International Journal of Computational Geometry & Applications
© World Scientific Publishing Company

EFFICIENT NON-INTERSECTION QUERIES ON AGGREGATED GEOMETRIC DATA*

PROSENJIT GUPTA
International Institute of Information Technology
Gachibowli, Hyderabad 500 032, India.
pgupta@iiit.ac.in

Ravi Janardan
Department of Computer Science & Engineering, University of Minnesota
Minneapolis, MN 55455, U.S.A.
janardan@cs.umn.edu

Michiel Smid
School of Computer Science, Carleton University
Ottawa, Canada K1S 5B6.
michiel@scs.carleton.ca

Received (received date)
Revised (revised date)
Communicated by Joseph S.B. Mitchell

Geometric intersection searching problems are a well-studied class of query-retrieval problems with many applications. The goal here is to preprocess a set of geometric objects so that the ones that are intersected by a query object can be reported efficiently. Often, a more general version of the problem arises, where the data comes aggregated in disjoint groups and of interest are the groups, not the individual objects, that are intersected by the query object. One approach to a generalized problem is to ignore the grouping, solve the corresponding classical problem, and then infer the groups from the reported answer. However, this is not efficient, since the number of objects intersected can be much larger than the number of groups (i.e., the output size). The problem of designing efficient, output-sensitive query algorithms for generalized intersection searching has received much attention in recent years, and such solutions have been developed for several problems.

This paper considers a new class of generalized query-retrieval problems. Specifically, given aggregated geometric data the goal is to report the distinct groups such that no objects from those groups are intersected by the query. Of interest in these generalized non-intersection searching problems are solutions where the query time is sensitive to the output size, i.e., the number of groups reported. Unfortunately, the obvious approaches of (i) solving the corresponding generalized intersection searching problem and report-

*A preliminary version of this paper appeared in the Proceedings of the *Eleventh Annual International Computing and Combinatorics Conference (COCOON)*, Kunming, China, Aug. 2005, Lecture Notes in Computer Science, Vol. 3595, Springer–Verlag, pp. 544–553.

ing the complement, or (ii) solving a generalized intersection searching problem with the complement of the query are either inefficient or incorrect. This paper provides efficient, output-sensitive solutions to several generalized non-intersection searching problems, using techniques such as geometric duality, sparsification, persistence, filtering search, and pruning.

Keywords: Geometric searching; generalized non-intersection problems; output-sensitive querying.

1. Introduction

1.1. *Motivation*

Consider the following scenario. The U.S. mutual fund universe consists of about 8,300 mutual funds aggregated into roughly 560 fund families, circa 2003 (Ref. 25).[a] Confronted with this bewildering array of choices, an investor might wish to consolidate his/her holdings by identifying a small number of families whose funds meet desired criteria (e.g., rate of return, volatility, and risk level—each specified as a range of values). This can be accomplished by representing each fund as a point in three dimensions, querying the point-set with a box defined as the Cartesian product of the three ranges, and sifting through the response to identify the different families that the retrieved funds correspond to. In computational geometry terms, this is merely the *standard* orthogonal range searching problem in $\mathbb{R}^3$ (Ref. 8). However, this approach is not efficient for the problem at hand since the query time depends on the number of funds satisfying the query range rather than the number of distinct fund families, which can be much smaller. (For instance, in December 2004, a query using Yahoo's mutual fund screener yielded 935 funds in just 197 families that had a 1-year return of at least 10% and market capitalization between one and five billion dollars.)

Ideally, one would like a query time that depends on the output size, which in this case is the number of fund families. This can be accomplished by assigning each point a "color" that depends on its fund family membership and requesting the distinct colors of the points that are in the query range. This is a *generalized* orthogonal range search problem ("generalized" because it includes the standard problem above as a special case, when each color class has cardinality 1). Several types of such generalized problems have been considered in the literature and efficient, output-sensitive solutions have been developed for them.[3,4,5,6,7,15,16,17,18,20,22,27,28,30]

Returning to our example, suppose instead that our investor is interested in identifying those fund families none of whose funds meet a specified set of criteria. For instance, a conservative investor might want to avoid families whose funds have underperformed with high risk and high volatility. In the above generalized setting, the goal would be to report the distinct colors such that no points of those colors are in the query range that specifies the undesirable levels of performance, risk, and

[a]Worldwide, across 40 major countries, there are about 53,000 funds in an indeterminate number of families.[25]

volatility.

At first sight, it would appear that this problem can be solved quite easily using the generalized approach, by either (i) reporting the complement of the set of distinct colors found in the range, or (ii) querying with the complement of the query range. Unfortunately, this is not the case. Approach (i) is not output-sensitive since the query time would depend on the number of distinct colors in the range, which can be much greater than the number of colors that avoid the range (the output size). Approach (ii) may not even yield the correct answer: a color found in the complement of the query range could also occur within the range! Another problem with this approach is that the complement of the query may not have a compact representation and may need to be split into "simpler" queries (e.g., the complement of a query box would need to be represented as the union of several semi-infinite boxes). Reconciling the responses to these simpler queries and filtering out duplicates could be inefficient. Thus, it appears that a different approach is needed.

As another example, consider the problem of VLSI layout design and verification.[29] Here millions of circuit elements, such as wires, can be aggregated into a relatively small number of circuits (i.e., connected components), and the layout designer is often interested in knowing how the electrical connectivity of the layout is affected when a new circuit element is added. By assigning each circuit element a color that depends on the circuit it belongs to, the problem of determining which circuits get connected by the addition of a new circuit element can be formulated as a generalized intersection problem. On the other hand, deciding which circuits are unaffected by the addition of the new element is a generalized non-intersection query. In the former case, the colors returned by the query correspond to the circuits that get connected; in the latter case, they correspond to the circuits that remain unaffected.

1.2. *Contributions*

Formally, the problem we consider is the following:

- Preprocess a set, S, of n colored geometric objects into a data structure, so that for any query object, q, the distinct colors of the objects in S for which q intersects no objects of those colors can be reported efficiently.

For brevity, we will refer to such a problem as *a generalized non-intersection problem on S with q*. We measure the efficiency of a solution to such a problem by the size of the data structure and its query time. Typically, but not exclusively, we seek solutions with linear or close-to-linear storage (e.g., $O(n \cdot \text{polylog}(n))$) and low output-sensitive query times of the form $O(f(n) + I \cdot g(n))$, where f and g are "small" functions (e.g., polylogarithmic or sublinear in n), and I is the output size (i.e., the number of distinct colors reported). In general, we make no assumptions about the number of different colors in S; it can range from a small constant all the way to n. We also make no assumptions about the number of objects of a given color in S.

As mentioned earlier, for a generalized non-intersection problem, neither solving the corresponding generalized intersection problem and reporting the complement nor solving one or more generalized problems with the complement of the query is a satisfactory, or necessarily correct, solution. In this paper, we develop a different set of techniques, based on methods such as geometric duality, sparsification, persistence, filtering search, and pruning, to solve a variety of generalized non-intersection problems. Table 1 summarizes our results.[b] To the best of our knowledge, there has, surprisingly, been no systematic study of such generalized non-intersection problems prior to this paper. (We are aware of one related result[27], which we mention at the end of Section 3.1.)

Table 1. Summary of results for generalized non-intersection problems on S with q. All bounds are "big-Oh" and worst case. Rectangles are axes-parallel, t (in the result corresponding to Theorem 10) is the number of colors in S, I is the output size, $d > 0$ is an integer constant, $\varepsilon > 0$ is an arbitrarily small constant, u is a parameter in the range $\log n \leq u \leq n$, and m is a parameter in the range $n \leq m \leq n^2$.

Ambient space	Colored objects in S	Query q	Space	Query time	Theorem #
$\mathbb{R}^1$	points	interval	n	$\log n + I$	2
$\mathbb{R}^1$	intervals	interval	$n \log^2 n$	$\log n + I \log^2 n$	1
$\mathbb{R}^2$	points	quadrant	n	$\log n + I$	3
$\mathbb{R}^2$	points	grounded rect.	$n \log n$	$\log n + I$	5
$\mathbb{R}^2$	points	rectangle	$(n^2/u) \log n$	$u + I$	6
$\mathbb{R}^2$	horizontal line segs.	vertical line segment	$n \log n$	$\log n + I$	7
$\mathbb{R}^1$	intervals	point	n	$\log n + I$	8
$\mathbb{R}^2$	rectangles	point	$n \log^2 n$	$\log n + I \log^2 n$	1
$\mathbb{R}^d$ ($d \geq 2$)	hyper-rects.	point	$n^d \log^{d-2} n$	$\log^{d-1} n + I$	8
$\mathbb{R}^2$	points	halfplane	$n \log n$	$\log^2 n + I$	9
			tn	$\log n + I$	10
$\mathbb{R}^3$	points	halfspace	$n \log^2 n$	$n^{1/2+\varepsilon} + I$	9
			$n^{2+\varepsilon}$	$\log^2 n + I$	
$\mathbb{R}^2$	convex polygon	convex polygon (constant size)	$n^{1+\varepsilon}$	$\log^2 n + I$	11
		convex polygon	$n^{2+\varepsilon}$	$\lvert q\rvert \log n + I$	12
			m	$\lvert q\rvert n^{1+\varepsilon}/\sqrt{m} + I$	

[b]We also show how some of our results can be used to solve the following single-shot (i.e., non-query) non-intersection problem efficiently: Given a set of convex polygons in $\mathbb{R}^2$, report all non-intersecting pairs output-sensitively, where the output size is the number of non-intersecting pairs. This result is described in Section 7.3 and may be of independent interest. (It is not reported in Table 1 in order to maintain the unity of the query problems listed there, which is the focus of this paper.)

The rest of the paper is organized as follows. In Section 2, we present a general technique for solving non-intersection problems by incorporating solutions to certain generalized intersection problems within a prune-and-search framework. In Sections 3–7, we consider generalized non-intersection problems for various types of colored objects and queries, including orthogonal ranges, line segments, points, halfplanes, and convex polygons. We conclude in Section 8.

2. A General Approach for Non-intersection Queries

We describe a method that can be used for any generalized non-intersection query on a set S of colored objects with a query object q. All that is required is a data structure for the corresponding generalized intersection problem on S and q which can answer counting queries. (Since the method is quite general, it does not always yield the best possible bounds. We will see later that some of the bounds can be improved by taking advantage of problem-specific features.)

We store the distinct colors in S at the leaves of a balanced binary tree, T, in no particular order. For any node v of T, let $C(v)$ be the set of colors stored in the leaves of v's subtree and let $S(v) \subseteq S$ be the set of objects whose colors appear in $C(v)$. We store the following information at v:

(1) A count, $\mathit{count}(v) = |C(v)|$, of the number of distinct colors in v's subtree.
(2) A data structure, $G(v)$, to answer generalized counting queries on $S(v)$ with q. That is, $G(v)$ returns the number of distinct colors among the objects in $S(v)$ that are intersected by q.

To answer a non-intersection query on S with q, we do a depth-first search of T. Let v be the current node and suppose that it is a non-leaf. If the query on $G(v)$ with q returns a count equal to $\mathit{count}(v)$, then we abandon the search below v. (This is because q intersects at least one object of $S(v)$ for each color in $C(v)$, so the response to the non-intersection query on $S(v)$ with q is the empty set.) If the count returned by the query on $G(v)$ with q is less than $\mathit{count}(v)$, then we proceed to search recursively below v (since there must be at least one color in $C(v)$ for which q intersects no object of that color in $S(v)$). If v is a leaf, then we output the color stored at v if and only if the count returned by the query on $G(v)$ with q is zero.

This approach is similar to one given by us in Refs. 19 and 20 for generalized intersection queries, with one key difference. There we pruned the search below a node v if and only if q did not intersect any object in $S(v)$, which we were able to decide efficiently by using for $G(v)$ a structure that detects if q intersects any object in $S(v)$—a standard problem. Here, we need to prune the search below v if and only if q intersects at least one object of each color that occurs among the objects in $S(v)$. This is most easily determined by using a generalized counting structure for $G(v)$, as above; a standard counting would not reveal this information and a standard reporting query would be too expensive.

Theorem 1. *Let $M(m)$ and $f(m)$ be, respectively, the space and query time complexity of the data structure $G(v)$, where $m = |S(v)|$. Assume that $M(m)/m$ and $f(m)$ are non-decreasing functions for non-negative values of m. Then a set, S, of n colored objects can be preprocessed into a data structure of size $O(M(n)\log n)$ such that a generalized non-intersection query on S with q can be answered in time $O(f(n) + I \cdot f(n)\log n)$, where I is the number of colors reported.*

Proof. The proof is along the lines of the one in Refs. 19 and 20 for the generalized intersection problem, with some differences. We give it here for completeness.

We first prove the correctness of the query algorithm by showing that a color c is reported if and only if there is no c-colored object in S intersecting q.

Suppose that c is reported. This implies that a leaf v is reached in the search such that v stores c and the query on $G(v)$ with q returns zero. Thus, no object in $S(v)$ intersects q. Since v is a leaf, the objects in $S(v)$ are precisely the c-colored objects of S and the claim follows.

For the converse, suppose that q intersects no c-colored object. Let v be the leaf storing c. Now, c is in $C(v')$, for each node v' on the root-to-v path in T, so the query on $G(v')$ with q will return a count that is less than $\mathit{count}(v')$. Therefore, v will eventually be reached in the search and the color c will be reported.

We now analyse the space and time complexity of the query algorithm. If $v_1, v_2, \ldots, v_r$ are the nodes of T at any level, then the total space used by T at that level is $\sum_{i=1}^{r} M(|S(v_i)|) = \sum_{i=1}^{r} |S(v_i)| \cdot (M(|S(v_i)|)/|S(v_i)|) \leq \sum_{i=1}^{r} |S(v_i)| \cdot (M(n)/n) = M(n)$, since $\sum_{i=1}^{r} |S(v_i)| = n$ and since $|S(v_i)| \leq n$ implies that $M(|S(v_i)|)/|S(v_i)| \leq M(n)/n$. (Recall that $M(m)/m$ is non-decreasing for $m \geq 0$.) Since there are $O(\log n)$ levels, the overall space is $O(M(n)\log n)$.

The query time can be upper-bounded as follows: If $I = 0$, then the query on $G(\text{root})$ with q returns a count equal to $\mathit{count}(\text{root})$ and we abandon the search at the root itself; in this case, the query time is just $O(f(|S(\text{root})|)) = O(f(n))$. Suppose that $I \neq 0$. Call a visited node v *fruitful* if the query on $G(v)$ with q returns a count that is less than $\mathit{count}(v)$; call it *fruitless*, otherwise. Each fruitful node can be charged to some color in its subtree that gets reported eventually. Since the number of times any reported color can be charged is $O(\log n)$ (the height of T) 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 T is a binary tree, it follows that there are only $O(I \log n)$ fruitless nodes. Hence the number of nodes visited by the search is $O(I \log n)$. At each such node, v, we spend $f(|S(v)|)$ time, which is $O(f(n))$ since $|S(v)| \leq n$. (Recall that $f(m)$ is non-decreasing for $m \geq 0$.) Thus the total time spent in doing queries at the visited nodes is $O(I \cdot f(n)\log n)$ and the claimed query time bound follows. □

Applications: The above method can be used to derive efficient solutions for several generalized non-intersection problems, as summarized in Table 2. In each case, $G(v)$ is a structure from Ref. 16.

Table 2. Summary of results for generalized non-intersection problems on S with q using the general method. All bounds are "big-Oh" and worst case. Rectangles are axes-parallel, I is the output size, and $M(m)$ and $f(m)$ are, respectively, the space and query time complexity of the data structure for generalized intersection counting used in the general method.

Ambient space	Colored objects in S	Query q	$M(m)$	$f(m)$	Space	Query time
$\mathbb{R}^1$	intervals	interval	$m \log m$	$\log m$	$n \log^2 n$	$\log n + I \log^2 n$
$\mathbb{R}^1$	intervals	point	m	$\log m$	$n \log n$	$\log n + I \log^2 n$
$\mathbb{R}^2$	points	rectangle	$m^2 \log^2 m$	$\log^2 m$	$n^2 \log^3 n$	$\log^2 n + I \log^3 n$
$\mathbb{R}^2$	rectangles	point	$m \log m$	$\log m$	$n \log^2 n$	$\log n + I \log^2 n$

3. Querying Points with Orthogonal Ranges

We show how to answer generalized non-intersection queries on sets of colored points in $\mathbb{R}^1$ and $\mathbb{R}^2$, with queries such as intervals, quadrants, grounded rectangles, and arbitrary rectangles—all axes-parallel—where the solutions build successively upon preceding ones.

3.1. *Querying with intervals in $\mathbb{R}^1$*

Given a set, S, of n colored points on the real line, $\mathbb{R}^1$, we wish to answer a generalized non-intersection query with an interval $q = [a, b]$. We solve this problem by transforming it into a standard range searching problem in $\mathbb{R}^2$.

For each color, c, let S_c be the points of color c. We sort the points of S_c in nondecreasing order as $p_1, p_2, \ldots, p_r$, where $r = |S_c|$. We transform this sequence of points into a sequence of intervals $(-\infty, p_1], [p_1, p_2], \ldots [p_r, +\infty)$, which we denote by L_c.

Intuitively, if q misses all points of color c, then it must fall into exactly one of the gaps defined by consecutive points of S_c (i.e., intervals of L_c). On the other hand, if q contains points of color c, then it must straddle two or more gaps. This is formalized below.

Lemma 1. *Query interval $q = [a, b]$ does not contain any points of color c if and only if q is contained properly in exactly one interval of L_c.*

Proof. Suppose that q contains no point of S_c. Assume, for a contradiction, that q is not contained properly in any single interval of L_c. Then, since the intervals in L_c are non-overlapping (except at endpoints) and cover $\mathbb{R}^1$, q must overlap two or more intervals of L_c and, hence, must contain their shared endpoints—a contradiction.

Conversely, suppose that q is contained properly in a single interval of L_c. This interval cannot overlap any other interval of L_c, since the interiors of the intervals in L_c are disjoint. Thus, q cannot contain the endpoint of any other interval of L_c, hence q contains no point of S_c. □

Lemma 1 implies that the original generalized non-intersection problem can be re-phrased as: "Given the collection of intervals in L_c, for all colors c, report the intervals that contain q." Observe that this is a *standard* intersection problem, because, for each c, at most one interval of L_c can contain q.

To solve the above problem, we map each interval $[x, y]$ to a point (x, y) in $\mathbb{R}^2$, and associate with it the color of $[x, y]$. Note that $q = [a, b] \subset [x, y]$ if and only if $x < a$ and $y > b$, i.e. if and only if point (x, y) is contained properly in the northwest quadrant of the point (a, b). The points contained in such a query quadrant can be reported efficiently by storing the points (x, y) in a priority search tree. [26] For a set of m points (x, y), the query time and space are $O(\log m + k)$ and $O(m)$, respectively, where k is the number of points reported. Note that k is also the output size, I, of the original, generalized non-intersection problem. Moreover, if the original problem had n colored points on $\mathbb{R}^1$, then the transformed problem has $m = O(n)$ points. Finally, we note that the solution can be made dynamic to accommodate the insertion and deletion of colored points in S in $O(\log n)$ time. For each color c, we maintain the points of S_c in a balanced binary search tree. We use this to update L_c whenever a point is inserted or deleted in S_c, and then update the priority search tree with the corresponding point in $\mathbb{R}^2$, all in time $O(\log n)$. (We note that the static problem on integer inputs has been solved independently in Ref. 27, in time $O(I)$, using a different approach.)

Theorem 2. *A set, S, of n colored points on $\mathbb{R}^1$ can be preprocessed into a data structure of size $O(n)$, so that for any query interval q, a generalized non-intersection query on S with q can be answered in time $O(\log n + I)$, where I is the number of colors reported. Moreover, colored points can be inserted or deleted in S in time $O(\log n)$.*

3.2. *Querying with quadrants in $\mathbb{R}^2$*

Given a set, S, of n colored points in $\mathbb{R}^2$, we wish to answer a generalized non-intersection query for a quadrant defined by a query point $q = (a, b)$. Specifically, we consider the north-east quadrant, $NE(q)$, defined as the set of all points (x, y) such that $x \geq a$ and $y \geq b$. (Note that this is different from the quadrant problem that arose in Section 3.1; that problem was a standard intersection problem.)

As before, let S_c be the set of points of color c. Let M_c be the set of *maximal points* of S_c; that is, $M_c \subseteq S_c$ consists of points each of whose north-east quadrants contains no point of S_c.

M_c can be used to define a set, H_c, of horizontal line segments that form a staircase-like structure which descends rightwards. Specifically, sort M_c by decreasing y-coordinates. For each pair of consecutive points, (x_p, y_p) and (x_r, y_r), in this order, where $y_p > y_r$ (note that $x_p < x_r$), insert the horizontal segment $((x_p, y_r), (x_r, y_r)]$ into H_c. Also include in H_c, a horizontal ray $((-\infty, y_h), (x_h, y_h)]$, where (x_h, y_h) is the highest point of M_c, and a horizontal ray $[(x_\ell, -\infty), (\infty, -\infty))$, where (x_ℓ, y_ℓ) is the lowest point of M_c. Note that all the segments in H_c (including the two rays) are open on the left.

Intuitively, if $NE(q)$ does not contain any point of S_c, hence also of M_c, then it does not intersect any segment from H_c. Thus, q must lie "above" the staircase defined by H_c. We compute the staircases for each color and then shoot a ray downward from q to find the ones that are intersected. The ray intersects a staircase below q exactly once, which allows the generalized problem at hand to be framed as a standard one. We formalize this below. We say that the intersection of $Ray(q)$ with a segment of H_c is *proper* if q is strictly above the segment.

Lemma 2. *Let $Ray(q)$ be the ray that emanates from q and is directed downwards. Then $NE(q)$ contains no point of S_c if and only if $Ray(q)$ intersects properly exactly one segment of H_c.*

Proof. Suppose that $NE(q)$ contains no point of S_c. The vertical projections of the segments of H_c partition the real line. Thus the projection of q along $Ray(q)$ is contained in the projection of exactly one segment, s, of H_c. We show below that q is strictly above s, which proves the "only if" part of the lemma.

If s is the lower horizontal ray of H_c, which is at y-coordinate $-\infty$, then q is clearly strictly above s. Otherwise, let the right endpoint of s be (u, v); note that this is a point of M_c. Since, by assumption, $NE(q)$ does not contain (u, v), we have $u < a$ or $v < b$. However, since the projection of q along $Ray(q)$ is contained in the projection of s and since (u, v) is the right endpoint of s, we have $a \leq u$. Thus, $v < b$, so q is strictly above s.

Conversely, let $Ray(q)$ intersect properly exactly one segment, g, of H_c. Let g_x be the x-coordinate of the left endpoint defining g and let g_y be the y-coordinate of g. Note that we have $a > g_x$ (since g is open on the left) and $b > g_y$ (since $Ray(q)$ intersects g properly). Note also that for any point $(x, y) \in S_c$, we have that $x \leq g_x$ or $y \leq g_y$. This follows from the facts that (i) every point in S_c is either in M_c or in the south-west quadrant of some point of M_c and (ii) as we walk along the points of M_c in decreasing order of their y-coordinates, their x-coordinates increase (i.e., the staircase descends rightwards). It follows that $x < a$ or $y < b$, which implies that (x, y) is not in $NE(q)$. □

Our goal now is to solve the standard problem of reporting the segments of H_c, for all colors c, that are intersected by $Ray(q)$. Given S, we compute and store the segments of H_c, for all colors c, in a hive-graph[8] and query this with $Ray(q)$, for any query point q.

The total size of the sets H_c, for all colors c, is $O(n)$ and they can all be computed in time $O(n \log n)$. The time to build the hive-graph is $O(n \log n)$. (Since the query is a ray, a point location structure for the hive-graph is not needed; see Ref. 8.) The query time is $O(\log n + k)$, where k is the number of intersected segments. This is $O(\log n + I)$, since, by Lemma 2, it follows that $k = I$.

Theorem 3. *A set, S, of n colored points in the plane can be preprocessed in time $O(n \log n)$ into a data structure of size $O(n)$, so that for any query point q,* *a generalized non-intersection query on S with the north-east quadrant of q can be answered in time $O(\log n + I)$, where I is the number of colors reported.*

3.2.1. *A semi-dynamic solution*

The preceding ideas can be extended to also allow efficient insertion of colored points, via the following data structures.

(1) A structure $\mathcal{D}$ storing the segments in the sets H_c, for all colors c. $\mathcal{D}$ allows efficient reporting of those stored segments that are intersected by a downward-directed query ray and, moreover, allows segments to be inserted and deleted. $\mathcal{D}$ is implemented as discussed in Ref. 9; for a set of n segments, $\mathcal{D}$ uses $O(n)$ space, supports updates in $O(\log n)$ time, and queries in $O(\log^2 n + k)$ time, where k is the number of segments reported.
(2) For each color, c, a balanced search tree, T_c, which stores the segments of H_c by increasing y-coordinates. As shown in Ref. 16, with this collection of trees it is possible to dynamically maintain H_c when a new c-colored point p is inserted, using $O((m+1)\log n)$ time (or even $O(\log n + m)$ time), where m is the number of points in M_c that cease to be maximal when p is inserted. Additionally, the set of all colors is stored in a balanced binary tree T.

To answer a query, we use $\mathcal{D}$. To insert a new point, p, of some color, c, we use T_c to identify the points of M_c that cease to be maximal, delete the corresponding segments of H_c from T_c and $\mathcal{D}$, and insert a segment of H_c associated with p into T_c and $\mathcal{D}$. If necessary, we also update the set of colors using T. The correctness and time complexity of the query algorithm follows from the previous discussion and the query time bound for $\mathcal{D}$, while that of the update algorithm follows from the discussion in Ref. 16.

Theorem 4. *Let S be a set of colored points in $\mathbb{R}^2$ and let n be its current size. S can be preprocessed into a data structure of size $O(n)$, so that for any query point q, a generalized non-intersection query on S with the north-east quadrant of q can be answered in time $O(\log^2 n + I)$, where I is the number of colors reported. Moreover, if S is initially empty, then the amortized cost of inserting n colored points into S is $O(\log n)$ per insertion.*

3.3. *Querying with grounded rectangles in $\mathbb{R}^2$*

The data structure underlying Theorem 2 can be coupled with the notion of persistence[13] to answer generalized non-intersection queries on a set, S, of n colored points in $\mathbb{R}^2$ with a grounded query rectangle $q = [a, b] \times [f, \infty)$. We note that the data structure of Theorem 2 fits into the framework of Ref. 13, as it is a linked structure of bounded in-degree.

We create a linked list, L, which contains the c-colored point of maximum y-coordinate, for each color c (ties broken arbitrarily); L is sorted by non-decreasing y-

coordinates. Next, we sort the points of S by non-increasing y-coordinates and insert them in this order into a partially persistent version of the structure of Theorem 2, using the x-coordinate as the key. To answer a query $q = [a, b] \times [f, \infty)$, we proceed in two steps. First, we access the version of the persistent data structure corresponding to the smallest y-coordinate greater than or equal to f and query it with the interval $[a, b]$. Second, we traverse L in the order of nondecreasing y-coordinates and report the colors of all points in it with y-coordinate less than f.

The first step above considers only the subset S' of S consisting of points with y-coordinate greater than or equal to f. Since q is infinite in the positive y-direction, a generalized non-intersection query on S with q is equivalent to a generalized non-intersection query on the x-coordinates of the points in S' with the interval $[a, b]$. However, note that it is possible that there are colors such that all points of those colors in S have y-coordinate less than f. These colors need to be reported but they will not be found in the first step of the query. However, the second step of the query, using L, will report each such color exactly once.

To build the partially persistent structure, we do n insertions, each resulting in $O(\log n)$ memory modifications. Thus the partially persistent structure takes $O(n \log n)$ space. The query time is the same as in Theorem 2. The traversal of L takes time proportional to the number of points reported from L, all of which contribute to the output size I.

Theorem 5. *A set, S, of n colored points in $\mathbb{R}^2$ can be preprocessed into a data structure of size $O(n \log n)$, so that for any grounded query rectangle $q = [a, b] \times [f, \infty)$, a generalized non-intersection query on S with q can be answered in time $O(\log n + I)$ time, where I is the number of colors reported.*

3.4. *Querying with orthogonal rectangles in $\mathbb{R}^2$*

The data structure of Theorem 5 can be incorporated within the filtering search paradigm[8] to answer efficiently generalized non-intersection queries on a set S of n colored points in $\mathbb{R}^2$ with a general (axes-parallel) query rectangle $q = [a, b] \times [f, g]$. The solution uses $O((n^2/u) \log n)$ space and has a query time of $O(u + I)$, where u is a parameter, $\log n \leq u \leq n$. (This yields, for example, a solution with $O(n^2)$ (resp. $O(n^{3/2} \log n)$) space and $O(\log n + I)$ (resp. $O(n^{1/2} + I)$) query time.) The idea is to partition $\mathbb{R}^2$ into regions within which q behaves like a grounded rectangle. However, in doing so, q may retrieve some colors that are not in the true output and may also fail to report some colors that should be in the true output. Fortunately, both of these situations can be rectified efficiently to produce the correct output.

Let $p_1, p_2, \ldots, p_n$ be the points of S, sorted in increasing order of their y-coordinates. (For simplicity, we assume that the y-coordinates in S are distinct.) For each k, $1 \leq k \leq n/u$, let $S_k = \{p_1, p_2, \ldots, p_{ku}\}$, and let y_k be the y-coordinate of the point p_{ku}.

Our data structure consists of the following components:

- An instance, D_k, of the data structure of Theorem 5 storing S_k, $1 \leq k \leq n/u$.
- A linked list L of reals sorted in decreasing order. There is one entry in L for each color in S and it is the minimum of the y-coordinates of points of that color.
- A real array C indexed by color. Each entry in C is the minimum of the y-coordinates of points of that color.
- Integer arrays B and E indexed by color. Both are initialized to zero.
- Linked lists A and F to store colors. Both are initialized to empty.

A query on S, with $q = [a, b] \times [f, g]$, is answered as follows:

(1) Using binary search, we find the index k such that $y_k \leq g < y_{k+1}$.

Thus, q's upper edge is on or above the horizontal line $y = y_k$. W.r.t. the subset of S on or below this line (i.e., S_k), q functions like the grounded rectangle $q' = [a, b] \times [f, \infty)$. Let $\hat{q}$ be the portion of q lying above the line $y = y_k$.

(2) We query D_k with q' and store the reported colors (i.e., the ones found not to be in q') in the list A. For each color c in A, we set $B[c] = 1$, and store with $B[c]$ a pointer to its occurrence in A.

At this stage, A may contain colors that are actually in $\hat{q}$, and hence should not be reported. We say that these colors are *over-retrieved.* A may also be missing certain colors that are not in q because these colors do not occur in the set S_k stored in D_k. These colors should be reported; we call them *under-retrieved* colors. The over-retrieved colors are filtered out and the under-retrieved colors are discovered in the next few steps.

(3) We first identify the distinct colors among the points p_j, where $ku + 1 \leq j \leq (k + 1)u$. (These are the points in the set $S_{k+1} \setminus S_k$.) For each point p_j, if $E[c]$ is 0, then we set it to 1 and insert c into list F, where c is the color of p_j. After this sub-step, F contains the desired distinct colors.

Next, we use F to reset E to zero. We then take each p_j and if it is in $\hat{q}$ then we set $E[c]$ to 1. After this sub-step, we know both the distinct colors in the set $S_{k+1} \setminus S_k$ and the membership in $\hat{q}$ of each color in this set. (Note that array E is used in two roles here—first to populate F and then to record membership in $\hat{q}$.)

(4) We now proceed to identify the over-retrieved colors. For each color c in list F, if $E[c] = 1$ and $B[c] = 1$, then color c is present in $\hat{q}$ (hence in q) and also in the output list A. Therefore, c is over-retrieved, so we remove it from A (by following the pointer stored with $B[c]$).

(5) Next, we identify some of the under-retrieved colors, specifically the colors c that are not in S_k but are in S_{k+1}. (This step below could have been combined with the previous one, but we have chosen to separate them for clarity.)

For each color c in list F, if $E[c] = 0$ then c is not in $\hat{q}$. If c is not in S_k, then it is under-retrieved and needs to be added to the output list A. Observe that c is not in S_k iff $C[c] > y_k$. Therefore, if $E[c] = 0$ and $C[c] > y_k$, then we add

c to A. (Note that we cannot conclude that c is not in S_k by merely checking whether $B[c] = 0$; the latter condition can hold if c is present in q', in which case c should not be part of the output.)

(6) Now, we identify the remaining under-retrieved colors, i.e., the colors c that are not in S_k but are in S_i, $i > k+1$.

We traverse the list L, by decreasing y-coordinates and for each color c whose y-coordinate is larger than y_{k+1}, we add c to A. Observe that by definition of L, color c satisfies the condition that it is not in S_k but is in S_i, $i > k+1$.

(7) Finally, we output the colors in A as the answer to the query q. Also, to initialize the structure for the next query, we set the non-zero entries in B and E to zero, using the lists A and F, respectively, and then reset A and F to empty.

Theorem 6. *A set, S, of n colored points in $\mathbb{R}^2$ can be preprocessed into a data structure of size $O((n^2/u)\log n)$, so that for any query rectangle $q = [a,b] \times [f,g]$, a non-intersection query on S with q can be answered in time $O(u+I)$, where I is the number of colors reported and u is a parameter, $\log n \le u \le n$.*

Proof. The correctness of the query algorithm follows from the discussion above.

For the space bound, note that by Theorem 5, the size of each data structure D_k is $O(n_k \log n_k)$, where $n_k = |S_k| = ku$. Hence, the total space used is $O(\sum_{k=1}^{n/u} ku \log n_k) = O(u \log n \sum_{k=1}^{n/u} k) = O((n^2/u)\log n)$. The rest of the data structure clearly has size $O(n)$.

For the query time, note that colors that belong to the final output are generate in steps 2, 5, and 6. Accordingly, let I_2, I_5, and I_6 be the sizes of the (portions of the) final outputs generated in these steps (i.e., the overall output size $I = I_2 + I_5 + I_6$). Let I' be the number of over-retrieved colors generated in step 2 (and eliminated in step 4). Note that $I' \le u$ since the number of over-retrieved colors cannot be larger than the size of $S_{k+1} \setminus S_k$.

Step 1 takes $O(\log n)$ time, step 2 takes $O((I_2+I')+\log n)$ time (by Theorem 5), step 3 takes $O(u)$ time, step 4 takes $O(u+I')$ time, step 5 takes $O(u+I_5)$ time, step 6 takes $O(1+I_6)$ time, and step 7 takes $O(I_2+I_5+I_6+u)$ time. So the total query time is $O((I_2+I_5+I_6)+I'+u+\log n) = O(u+I)$, since $I' \le u$ and $\log n \le u$. □

4. Queries Involving Orthogonal Line Segments

We consider answering generalized non-intersection queries on a set of colored horizontal line segments in $\mathbb{R}^2$ with a vertical query segment q, whose endpoints are (a,f) and (a,g). Our solution is based on two persistent data structures—one a persistent version of the structure of Theorem 2 and the other a persistent red-black tree.

Suppose that we know beforehand the x-coordinate, a, of q, and, for this position, we know the y-coordinates of all the segments that intersect the supporting line of q. Call the distinct colors of these segments the *active colors*; the remaining colors are the *inactive colors*. We can solve our generalized non-intersection problem, w.r.t.

the active colors, by using an instance of the data structure in Theorem 2. Since we do not know q ahead of time, we make the structure of Theorem 2 persistent, so that it can handle any query q. Note that this only reports the active colors that are not intersected by q. We also need to report all the inactive colors. For this, we track the set of inactive colors over all possible queries q, again using persistence.

We sort the endpoints of the segments in S by nondecreasing x-coordinates (favoring right endpoints over left endpoints, in the case of ties), and sweep over them with a vertical line L. Let D be an instance of the data structure of Theorem 2, let T be a red-black tree, and let B be an array indexed by color. At any time in the sweep:

- D stores a set of colored points that are the y-coordinates of the segments that are intersected currently by L,
- T stores the inactive colors, and
- B stores, for each color, the number of segments of that color that are intersected by L.

Note that both D and T are linked structures of bounded in-degree, so the results of Ref. 13 apply.

Initially, L is to the left of all the segments, so no segments are intersected by it and all the colors are inactive. Thus, D is empty, T contains all the colors, and each entry of B is zero. We build the persistent versions of D and T as follows.

In a general step, suppose that L reaches an endpoint e of a horizontal segment h, of color c.

- If e is the left endpoint of h, then we insert persistently the y-coordinate of h in D, with color c. We increment $B[c]$ and, if $B[c] = 1$ now, then we delete c persistently from T. (If $B[c] = 0$ before the increment, then c was inactive so far; it now becomes active, so it must be deleted from T. If $B[c] > 0$ before the increment, then c was already active and would have been deleted from T earlier in the sweep.)
- If e is the right endpoint of h, then we delete persistently the y-coordinate of h from D. We decrement $B[c]$ and, if $B[c] = 0$ now, then we insert c persistently into T (since c has now become inactive).

Denote the persistent versions of D and T as $\mathcal{D}$ and $\mathcal{T}$, respectively. Given q, with endpoints (a, f) and (a, g), we locate in $\mathcal{D}$ the instance of D corresponding to the largest x-coordinate that is at most a and answer the query underlying Theorem 2 using the interval $[f, g]$. In addition, we also output the colors in the corresponding instance, T, of $\mathcal{T}$.

The correctness of the query algorithm follows from the discussion above. The time to build $\mathcal{D}$, $\mathcal{T}$, and B is $O(n \log n)$, since updates on D and T take $O(\log n)$ time apiece and updates on B take $O(1)$ time, and there are $O(n)$ updates. The number of memory modifications on D and T is $O(\log n)$ per update (since their update times are $O(\log n)$), so the additional space needed to make them persistent (i.e., to build $\mathcal{D}$ and $\mathcal{T}$) is $O(n \log n)$. (Note that B is needed only to maintain T during the sweep, and is not used in answering a query, so it need not be made persistent.)

Theorem 7. *A set, S, of n colored, horizontal line segments in $\mathbb{R}^2$, can be preprocessed in time and space $O(n \log n)$, so that for any vertical query segment, q, a generalized non-intersection query on S with q can be answered in time $O(\log n + I)$, where I is the number of colors reported.*

5. Querying Ranges with Points in $\mathbb{R}^d$

We consider answering generalized non-intersection queries on a set of n colored, orthogonal ranges (i.e., axes-parallel hyper-rectangles) in $\mathbb{R}^d$ with a query point q, where $d > 0$ is a constant.

For each color c, let R_c be the set of colored ranges of color c. We compute the complement of the union of the ranges in R_c; call this complement U_c. Clearly, q does not intersect any range in R_c if and only if q is in U_c.

We partition U_c into a collection, T_c, of axes-parallel ranges, and associate the color c with each. Note that q does not intersect any range in R_c if and only if it intersects exactly one range in T_c. Thus, we have reduced the original, generalized non-intersection problem to a standard intersection problem. Specifically, we wish to preprocess the collection of the sets T_c, for all colors c, to report the ones that are intersected by q. For $d > 1$, this problem can be solved in $O(m \log^{d-2} m)$ space and $O(\log^{d-1} m + k)$ query time for a set of m ranges in $\mathbb{R}^d$ (see Ref. 8), where k is the output size; clearly, $k = I$—the output size for the generalized non-intersection problem. The size of U_c is $O(n_c^d)$, where $n_c = |R_c|$. Thus T_c has $O(n_c^d)$ ranges and the total number of ranges, taken over all colors c, is $m = O(\sum_c n_c^d) = O(n^d)$.

For $d = 1$, U_c, hence T_c, consists of disjoint intervals and the standard problem to be solved is to preprocess these intervals, for all colors, in order to report the ones that contain q. This can be solved in $O(n)$ space and $O(\log n + I)$ query time using an interval tree.[12]

Theorem 8. *A set, S, of n colored, axes-parallel ranges in $\mathbb{R}^d$ ($d > 1$) can be preprocessed into a data structure of size $O(n^d \log^{d-2} n)$, so that for any query point q a generalized non-intersection query on S with q can be answered in time $O(\log^{d-1} n + I)$, where I is the number of colors reported. For $d = 1$, the bounds are $O(n)$ space and $O(\log n + I)$ query time.*

6. Querying Points with Halfspaces in $\mathbb{R}^d$, $d = 2, 3$

We consider how to answer generalized non-intersection queries on a set, S, of n colored points in $\mathbb{R}^d$, $d = 2, 3$, with a query halfspace bounded by a hyperplane q. Specifically, if x_i, $1 \le i \le d$, are the coordinate axes, then the query is the open halfspace, q^-, consisting of all points in $\mathbb{R}^d$ that lie below q in direction x_d. (For now, we assume that q is not vertical w.r.t. direction x_d; at the end of this section, we show how to remove this assumption.)

Our approach is based on transforming the problem to a standard intersection problem in a dual space. Let $\mathcal{F}$ denote the well-known point-hyperplane duality transform[14]: If $p = (p_1, \ldots, p_d)$ is a point in $\mathbb{R}^d$, then $\mathcal{F}(p)$ is the hyperplane $x_d = 2p_1x_1 + \cdots + 2p_{d-1}x_{d-1} - p_d$. If $H : x_d = a_1x_1 + \cdots + a_{d-1}x_{d-1} + a_d$ is a non-vertical hyperplane in $\mathbb{R}^d$, then $\mathcal{F}(H)$ is the point $(a_1/2, \ldots, a_{d-1}/2, -a_d)$. It is easily verified that p is above (resp. on, below) H, in the x_d-direction, if and only if $\mathcal{F}(p)$ 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 the colored points in S to a set, S', of colored hyperplanes and map q to the point $q' = \mathcal{F}(q)$, all in $\mathbb{R}^d$. Note that the halfspace q^+ maps to a ray, $Ray(q)$, which emanates from q' and is directed downwards. The following lemma allows us to re-state the generalized non-intersection problem as a standard intersection problem.

Lemma 3. *Let E_c be the upper-envelope of the hyperplanes of color c, i.e., the locus of points of maximum x_d-coordinate on these hyperplanes. There are no points of color c in q^- if and only if $Ray(q)$ intersects E_c. Moreover, if $Ray(q)$ intersects E_c, then it does so exactly once.*

Proof. Suppose that there are no points of color c in q^-. Thus, all such points are in q^+. By duality, the hyperplanes of color c in S' all lie on or below q'. Thus, $Ray(q)$ intersects all these hyperplanes, hence also E_c.

For the converse, suppose that $Ray(q)$ intersects E_c. Let u be the point of intersection. By definition of E_c, u is the highest point, on or below q', of any hyperplane of color c. Thus all hyperplanes of color c lie on or below q', hence all points of color c are in q^+.

Finally, if $Ray(q)$ intersects E_c at more than one point, then we have, on or below q', at least two distinct points of E_c of maximum x_d-coordinate, which is impossible. $\square$

By Lemma 3, it suffices to determine from the collection of envelopes of different colors, those that are intersected by $Ray(q')$; the number, k, of such intersected envelopes is also the number, I, of colors not in q^-. Efficient solutions are known for this ray-envelope intersection problem in $\mathbb{R}^2$ and $\mathbb{R}^3$.[17] In $\mathbb{R}^2$ it can be solved in $O(n \log n)$ space and $O(\log^2 n + k)$ query time. In $\mathbb{R}^3$, it can be solved in $O(n \log^2 n)$ space and $O(n^{1/2+\varepsilon} + k)$ query time, or in $O(n^{2+\varepsilon})$ space and $O(\log^2 n + k)$ query time. (Note that the approach itself is valid for any dimension d.)

Theorem 9. *A set, S, of n colored points in $\mathbb{R}^d$ can be preprocessed into a data structure, so that given any query hyperplane q, a generalized non-intersection query on S with an open halfspace of q can be answered with the following space and query time bounds: $O(n \log n)$ space and $O(\log^2 n + I)$ query time for $d = 2$; and*

$O(n \log^2 n)$ (resp. $O(n^{2+\varepsilon})$) space and $O(n^{1/2+\varepsilon} + I)$ (resp. $O(\log^2 n + I)$) query time for $d = 3$. Here I denotes the number of colors reported.

To handle the case where q is vertical w.r.t. direction x_d, we proceed as follows: For each direction x_i, $1 \leq i \leq d$, we construct the data structure of Theorem 9, by defining the duality transform $\mathcal{F}$ w.r.t. direction x_i; i.e., x_i plays the role of x_d in the definition given above for $\mathcal{F}$. Note that for any query q there is always some direction x_i w.r.t. which q is non-vertical. We use the corresponding data structure to answer the query. The performance bounds are asymptotically the same as those given in Theorem 9.

6.1. *A more efficient solution in $\mathbb{R}^2$ for few colors*

The above solution for $d = 2$ can be improved when t—the number of colors in S—is $O(\log n)$. As before, we may assume that the query q (a line) is non-vertical. Our approach is to represent each set of colored points by a sparse subset consisting of their convex hull and establish, in terms of the different hulls, a necessary and sufficient condition under which a color should be reported. Then, using geometric duality, we transform the generalized non-intersection problem to a standard one and solve the latter.

For each color c, let CH_c denote the convex hull of the points of color c. The boundary of CH_c can be partitioned into an *upper chain*, U_c, and a *lower chain*, L_c, whose endpoints are the leftmost and rightmost vertices of CH_c.

Lemma 4. *There are no points of color c in q^- if and only if L_c is contained in the closed halfplane q^+.*

Proof. If there are no points of color c in q^-, then all points of color c are contained in q^+. Therefore, CH_c, hence L_c, is contained in q^+.

Conversely, if L_c is contained in q^+, then so is U_c, since no point of U_c can be below L_c. Thus, CH_c is in q^+, hence all points of color c are in q^+. □

In view of Lemma 4, our goal now is to report the distinct colors, c, such that L_c is in q^+. We map this problem to a standard intersection problem in the dual space, via the duality transformation $\mathcal{F}$ described earlier.

It is well-known that, under this mapping, L_c maps to an infinite convex chain L'_c, which is downwards convex—the first and last edges of L'_c are semi-infinite lines. (This follows, for instance, from the discussion in Ref. 24 (pages 246–248).) As before, the line q dualizes to a point q' and the halfplane q^+ dualizes to a vertical ray, $Ray(q)$, which emanates from q' and is directed downwards. Moreover, L_c is in q^+ if and only if $Ray(q)$ intersects L'_c. Note that, due to convexity, if $Ray(q)$ intersects L'_c, then it does so exactly once. Thus, our generalized non-intersection problem has been transformed to a standard intersection problem, where the goal is to simply report all the dual chains that are intersected by $Ray(q)$. Note that the output size, k of the standard problem is the same as the output size, I, of the generalized problem.

We solve the ray-chain intersection problem as follows. We associate with each dual chain L_c the color c. We draw a vertical line through each vertex and intersection point in the collection of the dual chains of different colors. Within each strip so obtained, the chain segments are non-intersecting, so they can be totally ordered. Given $Ray(q)$, we locate, via binary search, the strip containing the x-coordinate of q', and then do a second binary search within this strip using the y-coordinate of q'. We report the color of each chain for which there is a segment in the strip that is on or below q'. The query time is $O(\log n + k) = O(\log n + I)$.

What is the space bound? Any two chains, corresponding to colors c and c', intersect $O(s + s')$ times where s and s' are the sizes of the chains corresponding to colors c and c', respectively. Summed over all pairs (c, c'), this gives $O(tn)$, where t is the number of different colors in the input set.

Theorem 10. *A set, S, of n colored points in $\mathbb{R}^2$ can be stored in a data structure of size $O(tn)$, so that for any query line, q, a generalized non-intersection query on S with an open halfplane of q can be answered in time $O(\log n + I)$, where I is the number of colors reported and t is the number of different colors in S.*

7. Queries Involving Convex Polygons

We consider the following problem: Let S be a set of convex polygons in $\mathbb{R}^2$, with a total of n vertices. (We take a polygon to consist of both the boundary and the interior.) We wish to preprocess S into a data structure so that the polygons P in S that are not intersected by a query convex polygon, q, can be reported output-sensitively, i.e., in time proportional to the number of missed polygons. (Note that the need for output-sensitivity precludes any approach which involves testing individual vertices/edges in S for non-intersection with q.)

Observe that the problem makes no explicit mention of colors. Colors are implicit in the way the vertices in S are aggregated into polygons. We will see later that an efficient solution to this problem will involve answering certain generalized queries on colored points.

We first consider the case where q has constant size (Section 7.1); subsequently, we relax this requirement (Section 7.2). We also show how these results can be used to solve an all-pairs non-intersection problem (Section 7.3).

7.1. *Querying with a constant-size convex polygon q*

Our approach is based on the notion of a separating line. Let X and Y be convex polygons in the plane. We say that a line L *separates* X from Y if X is contained in a closed halfplane defined by L and Y is contained in the complementary open halfplane, or vice versa; the line L is called a *separating line* for X and Y. We will use the following lemma. (Variants of this lemma have appeared in the literature without proof—see, for example, Refs. 11 and 21—and the result appears to be part of the folklore. For completeness, we give a proof in the Appendix.)

Lemma 5. *Convex polygons X and Y in the plane are disjoint if and only if there is an edge e of X or Y such that the line, $L(e)$, supporting e separates X from Y.*

Let q be a constant-size query polygon. By Lemma 5, our non-intersection problem is equivalent to finding all polygons $P \in S$ such that P and q have a separating line $L(e)$, for some edge e in P or q. (Note that if P and q are disjoint then there can be many such edges e.)

Given q, we first find all polygons $P \in S$ for which there is a supporting line $L(e)$, for some edge e of q, that separates P from q. Next, we find all polygons $P \in S$ for which there is a supporting line $L(e)$, for some edge e of P, that separates P from q. As we will see, the two approaches are quite different.

7.1.1. *Non-intersecting polygons identified by separating lines from q*

We color each polygon $P \in S$ with a distinct color and associate this color with its vertices. We then preprocess the colored vertices of all the polygons into a data structure, D, for generalized halfplane non-intersection queries (Section 6). For each edge $e \in q$, we compute $L(e)$ and query D with the closed halfplane below (resp. above) $L(e)$ if e belongs to the upper chain $UC(q)$ (resp. lower chain $LC(q)$). We report the union of the colors (i.e., polygons) so found.

Lemma 6. *The query algorithm reports a polygon $P \in S$ if and only if there is an edge $e \in q$ such that $L(e)$ separates P from q. It runs in $O(\log^2 n + I)$ time and uses $O(n \log n)$ space.*

Proof. We first establish correctness. Suppose that there is an edge e in q such that $L(e)$ separates P from q. Without loss of generality, let $e \in UC(q)$. Thus, P is in the open halfplane above q. Therefore, a generalized non-intersection query with the closed halfplane below q will report the color of P, hence P.

For the converse, suppose that P is reported. Thus, P is reported by the halfplane query corresponding to the supporting line $L(e)$ of at least one edge e of q. Consider such an e and assume that $e \in UC(q)$. Thus, P is reported when querying with the closed halfplane below $L(e)$, which contains q. Since P is reported, it must be contained in the open halfplane above $L(e)$. Thus, $L(e)$ separates P from q.

The query time and space bounds follow from Theorem 9. (Note that a polygon P can be reported multiple times, corresponding to separating lines defined by different edges of q, but the multiplicity is upper-bounded by a constant, since q has constant size. Thus, the query time is output-sensitive.) □

7.1.2. *Non-intersecting polygons identified by separating lines from P*

The previous approach cannot be used here since the number of polygons P, and their total size, is too large. Instead, we will consider the problem in the dual space and solve it as a certain generalized intersection (as opposed to non-intersection) searching problem. We first introduce the latter problem.

Generalized grounded trapezoidal range search

By a *grounded trapezoid* we mean a three-sided trapezoid which has left and right sides that are supported by vertical lines, ℓ and r, respectively, a top side which is supported by some non-vertical line t, and whose bottom side does not exist (i.e., the trapezoid is unbounded downwards). The *generalized grounded trapezoidal range search* problem is to preprocess a set S of n colored points in the plane so that the distinct colors of the points that are contained in a grounded query trapezoid, q, can be reported output-sensitively. We show how to obtain an efficient solution to this problem.

Let ℓ^+ be the closed halfplane to the right of ℓ, r^- the closed halfplane to the left of r, and t^- the closed halfplane below t. Let q' be the trapezoid defined by the intersection of ℓ^+ and t^-. We first obtain an efficient solution for the query q'. Since q' is the intersection of $O(1)$ halfplanes, by Corollary 7.3(ii) in Ref. 30 the query on S with q' can be answered in $O(\log n + I)$ time and $O(n^{2+\varepsilon})$ space, where I is the output size (i.e., number of colors reported) for this query and $\varepsilon > 0$ is an arbitrarily small constant. Call this data structure D_1.

It is possible to further improve this solution, using a result presented in Ref. 18. First, note that q' can also be viewed as simply the halfplane t^- to which is applied the semi-infinite range restriction $[x_\ell, \infty)$, where x_ℓ is the x-coordinates of ℓ. Let D_2 denote the data structure given in Ref. 17 for answering a generalized halfplane range search query using t^- alone (i.e., without the range restriction). This structure uses $O(n \log n)$ space and has a query time of $O(\log^2 n + I)$, where I is the output size for this query. Applying the result in Ref. 18 to D_1 and D_2 yields a new data structure, D_3, for query q', which uses just $O(n^{1+\varepsilon})$ space and has a query time of $O(\log^2 n + I)$.

Let us now consider the original query q. Since q, too, is the intersection of a constant number of halfplanes, i.e., ℓ^+, r^-, and t^-, we can again apply Corollary 7.3(ii) in Ref. 30 to obtain a data structure, D_4, to answer the query q on S in $O(\log n + I)$ time and $O(n^{2+\varepsilon})$ space. Note that we can view q as the query q' with the range restriction $(-\infty, x_r]$, where x_r is the x-coordinates of r. We can now apply the result in Ref. 18 to D_3 and D_4 to obtain our final data structure, D_5, for the generalized grounded trapezoidal range search problem on S with q. This structure uses $O(n^{1+\varepsilon})$ space and has a query time of $O(\log^2 n + I)$.

Note that a similar discussion also applies to query trapezoids that have a bottom side but no top side (i.e., unbounded upwards). We will use both types in our later discussion. From the above discussion, we have the following:

Lemma 7. *Let S be a set of n colored points in the plane. S can be preprocessed into a data structure of size $O(n^{1+\varepsilon})$ so that the I distinct colors of the points of S that are contained in a grounded query trapezoid, q, can be reported in $O(\log^2 n + I)$. Here $\varepsilon > 0$ is an arbitrarily small constant.*

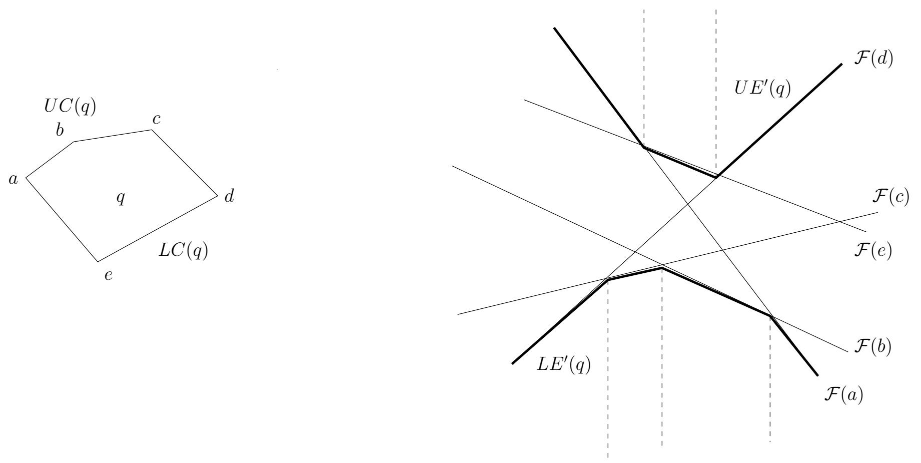

Fig. 1. Dualization of the query polygon, q, and the resulting grounded query trapezoids. The lower and upper envelopes, $LE'(q)$ and $UE'(q)$, respectively, are shown in heavy lines. (The figure is meant to be schematic and is not to scale.)

We now return to the main problem of this section, i.e., computing disjoint pairs (P, q) that are identified by separating lines supporting edges of polygons $P \in S$. We color each polygon $P \in S$ with a different color. For each P, we dualize the supporting lines of the edges on its lower chain to points and associate with each point the color of P. We preprocess these points into an instance, D_L, of the data structure of Lemma 7 for query trapezoids that are unbounded downwards. Symmetrically, we build a second data structure, D_U, corresponding to the upper chains of the polygons P to answer queries with trapezoids that are unbounded upwards.

Given q, we proceed as follows: We first dualize the vertices of its upper chain to lines and compute their lower envelope $LE'(q)$.[c] We then partition the space below the upper chain into grounded trapezoids whose upper edges correspond to edges of the lower envelope (so the trapezoids are unbounded downwards)—see Figure 1. We then perform a grounded trapezoidal query on D_L with each of these trapezoids. Symmetrically, we query D_U with a set of upward-unbounded trapezoids obtained from the upper envelope, $UE'(q)$, of the lines that are dual to vertices on the lower chain of q. Finally, we return the union of the colors (polygons) reported by all the queries.

Lemma 8. *The query algorithm reports a polygon $P \in S$ if and only if there is an edge $e \in P$ such that $L(e)$ separates P from q. It runs in $O(\log^2 n + I)$ time and uses $O(n^{1+\varepsilon})$ space.*

[c]We simply use a prime (i.e., $'$) to indicate that the lower envelope $LE'(q)$ is in the dual space; this is in lieu of more cumbersome notation along the lines of $LE(\mathcal{F}(UC(q)))$. Similarly for the upper envelope $UE'(q)$.

Proof. We first establish correctness. Suppose that there is an edge $e \in P$ such that $L(e)$ separates P from q. Without loss of generality, assume that e belongs to the lower chain, $LC(P)$, of P; thus q is contained in the closed halfplane below $L(e)$. Since every vertex, v, of q is on or below $L(e)$, by the properties of the duality transform $\mathcal{F}$ (Section 6), in the dual space every line $\mathcal{F}(v)$ is on or above the point $\mathcal{F}(L(e))$. In particular, the lower envelope of the lines $\mathcal{F}(v)$ is on or above the point $\mathcal{F}(L(e))$. Thus, the generalized grounded trapezoidal query with one of the trapezoids defined by an edge of the lower envelope will report the color of P, hence P.

For the converse, suppose that (the color of) P is reported. Thus, there is some point of that color in the dual space which is reported when querying with a grounded trapezoid. This point is the dual of the supporting line of an edge e of P. Without loss of generality, assume that the query trapezoid is unbounded downwards, i.e., it is defined by an edge of the lower envelope of the lines that are dual to the vertices of q. Thus, all the dual lines are above the point, so in the primal space all the vertices of q are on or below $L(e)$. Thus, $L(e)$ separates P from q.

The space bound follows from Lemma 7 and the fact that the total number of colored points to be stored is n. For the query time, note that the size of each envelope is a constant, since q has constant size. Thus, there are $O(1)$ generalized queries with grounded trapezoids and the time for any query is $O(\log^2 n + I)$. (A color may get reported in more than one query, but its multiplicity across all queries is a constant since q has constant size and since the queries are generalized.) $\square$

7.1.3. *Putting it all together*

Lemmas 6 and 8, taken together with Lemma 5, allow us to now solve the problem stated at the beginning of Section 7, for a query polygon, q, of constant size, as stated in the following theorem.

Theorem 11. *Let S be a set of convex polygons in $\mathbb{R}^2$ with a total of n vertices. S can be preprocessed into a data structure of size $O(n^{1+\varepsilon})$ so that for any query convex polygon, q, of constant size, the polygons in S that do not intersect q can be reported in time $O(\log^2 n + I)$ where I is the output size. Here $\varepsilon > 0$ is a constant.*

7.2. *Querying with a general convex polygon*

The approach of Section 7.1 will not yield an efficient solution if the size of q is not a constant, since a polygon of S that is disjoint from q can be reported a non-constant number of times. Instead, we solve the problem in the dual space, by reducing it to a (standard) intersection searching problem on certain line segments.

We will need the following lemmas, the first of which is straightforward from the properties of the duality transform $\mathcal{F}$ (Section 6).

Lemma 9. *Let X be a convex polygon and let e be any edge of X. Then e is in $UC(X)$ (resp. $LC(X)$) if and only if $\mathcal{F}(L(e))$ is a vertex of $LE'(X)$ (resp. $UE'(X)$).*

We say that two convex chains *intersect properly* if the chains intersect and cross each other (i.e., they do not merely touch one another).

Lemma 10. *Convex polygons X and Y are disjoint if and only if either $UE'(X)$ and $LE'(Y)$ intersect properly or $LE'(X)$ and $UE'(Y)$ intersect properly.*

Proof. Suppose that X and Y are disjoint. By Lemma 5, there is an edge e of X or Y such that $L(e)$ separates X from Y. Without loss of generality, let e be in $UC(X)$. Thus, all vertices of Y are above $L(e)$, so the point $\mathcal{F}(L(e))$ is above the lines dual to the vertices of Y. Hence, $\mathcal{F}(L(e))$ is above $UE'(Y)$. Moreover, by Lemma 9, $\mathcal{F}(L(e))$ is on $LE'(X)$. Since $LE'(X)$ and $UE'(Y)$ are infinite, it follows that they intersect and do so properly.

Conversely, suppose that $LE'(X)$ and $UE'(Y)$ intersect properly. Then either there is a vertex, u, of $LE'(X)$ above $UE'(Y)$ or there is a vertex, v, of $UE'(Y)$ below $LE'(X)$. Without loss of generality, suppose that the former case holds. Since u is on $LE'(X)$, by Lemma 9 there is an edge e of $UC(X)$ such that $\mathcal{F}(L(e)) = u$. Since u is above $UE'(Y)$, it is above all the lines that are dual to the vertices of Y. Thus, $L(e)$ is below all the vertices of Y. It follows that X and Y are disjoint. $\square$

Given the set, S, of input polygons, we compute the envelopes $LE'(P)$ and $UE'(P)$ for each $P \in S$. We then store the line-segments of $LE'(P)$ (resp. $UE'(P)$, for all $P \in S$, in a data structure, $D_\ell(S)$ (resp. $D_u(S)$) for (standard) segment intersection queries. Given a query convex polygon, q, we compute $LE'(q)$ and $UE'(q)$. We then query $D_u(S)$ (resp. $D_\ell(S)$) with the line-segments of $LE'(q)$ (resp. $UE'(q)$) and report the polygons associated with the segments returned by the query.

For $D_\ell(S)$ and $D_u(S)$ we can use the structure proposed in Ref. 23 or in Ref. 2. The former approach stores a set of n line-segments in $O(n^{2+\varepsilon})$ space and reports the k segments that are intersected by a query segment in time $O(\log n + k)$. The latter uses $O(m)$ space and answers queries in $O(n^{1+\varepsilon}/\sqrt{m} + k)$ time, for any parameter m, $n \leq m \leq n^2$, and any constant $\varepsilon > 0$. We conclude:

Theorem 12. *Let S be a set of convex polygons in $\mathbb{R}^2$ with a total of n vertices. S can be preprocessed into a data structure of size $O(n^{2+\varepsilon})$ (resp. $O(m)$) such that for any query convex polygon, q, the polygons in S that are not intersected by q can be reported in time $O(|q| \log n + I)$ (resp. $O(|q| n^{1+\varepsilon}/\sqrt{m} + I)$), where I is the output size, $\varepsilon > 0$ is any constant, and m is a parameter, $n \leq m \leq n^2$.*

Proof. Correctness follows from Lemma 10. For the performance bounds note that (i) the total number of envelope segments stored is $\Theta(n)$, and (ii) there are exactly two points at which $LE'(P)$ and $UE'(q)$, and similarly $UE'(P)$ and $LE'(q)$, intersect. The latter implies that a query on $D_\ell(S)$ and on $D_u(S)$ will report any polygon P only a constant number of times. This allows us to use a standard intersection search structure and yet get output-sensitive query times. The space and query time bounds now follow from the corresponding bounds for the data structures of Refs. 23 and 2. □

7.3. *Application to all-pairs non-intersection*

We describe briefly how the results of Section 7.1 and 7.2 can be used to solve the following problem: Given a set S of convex polygons, with a total of n vertices, report output-sensitively all pairs of polygons that are non-intersecting. (Note that this is a single-shot problem, not a query problem.)

For each convex polygon $P \in S$, we compute the segments of $UE'(P)$ and $LE'(P)$ and color them red and blue, respectively. We then apply the red-blue segment intersection algorithm given in Ref. 1 to compute all intersections involving a red segment and a blue segment. For each pair of intersecting edges, we report the corresponding polygons as a non-intersecting pair.

The correctness of the method follows from Lemma 10. The running time and space of the red-blue segment intersection algorithm are $O(n^{4/3} \cdot \text{polylog}(n) + k)$ and $O(n^{4/3}/\text{polylog}(n))$, respectively, where k is the number of pairs of intersecting segments. Note that $k = \Theta(I)$, where I is the number of pairs of non-intersecting polygons, since the number of intersections between an upper envelope and a lower envelope is a constant.

If the input polygons are all of constant size, then we can apply the approach underlying Lemma 6, using, in turn, each polygon from S as a query polygon.

Theorem 13. *Let S be a set of convex polygons in $\mathbb{R}^2$, with a total of n vertices. It is possible to report all non-intersecting pairs of polygons in S output-sensitively, in $O(n^{4/3} \cdot polylog(n) + I)$ time and $O(n^{4/3}/polylog(n))$ space, where I is the output size. If the size of each polygon is upper-bounded by a constant, then the problem can be solved in $O(n \log^2 n + I)$ time and $O(n \log n)$ space.*

8. Conclusion

We have considered a new class of non-intersection queries on aggregated geometric data and have presented efficient solutions to a number of these problems (Table 1). Our solutions are based on a combination of techniques including geometric duality, sparsification, persistence, filtering search, and pruning.

We have focused on reporting problems, since a generalized non-intersection counting problem can be solved easily given a solution to the corresponding generalized intersection counting problem: we simply subtract from the total number, of colors in the input set the count returned by the intersection counting problem and output this as the answer to the non-intersection counting problem.

Some of our results have allowed for dynamic or semi-dynamic updates. However, dynamization for generalized non-intersection queries is largely open and an area which we are pursuing. Another direction for further work is to consider generalized non-intersection queries in higher dimensions.

Acknowledgements

We thank the two anonymous referees for several suggestions and corrections that helped improve the paper considerably.

Research of PG was supported, in part, by grants DST/INT/US/NSF-RPO-0155/04 and SR/S3/EECE/22/2004 from the Department of Science and Technology, Govt. of India. Research of RJ was Research supported, in part, by the National Science Foundation under grant INT–0422775. Research of MS was supported by NSERC.

References

1. P.K. Agarwal. Partitioning arrangements of lines: II. Applications. *Discrete and Computational Geometry*, 5, 1990, 533–573.
2. P.K. Agarwal and M. Sharir. Applications of a new space partitioning technique. *Discrete and Computational Geometry*, 9, 1993, 11–38.
3. P.K. Agarwal, S. Govindarajan, and S. Muthukrishnan. Range Searching in Categorical Data: Colored Range Searching on Grid, *Proceedings of the 10th European Symposium on Algorithms*, Lecture Notes in Computer Science, Vol. 2461, Springer-Verlag, 2002, 323–334.
4. P.K. Agarwal and M. van Kreveld. Connected component and simple polygon intersection searching. *Algorithmica* 15(6), 1996, 626–660.
5. P. Bozanis, N. Kitsios, C. Makris, and A. Tsakalidis. New upper bounds for generalized intersection searching problems. *Proceedings of the 22nd ICALP*, Lecture Notes in Computer Science, Vol. 944, Springer-Verlag, 1995, 464–475.
6. P. Bozanis, N. Kitsios, C. Makris, and A. Tsakalidis. Red-Blue intersection reporting for objects of non-constant size. *The Computer Journal*, 39(6), 1996, 541–546.
7. P. Bozanis, N. Kitsios, C. Makris, and A. Tsakalidis. New results on intersection query problems. *The Computer Journal*, 40(1), 1997, 22–29.
8. B. Chazelle. Filtering search: A new approach to query-answering. *SIAM J. Comput*, 15(3), 1986, 703–724.
9. S.W. Cheng, R Janardan. Efficient dynamic algorithms for some geometric intersection problems. *Information Processing Letters* 36(5), 1990, 251–258.
10. J. Czyzowicz, E. Rivera-Campo, J. Urrutia, and J. Zaks. Separating convex sets in the plane. *Discrete and Computational Geometry*, 7, 1992, 189–195.
11. J. Czyzowicz, E. Rivera-Campo, J. Urrutia, and J. Zaks. Separating convex sets on the plane. Manuscript at `www.math.unam.mx/~urrutia/online_papers/Separating.pdf`.
12. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*, Springer Verlag, 2000.
13. 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.
14. H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag (1987).
15. P. Gupta. Efficient algorithms and data structures for geometric intersection problems. *Ph.D. dissertation*, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1995.
16. P. Gupta, R. Janardan, and M. Smid. Further results on generalized intersection searching problems: counting, reporting, and dynamization, *Journal of Algorithms*, 19, 1995, 282–317.
17. P. Gupta, R. Janardan, and M. Smid. Algorithms for generalized halfspace range

searching and other intersection searching problems, *Computational Geometry: Theory and Applications*, 5, 1996, 321–340.
18. P. Gupta, R. Janardan, and M. Smid. A technique for adding range restrictions to generalized searching problems. *Information Processing Letters*, 64, 1997, 263–269.
19. P. Gupta, R. Janardan and M. Smid. Algorithms for some intersection searching problems involving circular objects, *International Journal of Mathematical Algorithms*, 1, 1999, 35–52.
20. P. Gupta, R. Janardan and M. Smid. Computational Geometry: Generalized Intersection Searching *Handbook of Data Structures*, S. Sahni and D. Mehta, eds., CRC Press, 2004, Ch. 64, 1–17.
21. K. Hope and M. Katchalski. Separating plane convex sets. *Mathematica Scandinavica*, 66, 1990, 44–46.
22. R. Janardan and M. Lopez. Generalized intersection searching problems. *International Journal on Computational Geometry & Applications*, 3, 1993, 39–69.
23. V. Koltun. Segment intersection searching problems in general settings. *Discrete and Computational Geometry*, 30, 2003, 25–44.
24. K. Mehlhorn. *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry.* Springer-Verlag (1984).
25. Information culled from the Investment Company Institute's Mutual Fund Factbook, 2003 (`www.ici.org`), and Yahoo.com (`finance.yahoo.com`).
26. E.M. McCreight. Priority search trees, *SIAM Journal of Computing*, 14(2), 1985, 257–276.
27. S. Muthukrishnan. Efficient algorithms for document retrieval problems. *Proceedings of the 13th Annual Symposium on Discrete Algorithms*, 2002, 657–666.
28. Q. Shi and J. JaJa. Optimal and near-optimal algorithms for generalized intersection reporting on pointer machines. Technical Report CS–TR–4542, Institute for Advanced Computer Studies (UMIACS), University of Maryland, College Park, MD, 2003
29. T.G. Szymanski and C.J. van Wyk. *Layout Analysis and Verification*, in Physical Design Automation of VLSI Systems, B. Preas and M. Lorenzetti, eds., Benjamin/Cummings (1988), 347–407.
30. M. van Kreveld. New results on data structures in computational geometry, *Ph.D. Thesis*, Department of Computer Science, University of Utrecht, the Netherlands, 1992.

Appendix A. Proof of Lemma 5

Proof. Suppose that a separating line $L(e)$ exists, for some e in X or Y. Then X is contained in a closed halfplane of $L(e)$ and Y is contained in the complementary open halfplane. Since these halfplanes are disjoint, it follows that X and Y are disjoint.

For the converse, suppose that X and Y are disjoint. We will show that $L(e)$ exists for some e in X or Y. Let $x \in X$ and $y \in Y$ be the closest points of X and Y. Note that the interior of the segment $s = \overline{xy}$ is disjoint from X and Y; note also that s has positive length, since X and Y are disjoint. Without loss of generality, assume that s is vertical and let x be above y. There are three cases:

(a) x **and** y **are vertices:** Note that since x and y are the closest points of X and Y, no part of X can be below the horizontal line through x and no part of Y

can be above the horizontal line through y.

Assume, without loss of generality, that the interior angle at x is greater than or equal to the interior angle at y. Let e_1 and e_2 be the edges of X incident with x. The lines $L(e_1)$ and $L(e_2)$ define four semi-infinite wedges. Consider the wedge that is vertically below x and imagine that a copy, W, of this wedge is translated to y, such that its apex coincides with y. There are three subcases:

- *Both edges of Y incident with y are contained in W:* (See Figure 2a(i).) By convexity, all of Y is in W. In this case, there are two possibilities: If e_1 and e_2 are on opposite sides of the vertical line through s, then both $L(e_1)$ and $L(e_2)$ are separating lines for X and Y. (See the left part of Figure 2a(i).) If e_1 and e_2 are on the same side of the vertical line through s, then exactly one of $L(e_1)$ and $L(e_2)$ is a separating line for X and Y. (See the right part of Figure 2a(i); here $L(e_1)$ is a separating line. The symmetric case is omitted).
- *Exactly one of the edges of Y incident with y is contained in W:* (See Figure 2a(ii).) In this case, $L(e_1)$ is a separating line for X and Y. (The symmetric case is omitted.)
- *Neither of the edges of Y incident with y is contained in W:* (See Figure 2a(iii).) Note that since the interior angle at x is greater than or equal to the interior angle at y, it is not possible for the edges of Y incident with y to be on opposite sides of W. In this case, $L(e_1)$ is a separating line for X and Y. (The symmetric case is omitted.)

(b) **x is a vertex and y is in the interior of an edge:** The edge containing y must be perpendicular to s; otherwise, x and y cannot be the closest points of X and Y. Then $L(e)$ is a separating line for X and Y, where e is the edge containing y. (See Figure 2b.)

(c) **x and y are both in the interiors of edges:** The edges containing x and y must be perpendicular to s; otherwise, x and y cannot be the closest points of X and Y. Then $L(e)$ and $L(e')$ are separating lines for X and Y, where e and e' are the edges containing y and x, respectively. (See Figure 2c.) We note that this case can also be reduced to case (b). □

We remark that in Refs. 11 and 21 a slightly different notion of separability is used, where both halfplanes in question are taken to be closed. Under this definition, it is possible that a separating line exists even if X and Y are not disjoint (for instance, two triangles that are almost disjoint, except that a vertex of one triangle is in the interior of an edge of the other). Consequently, only one direction of Lemma 5 is stated (without proof) in Refs. 11 and 21. (The journal version of Ref. 11 i.e., Ref. 10, does not contain this result.)

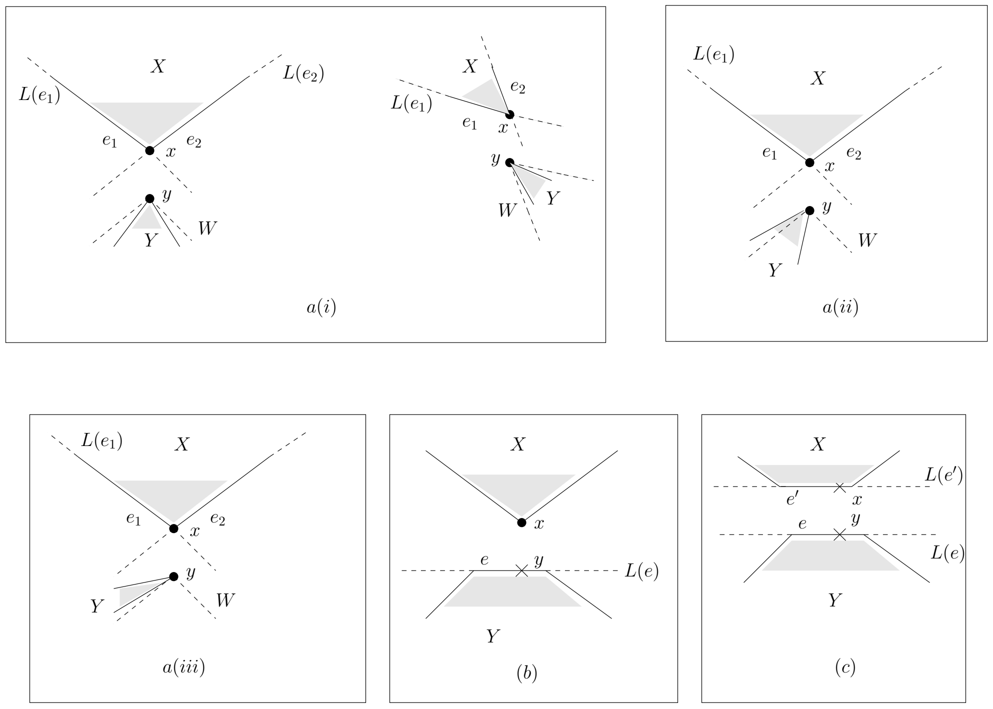

Fig. 2. The various cases in the proof of Lemma 5.