Gupta, R. Janardan, and M. Smid. Further results on generalized intersection searching problems: counting, reporting, and dynamization. *Journal of Algorithms*, 1995. To appear. A preliminary version is in *Proceedings of the 3rd Workshop on Algorithms and Data Structures*, Montréal, Canada, August 1993, pages 361–372.

[13] P. Gupta, R. Janardan, and M. Smid. On intersection searching problems involving curved objects. In *Proceedings of the 4th Scandinavian Workshop on Algorithm Theory*, Aarhus, Denmark, LNCS Vol. 824, pp. 183–194.

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

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

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

[17] J. Matoušek. Range searching with efficient hierarchical cuttings. *Discrete & Computational Geometry*, 10:157–182, 1992.

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

[19] M.H. Overmars and C.K. Yap. New upper bounds in Klee's measure problem. *SIAM Journal on Computing*, 20:1034–1045, 1991.

[20] 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, New York, 1988.

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

[22] E. Welzl. More on $k$-sets of finite sets in the plane. *Discrete & Computational Geometry*, 1:95–100, 1986.