1

Computational Geometry: Generalized (or Colored) Intersection Searching

Prosenjit Gupta
Heritage Institute of Technology, Kolkata

Ravi Janardan
University of Minnesota, Minneapolis

Saladi Rahul
University of Minnesota, Minneapolis

Michiel Smid
Carleton University, Ottawa

1.1 Geometric intersection searching problems

Problems arising in diverse areas, such as VLSI layout design, database query-retrieval, robotics, and computer graphics can often be formulated as *geometric intersection searching problems*. In a generic instance of such a problem, a set, S, of geometric objects is to be preprocessed into a suitable data structure so that given a query object, q, we can answer efficiently questions regarding the intersection of q with the objects in S. The problem comes in four versions, depending on whether we want to report the intersected objects or simply count their number—the *reporting* version and the *counting* version, respectively—and whether S remains fixed or changes through insertion and deletion of objects—the *static* version and the *dynamic* version, respectively. In the dynamic version, which arises very often owing to the highly interactive nature of the above-mentioned applications, we wish to perform the updates more efficiently than simply recomputing the data structure from scratch after each update, while simultaneously maintaining fast query response times. We call these problems *standard* intersection searching problems in order to distinguish them from the *generalized* intersection searching problems that are the focus of this chapter. Due to their numerous applications, standard intersection searching problems have been the subject of much study and efficient solutions have been devised for many of them (see, for instance, [6, 21] and the references therein).

0-8493-8597-0/01/\$0.00+\$1.50

The efficiency of a standard intersection searching algorithm is measured by the space used by the data structure, the query time, and, in the dynamic setting, the update time. In a counting problem, these are expressed as a function of the input size n (i.e., the size of S); in a reporting problem, the space and update time are expressed as a function of n, whereas the query time is expressed as a function of both n and the output size k (i.e., the number of intersected objects) and is typically of the form $O(f(n)+k)$ or $O(f(n)+k \cdot g(n))$, for some functions f and g. Such a query time is called *output-sensitive*.

1.1.1 Generalized intersection searching

In many applications, a more general form of intersection searching arises: Here the objects in S come aggregated in disjoint groups and of interest are questions regarding the intersection of q with the groups rather than with the objects. (q intersects a group if and only if it intersects some object in the group.) In our discussion, it will be convenient to associate with each group a different color and imagine that all the objects in the group have that color. Then, in the *generalized reporting* (resp., *generalized counting*) problem, we want to report (resp., count) the distinct colors of the objects intersected by q; in the dynamic setting, an object of some (possibly new) color is inserted in S or an object in S is deleted. Note that the generalized problem reduces to the standard one when each color class has cardinality 1. (We remark that the generalized problems discussed here are also sometimes referred to as *colored* problems; we use the two terms interchangeably.)

We give some examples of such generalized problems:

- Consider a database of mutual funds which contains for each fund its annual total return and its beta (a real number measuring the fund's volatility). Thus each fund can be represented as a point in two dimensions. Moreover, funds are aggregated into groups according to the fund family they belong to. A typical query is to determine the families that offer funds whose total return is between, say, 15% and 20%, and whose beta is between, say, 0.9 and 1.1. This is an instance of the generalized 2-dimensional range searching problem. The output of this query enables a potential investor to initially narrow his/her search to a few families instead of having to plow through dozens of individual funds (all from the same small set of families) that meet these criteria. (From a database perspective, this query is similar to an SQL query with a GROUP-BY clause.)
- In the Manhattan layout of a VLSI chip, the wires (line segments) can be grouped naturally according to the circuits they belong to. A problem of interest to the designer is determining which circuits (rather than wires) become electrically connected when a new wire is added. This is an instance of the generalized orthogonal segment intersection searching problem.

One approach to solving a generalized problem is to try to take advantage of solutions known for the corresponding standard problem. For instance, we can solve a generalized reporting problem by first determining the objects intersected by q (a standard reporting problem) and then reading off the distinct colors. However, the query time can be very high since q could intersect $k = \Omega(n)$ objects but only $O(1)$ distinct colors. For a generalized reporting problem, we seek query times that are sensitive to the number, i, of distinct colors intersected, typically of the form $O(f(n) + i)$ or $O(f(n) + i \cdot g(n))$, where f and g are polylogarithmic. (This is attainable using the approach just described if each color class has cardinality $O(1)$. On the other hand, if there are only $O(1)$ different color classes, we could simply run a standard algorithm on each color class in turn, stopping as soon as an intersection is found and reporting the corresponding color. The real challenge is when the number of color classes and the cardinalities of the color classes are not constants, but rather are (unknown) functions of n; throughout, we will assume this to be the case.) For a generalized counting problem, the situation is worse; it is not even clear how one can extract the answer for such a problem from the answer (a mere count) to the corresponding standard problem. One could, of course, solve the corresponding reporting problem and then count the colors, but this is not efficient. Thus it is clear that different techniques are needed.

In this chapter, we describe the research that has been conducted over the past two decades on generalized intersection searching problems. We begin with a brief review of known results and then discuss a variety of techniques for these problems. For each technique, we give illustrative examples and provide pointers to related work. We conclude with a discussion of possible directions for further research.

1.2 Summary of known results

Generalized intersection searching problems were introduced by Janardan and Lopez in [33]. Subsequent work in this area may be found in [7, 8, 12, 13, 14, 20, 24, 25, 26, 27, 28, 29, 30, 31, 34, 35, 36, 37, 43, 45, 46, 47, 48, 50, 52], among others. In this section, we give a broad overview of the work on these problems to date; details may be found in the cited references.

1.2.1 Axes-parallel objects

In [33], efficient solutions were given for several generalized reporting problems, where the input objects and the query were axes-parallel. Examples of such input/query pairs considered include: points/interval in $\mathbb{R}^1$; line segments/segment, points/rectangle, and rectangles/rectangle, all in $\mathbb{R}^2$; and rectangles/points in $\mathbb{R}^d$, where $d \geq 2$ is a constant. Several of these results were further extended in [27] to include counting and/or dynamic reporting, and new results were presented for input/query pairs such as intervals/interval in $\mathbb{R}^1$, points/quadrant in $\mathbb{R}^2$, and points/rectangle in $\mathbb{R}^3$. Furthermore, a new type of counting problem, called a *type-2 counting problem* was also introduced, where the goal was to count for each color intersected the number of objects of that color that are intersected. In [12], improved solutions were given for counting and/or reporting problems involving points/interval in $\mathbb{R}^1$, points/rectangle in $\mathbb{R}^2$, and line segments/segment in $\mathbb{R}^2$.

In [20, 24] variants of the reporting problem were considered for points/rectangle in $\mathbb{R}^2$, with the goal of reporting those colors that appeared "many" times in the output. Specifically, in [24], an efficient dynamic algorithm was given for reporting each color for which the number of points of that color in the query rectangle was more than a user-specified fraction of the total number points in the query rectangle. On the other hand, in [20], an efficient (approximation) algorithm was given for reporting each color such that the number of points of that color in the query rectangle was at least a user-specified fraction of the total number of points of that color in the input set. More recently, interesting connections have been shown between generalized counting problems and other problems. For instance, as shown in [37], standard range counting involving points/rectangle in $\mathbb{R}^2$ can be solved using an algorithm for generalized range counting on points/interval in $\mathbb{R}^1$. Also, as discussed at length in [34], it is possible to use an "offline" version of the generalized counting problem on points/rectangle in $\mathbb{R}^2$ to do sparse matrix multiplication.

1.2.2 Arbitrarily-oriented objects

Efficient solutions were given in [33] for generalized reporting on non-intersecting line segments using a query line segment. Special, but interesting, cases of intersecting line segments, such as when each color class forms a polygon or a connected component, were considered in [8]. Efficient solutions were given in [28] for input/query pairs consisting of points/halfspace in $\mathbb{R}^d$, points/fat-triangle, and fat-triangles/point in $\mathbb{R}^2$. (A *fat-triangle* is a triangle where each internal angle is at least a user-specified constant, hence "well-shaped".) Some of these results were improved subsequently in [12]. In [29], alternative bounds were obtained for the fat-triangle problems within the framework of a general technique for adding range restriction capability to a generalized data structure. Results were presented in [14] for querying, with a polygon, a set of polygons whose sides are oriented in at most a constant number of different directions, with a polygon. In [52], a general method was given for querying intersecting line segments with a segment and for querying points in $\mathbb{R}^d$ with a halfspace or a simplex. Generalized problems involving various combinations of circular objects (circles, discs, annuli) and points, lines, and line segments were considered in [30].

1.2.3 Problems on the grid

Problems involving document retrieval or string manipulation can often be cast in the framework of generalized intersection searching. For example, in the context of document retrieval, the following problem (among others) was considered in [43]: Preprocess an array of colored non-negative integers (i.e., points on the 1-dimensional grid) such that, given two indices into the array, each distinct color for which there is a pair of points in the index range at distance less than a specified constant can be reported efficiently. In the context of substring indexing, the following problem was considered in [25]: Preprocess a set of colored points on the 1-dimensional grid, so that given two non-overlapping intervals, the list of distinct colors that occur in both intervals can be reported efficiently. I/O efficient algorithms were given in the standard external memory model [53] for this problem. See [44, 45] and references therein for a discussion of more recent work on document retrieval and string manipulation problems. Other grid-related work in this area includes [7], where efficient solutions were given for the points/rectangle and rectangles/point problems, under the condition that the input and query objects lie on a d-dimensional grid.

1.2.4 Single-shot problems

In this class of problems, we are given a collection of geometric objects and the goal is to report all pairs that intersect. Note that there is no query object as such here and no notion of preprocessing the input. As an example, suppose that we are given a set of convex polygons with a total of n vertices in $\mathbb{R}^2$, and we wish to report or count all pairs that intersect, with the goal of doing this in time proportional to the number of intersecting pairs (i.e., output-sensitively). If the number of polygons and their sizes are both functions of n (instead of one or the other being a constant), then, as discussed in [31], standard methods (e.g., testing each pair of polygons or computing all boundary intersections and polygon containments in the input) are inefficient. In [31], an efficient and output-sensitive algorithm is given for this problem. Each polygon is assigned a color and then decomposed into simpler elements, i.e., trapezoids, of the same color. The problem then becomes one of reporting all distinct color pairs (c_1, c_2) such that a trapezoid of color c_1 intersects one of color c_2. An improved algorithm was given subsequently in [5] for both $\mathbb{R}^2$ and $\mathbb{R}^3$.

Other related work on such colored single-shot problems may be found in [13, 35]. In [35], interesting connections between sparse matrix multiplication and colored single-shot problems are established.

1.2.5 External memory and word-RAM algorithms

The results discussed earlier (and in the rest of this chapter) are in the well-known RAM model or pointer machine model. Recently, generalized problems have also been considered in other machine models such as the external memory model and the word-RAM model. In the external memory model [9], the data resides primarily on disk, in blocks of some fixed size, data transfer (I/O) between disk and main memory happens in blocks, and space and query time are measured in terms of the number of blocks used and the number of I/O operations, respectively. In this model, efficient algorithms were first given in [46] (see also [47]) for generalized range search in $\mathbb{R}^2$, where the query rectangle is grounded, i.e., 3-sided. These results have been improved subsequently in [37]. A limitation of these results is that they report each color $O(1)$ times, instead of exactly once, which results in an additional overhead in the external memory model for removal of duplicate colors via sorting. (This is in contrast to internal memory algorithms, where duplicate removal is easy.) This limitation has recently been removed in [48] where each color is reported exactly once. The above papers also present results in the word-RAM model [32], where it is assumed that standard arithmetic and bitwise operations can be done in constant time on a computer word of some fixed length.

1.3 Techniques

We describe in some detail several techniques that have emerged over the past several years for generalized intersection searching. Briefly, these include: a geometric transformation-based approach, an approach based on generating a sparse representation of the input, an approach based on persistent data structures, a generic method that is applicable to any reporting problem, an approach for searching on a subset of the input satisfying a specified range restriction, and an approach that exploits the output size to gain efficiency. We illustrate each method with examples. Finally, we also discuss a "reverse" transformation method that shows an interesting connection between a standard counting and a generalized counting problem.

1.3.1 A transformation-based approach

We describe an approach for certain reporting and counting problems, which transforms the original generalized reporting/counting problem to an instance of a related standard reporting/counting problem on which efficient known solutions can be brought to bear. We illustrate this approach by considering the *generalized* 1*-dimensional range searching* problem, where the input consists of a set, S, of n colored points in $\mathbb{R}^1$ and the query, q, is an interval. Let S be a set of n colored points on the x-axis. We show how to preprocess S so that for any query interval q, we can solve efficiently the dynamic reporting problem, the static and dynamic counting problems, and the static type-2 counting problem. The solutions for the dynamic reporting problem and the static and dynamic counting problems are from [27]. The type-2 counting solution is from [12].

We now describe the transformation. For each color c, we sort the distinct points of that color by increasing x-coordinate. For each point p of color c, let $pred(p)$ be its predecessor of color c in the sorted order; for the leftmost point of color c, we take the predecessor to be the point $-\infty$. We then map p to the point $p' = (p, \mathit{pred}(p))$ in the plane and associate with it the color c. Let S' be the resulting set of points. Given a query interval $q = [l, r]$, we map it to the grounded rectangle $q' = [l, r] \times (-\infty, l)$.

LEMMA 1.1 There is a point of color c in S that is in $q = [l, r]$ if and only if there is a point of color c in S' that is in $q' = [l, r] \times (-\infty, l)$. Moreover, if there is a point of color c in q', then this point is unique.

Proof Let p' be a c-colored point in q', where $p' = (p, \mathit{pred}(p))$ for some c-colored point $p \in S$. Since p' is in $[l, r] \times (-\infty, l)$, it is clear that $l \leq p \leq r$ and so $p \in [l, r]$.

For the converse, let p be the leftmost point of color c in $[l, r]$. Thus $l \leq p \leq r$ and since $\mathit{pred}(p) \notin [l, r]$, we have $l > \mathit{pred}(p)$. It follows that $p' = (p, \mathit{pred}(p))$ is in $[l, r] \times (-\infty, l)$. We prove that p' is the only point of color c in q'. Suppose for a contradiction that $t' = (t, \mathit{pred}(t))$ is another point of color c in q'. Thus we have $l \leq t \leq r$. Since $t > p$, we also have $\mathit{pred}(t) \geq p \geq l$. Thus $t' = (t, \mathit{pred}(t))$ cannot lie in q'—a contradiction. ■

Lemma 1.1 implies that we can solve the generalized 1-dimensional range reporting (resp., counting) problem by simply reporting the points in q' (resp., counting the number of points in q'), without regard to colors. In other words, we have reduced the generalized reporting (resp., counting) problem in $\mathbb{R}^1$ to the standard grounded range reporting (resp., counting) problem in $\mathbb{R}^2$. In the dynamic case, we also need to update S' when S is updated. We discuss these issues in more detail below.

The dynamic reporting problem

Our data structure consists of the following: For each color c, we maintain a balanced binary search tree, T_c, in which the c-colored points of S are stored in increasing x-order. We maintain the colors themselves in a balanced search tree CT, and store with each color c in CT a pointer to T_c. We also store the points of S' in a *balanced priority search tree* (PST) [42]. (Recall that a PST on m points occupies $O(m)$ space, supports insertions and deletions in $O(\log m)$ time, and can be used to report the k points lying inside a grounded query rectangle in $O(\log m + k)$ time [42]. Although this query is designed for query ranges of the form $[l, r] \times (-\infty, l]$, it can be trivially modified to ignore the points on the upper edge of the range without affecting its performance.) Clearly, the space used by the entire data structure is $O(n)$, where $n = |S|$.

To answer a query $q = [l, r]$, we simply query the PST with $q' = [l, r] \times (-\infty, l)$ and report the colors of the points found. Correctness follows from Lemma 1.1. The query time is $O(\log n + k)$, where k is the number of points inside q'. By Lemma 1.1, $k = i$, and so the query time is $O(\log n + i)$.

Suppose that a c-colored point p is to be inserted into S. If $c \notin CT$, then we create a tree T_c containing p, insert $p' = (p, -\infty)$ into the PST, and insert c, with a pointer to T_c, into CT. Suppose that $c \in CT$. Let u be the successor of p in T_c. If u exists, then we set $\mathit{pred}(p)$ to $\mathit{pred}(u)$ and $\mathit{pred}(u)$ to p; otherwise, we set $\mathit{pred}(p)$ to the rightmost point in T_c. We then insert p into T_c, $p' = (p, \mathit{pred}(p))$ into the PST, delete the old u' from the PST, and insert the new u' into it.

Deletion of a point p of color c is essentially the reverse. We delete p from T_c. Then we delete p' from the PST and if p had a successor, u, in T_c then we reset $\mathit{pred}(u)$ to $\mathit{pred}(p)$, delete the old u' from the PST, and insert the new one. If T_c becomes empty in the process, then we delete c from CT. Clearly, the update operations are correct and take $O(\log n)$ time.

THEOREM 1.1 *Let S be a set of n colored points on the real line. S can be preprocessed into a data structure of size $O(n)$ such that the i distinct colors of the points of S that are contained in any query interval can be reported in $O(\log n + i)$ time and points can be inserted and deleted online in S in $O(\log n)$ time.*

For the static reporting problem, we can dispense with CT and the T_c's and simply use a static form of the PST to answer queries. This provides a simple $O(n)$-space, $O(\log n + i)$-query time alternative to another solution given in [33].

The static counting problem

We store the points of S' in non-decreasing x-order at the leaves of a balanced binary search tree, T, and store at each internal node t of T an array A_t containing the points in t's subtree in non-decreasing y-order. The total space is clearly $O(n \log n)$. To answer a query, we determine $O(\log n)$ canonical nodes v in T such that the query interval $[l, r]$ covers v's range but not the range of v's parent. Using binary search we determine in each canonical node's array the highest array position containing an entry less than l (and thus the number of points in that node's subtree that lie in q') and add up the positions thus found at all canonical nodes. The correctness of this algorithm follows from Lemma 1.1. The total query time is $O(\log^2 n)$.

We can reduce the query time to $O(\log n)$ as follows: At each node t we create a linked list, B_t, which contains the same elements as A_t and maintain a pointer from each entry of B_t to the same entry in A_t. We then apply the technique of fractional cascading [16] to the B-lists, so that after an initial $O(\log n)$-time binary search in the B-list of the root, the correct positions in the B-lists of all the canonical nodes can be found directly in $O(\log n)$ total time. (To facilitate binary search in the root's B-list, we build a balanced search tree on it after the fractional cascading step.) Once the position in a B-list is known, the appropriate position in the corresponding A-array can be found in $O(1)$ time.

It is possible to reduce the space slightly (to $O(n \log n / \log\log n)$) at the expense of a larger query time ($O(\log^2 n / \log\log n)$), by partitioning the points of S' recursively into horizontal strips of a certain size and doing binary search, augmented with fractional cascading, within the strips. Details can be found in [27].

THEOREM 1.2 *Let S be a set of n colored points on the real line. S can be preprocessed into a data structure of size $O(n \log n)$ (resp., $O(n \log n / \log\log n)$) such that the number of distinctly-colored points of S that are contained in any query interval can be determined in $O(\log n)$ (resp., $O(\log^2 n / \log\log n)$) time.*

The dynamic counting problem

We store the points of S' using the same basic two-level tree structure as in the first solution for the static counting problem. However, T is now a $BB(\alpha)$ tree [54] and the auxiliary structure, $D(t)$, at each node t of T is a balanced binary search tree where the points are stored at the leaves in left to right order by non-decreasing y-coordinate. To facilitate the querying, each node v of $D(t)$ stores a count of the points in its subtree. Given a real number, l, we can determine in $O(\log n)$ time the number of points in $D(t)$ that have y-coordinate less than l by searching for l in $D(t)$ and adding up the count for each node of

$D(t)$ that is not on the search path but is the left child of a node on the path. It should be clear that $D(t)$ can be maintained in $O(\log n)$ time under updates.

In addition to the two-level structure, we also use the trees T_c and the tree CT, described previously, to maintain the correspondence between S and S'. We omit further discussion about the maintenance of these trees.

Queries are answered as in the static case, except that at each auxiliary structure we use the above-mentioned method to determine the number of points with y-coordinate less than l. Thus the query time is $O(\log^2 n)$. (We cannot use fractional cascading here.)

Insertion/deletion of a point is done using the worst-case updating strategy for $BB(\alpha)$ trees, and take $O(\log^2 n)$ time.

THEOREM 1.3 *Let S be a set of n colored points on the real line. S can be preprocessed into a data structure of size $O(n \log n)$ such that the number of distinctly-colored points of S that are contained in any query interval can be determined in $O(\log^2 n)$ time and points can be inserted and deleted online in S in $O(\log^2 n)$ worst-case time.*

The static type-2 problem

We wish to preprocess a set S of n colored points on the x-axis, so that for each color intersected by a query interval $q = [l, r]$, the number of points of that color in q can be reported efficiently. The solution for this problem originally proposed in [27] takes $O(n \log n)$ space and supports queries in $O(\log n + i)$ time. The space bound was improved to $O(n)$ in [12], as follows.

The solution consists of two priority search trees, PST_1 and PST_2. PST_1 is similar to the priority search tree built on S' in the solution for the dynamic reporting problem, with an additional count stored at each node. Let $p' = (p, \mathit{pred}(p))$ be the point that is stored at a node in PST_1 and c the color of p. Then at this node, we store an additional number $t_1(p')$, which is the number of points of color c to the right of p.

PST_2 is based on a transformation that is symmetric to the one used for PST_1. For each color c, we sort the distinct points of that color by increasing x-coordinate. For each point p of color c, let $\mathit{next}(p)$ be its successor in the sorted order; for the rightmost point of color c, we take the successor to be the point $+\infty$. We then map p to the point $p'' = (p, \mathit{next}(p))$ in the plane and associate with it the color c. Let S'' be the resulting set of points. We build PST_2 on S'', with an additional count stored at each node. Let $p'' = (p, \mathit{next}(p))$ be the point that is stored at a node in PST_2 and c the color of p. Then at this node, we store an additional number $t_2(p'')$, which is the number of points of color c to the right of $\mathit{next}(p)$.

We also maintain an auxiliary array A of size n. Given a query $q = [l, r]$, we query PST_1 with $q' = [l, r] \times (-\infty, l)$ and for each color c found, we set $A[c] = t_1(p')$, where p' is the point stored at the node where we found c. Then we query PST_2 with $q'' = [l, r] \times (r, +\infty)$ and for each color c found, we report c and $A[c] - t_2(p'')$, where p'' is the point stored at the node where we found c. This works because the queries on PST_1 and PST_2 effectively find the leftmost and rightmost points of color c in $q = [l, r]$ (cf. proof of Lemma 1.1). Thus, $A[c] - t_2(p'')$ gives the number of points of color c in q.

THEOREM 1.4 *A set S of n colored points on the real line can be preprocessed into a data structure of size $O(n)$ such that for any query interval, a type-2 counting query can be answered in $O(\log n + i)$ time, where i is the output size.*

Finally, we note that Theorem 1.1 combined with the notion of persistence (which is discussed in detail Section 1.3.3) yields an efficient solution to the *generalized grounded range reporting* problem in $\mathbb{R}^2$. In this problem, we wish to preprocess a set of n colored points in $\mathbb{R}^2$ so that the distinct colors of the points lying in any 3-sided axes-parallel query rectangle can be reported efficiently. The solution uses $O(n \log n)$ and $O(\log n + i)$ query time. As shown in [50], the space bound can be further improved to $O(n)$ by using persistence in conjunction with a transformation that maps the colored points in $\mathbb{R}^1$ to colored points in $\mathbb{R}^2$ such that the y-coordinates are integers in the range $[0 : \lceil \log n \rceil]$. (This transformation is different from the one underlying Lemma 1.1.) We refer the reader to [50] for details. In Section 1.3.6, we describe another solution to this problem which has the same bounds but is based on a different technique.

1.3.2 A sparsification-based approach

The idea behind this approach is to generate from the given set, S, of colored objects a colored set, S'—possibly consisting of different objects than those in S—such that a query object q intersects an object in S if and only if it intersects an object in S'. Moreover, if q intersects objects in S' then it intersects at most a constant number of them. This allows us to use a solution to a standard problem on S' to solve the generalized reporting problem on S. (In the case of a generalized counting problem, the requirement is more stringent: exactly one object in S' must be intersected.) We illustrate this method with the generalized halfspace range searching problem in $\mathbb{R}^d$, $d = 2, 3$.

Generalized halfspace range searching in $\mathbb{R}^2$ and $\mathbb{R}^3$

Let S be a set of n colored points in $\mathbb{R}^d$, $d = 2, 3$. In the *generalized halfspace range searching* problem we wish to preprocess S so that for any query hyperplane Q, the i distinct colors of the points lying in the closed halfspace Q^- (i.e., below Q) can be reported or counted efficiently. Without loss of generality, we may assume that Q is non-vertical since vertical queries are easy to handle. The approach described here is from [28].

We denote the coordinate directions by $x_1, x_2, \ldots, x_d$. Let $\mathcal{F}$ denote the well-known point-hyperplane duality transform [23]: If $p = (p_1, \ldots, p_d)$ is a point in $\mathbb{R}^d$, then $\mathcal{F}(p)$ is the hyperplane $x_d = p_1x_1 + \cdots + p_{d-1}x_{d-1} - p_d$. If $H : x_d = a_1x_1 + \cdots + a_{d-1}x_{d-1} + a_d$ is a (non-vertical) hyperplane in $\mathbb{R}^d$, 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, 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 S to a set S' of hyperplanes and map Q to the point $q = \mathcal{F}(Q)$, both in $\mathbb{R}^d$. Our problem is now equivalent to: "Report or count the i distinct colors of the hyperplanes lying on or above q, i.e., the hyperplanes that are intersected by the vertical ray r emanating upwards from q."

Let S_c be the set of hyperplanes of color c. For each color c, we compute the *upper envelope* E_c of the hyperplanes in S_c. E_c is the locus of the points of S_c of maximum x_d-coordinate for each point on the plane $x_d = 0$. E_c is a d-dimensional convex polytope which is unbounded in the positive x_d-direction. 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 $\mathbb{R}^2$, E_c is an unbounded convex chain and its facets are line segments; in $\mathbb{R}^3$, E_c is an unbounded convex polytope whose facets are convex polygons.

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 $\mathbb{R}^2$ and an edge or a vertex in $\mathbb{R}^3$. (This assumption can be removed; details can be found in [28].) Then, by definition of the upper envelope, it follows that (i) r intersects a c-colored hyperplane if and only if 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. We will show how to solve efficiently this ray-envelope intersection problem in $\mathbb{R}^2$ and in $\mathbb{R}^3$. This approach does not give an efficient solution to the generalized halfspace searching problem in $\mathbb{R}^d$ for $d > 3$; for this case, we will give a different solution in Section 1.3.4.

To solve the ray–envelope intersection problem in $\mathbb{R}^2$, 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 $v \in T$ if and only if $I(v) \neq \emptyset$ and s crosses $Strip(v)$ but not $Strip(parent(v))$. Let $\mathcal{E}(v)$ be the set of segments allocated to v. Within $Strip(v)$, the segments of $\mathcal{E}(v)$ can be viewed as lines since they cross $Strip(v)$ completely. Let $\mathcal{E}'(v)$ be the set of points dual to these lines. We store $\mathcal{E}'(v)$ in an instance $D(v)$ of the standard halfplane reporting (resp., counting) structure for $\mathbb{R}^2$ given in [17] (resp., [40]). 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 $\mathbb{R}^2$, a halfplane query at v using the query $\mathcal{F}(q)^- = Q^-$, which we do using $D(v)$. 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 1.5 *A set S of n colored points in $\mathbb{R}^2$ can be stored in a data structure of size $O(n \log n)$ so that the i distinct colors of the points contained 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 $D(v)$ 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, [19, page 675].) Thus, the overall query time is $O(\sum_{j=0}^{O(\log n)} (n/2^j)^{1/2})$, which is $O(n^{1/2})$. ■

The (standard) problem of reporting the segments in $\mathbb{R}^2$ that are intersected by a vertical query ray has been revisited recently in [4], in the context of solving a different problem defined on so-called "uncertain points". For a set of n segments, the solution in [4] uses $O(n)$ space and has a query time of $O(\log n + k)$, where k is the number of reported segments. The approach is based on using a combination of a segment tree and an interval tree together with some other ideas; we refer the reader to [4] for details. By the preceding discussion, this result implies an $O(n)$-space and $O(\log n + i)$-query time solution to the generalized halfspace range reporting problem in $\mathbb{R}^2$.

In $\mathbb{R}^3$, the approach is similar, but somewhat more complex. Our goal is to solve the ray–envelope intersection problem in $\mathbb{R}^3$. As shown in [28], this problem can be reduced to certain standard halfspace range queries in $\mathbb{R}^3$ on a set of triangles (obtained by triangulating the E_c's.) This problem can be solved by building a segment tree on the x-spans of the triangles projected to the xy-plane and augmenting each node of this tree with a data structure based on partition trees [39] or cutting trees [38] to answer the halfplane queries. Details may be found in [28].

THEOREM 1.6 *The reporting version of the generalized halfspace range searching problem for a set of n colored points in $\mathbb{R}^3$ 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.*

Additional examples of the sparsification-based approach may be found in [33]. (An example also appears in the next section, enroute to a persistence-based solution of a generalized problem.)

1.3.3 A persistence-based approach

Roughly speaking, we use persistence as follows: To solve a given generalized problem we first identify a different, but simpler, generalized problem and devise a data structure for it that also supports updates (usually just insertions). We then make this structure partially persistent [22] and query this persistent structure appropriately to solve the original problem.

We illustrate this approach for *generalized 3-dimensional range searching*, where we are required to preprocess a set, S, of n colored points in $\mathbb{R}^3$ so that for any query box $q = [a, b] \times [c, d] \times [e, f]$ the i distinct colors of the points inside q can be reported efficiently. We first show how to build a semi-dynamic (i.e., insertions-only) data structure for the generalized versions of the quadrant searching and 2-dimensional range searching problems. These two structures will be the building blocks of our solution for the 3-dimensional problem.

Generalized semi-dynamic quadrant searching

Let S be a set of n colored points in the plane. For any point $q = (a, b)$, the *northeast quadrant* of q, denoted by $NE(q)$, is the set of all points (x, y) in the plane such that $x \geq a$ and $y \geq b$. We show how to preprocess S so that for any query point q, the distinct colors of the points of S contained in $NE(q)$ can be reported, and how points can be inserted into S. This is the *generalized 2-dimensional quadrant range searching* problem supporting insertions. The data structure uses $O(n)$ space, has a query time of $O(\log^2 n + i)$, and an amortized insertion time of $O(\log n)$. This solution is based on the sparsification approach described previously.

For each color c, we determine the c-maximal points. (A point p is called *c-maximal* if it has color c and there are no points of color c in p's northeast quadrant.) We discard all points of color c that are not c-maximal. In the resulting set, let the predecessor, *pred*(p), of a c-colored point p be the c-colored point that lies immediately to the left of p. (For the leftmost point of color c, the predecessor is the point $(-\infty, \infty)$.) With each point $p = (a, b)$, we associate the horizontal segment with endpoints (a', b) and (a, b), where a' is the x-coordinate of $pred(p)$. This segment gets the same color as p. Let S_c be the set of such segments of color c. The data structure consists of two parts, as follows.

The first part is a structure $\mathcal{T}$ storing the segments in the sets S_c, where c runs over all colors. $\mathcal{T}$ supports the following query: given a point q in the plane, report the segments that are intersected by the upward-vertical ray starting at q. Moreover, it allows segments to be inserted and deleted. We implement $\mathcal{T}$ as the structure given in [18]. This structure uses $O(n)$ space, supports insertions and deletions in $O(\log n)$ time, and has a query time of $O(\log^2 n + l)$, where l is the number of segments intersected.

The second part is a balanced search tree CT, storing all colors. For each color c, we maintain a balanced search tree, T_c, storing the segments of S_c by increasing y-coordinate. This structure allows us to dynamically maintain S_c when a new c-colored point p is inserted. The general approach (omitting some special cases; see [27]) is as follows: By doing a binary search in T_c we can determine whether or not p is c-maximal in the current set of c-maximal points, i.e., the set of right endpoints of the segments of S_c. If p is not c-maximal, then we simply discard it. If p is c-maximal, then let $s_1, \ldots, s_k$ be the segments of S_c whose left endpoints are in the southwest quadrant of p. We do the following: (i) delete $s_2, \ldots, s_k$ from T_c; (ii) insert into T_c the horizontal segment which starts at p and extends leftwards upto the x-coordinate of the left endpoint of s_k; and (iii) truncate the segment s_1 by keeping only the part of it that extends leftwards upto p's x-coordinate. The entire operation can be done in $O(\log n + k)$ time.

Let us now consider how to answer a quadrant query, $NE(q)$, and how to insert a point into S. To answer $NE(q)$, we query $\mathcal{T}$ with the upward-vertical ray from q and report the colors of the segments intersected. The correctness of this algorithm follows from the easily proved facts that (i) a c-colored point lies in $NE(q)$ if and only if a c-maximal point lies in $NE(q)$ and (ii) if a c-maximal point is in $NE(q)$, then the upward-vertical ray from q must intersect a segment of S_c. The correctness of $\mathcal{T}$ guarantees that only the segments intersected by this ray are reported. Since the query can intersect at most two segments in any S_c, we have $l \leq 2i$, and so the query time is $O(\log^2 n + i)$.

Let p be a c-colored point that is to be inserted into S. If c is not in CT, then we insert it into CT and insert the horizontal, leftward-directed ray emanating from p into a new structure T_c. If c is present already, then we update T_c as just described. In both cases, we then perform the same updates on $\mathcal{T}$. Hence, an insertion takes $O((k+1)\log n)$ time.

What is the total time for n insertions into an initially empty set S? For each insertion, we can charge the $O(\log n)$ time to delete a segment s_i, $2 \leq i \leq k$, to s_i itself. Notice that none of these segments will reappear. Thus each segment is charged at most once. Moreover, each of these segments has some previously inserted point as a right endpoint. It follows that the number of segments existing over the entire sequence of insertions is $O(n)$ and so the total charge to them is $O(n \log n)$. The rest of the cost for each insertion ($O(\log n)$ for the binary search plus $O(1)$ for steps (ii) and (iii)) we charge to p itself. Since any p is charged in this mode only once, the total charge incurred in this mode by all the inserted points is $O(n \log n)$. Thus the time for n insertions is $O(n \log n)$, which implies an amortized insertion time of $O(\log n)$.

LEMMA 1.2 Let S be a set of n colored points in the plane. There exists a data structure of size $O(n)$ such that for any query point q, we can report the i distinct colors of the points that are contained in the northeast quadrant of q in $O(\log^2 n + i)$ time. Moreover, if we do n insertions into an initially-empty set then the amortized insertion time is $O(\log n)$.

Generalized semidynamic 2-dimensional range searching

Our goal here is to preprocess a set S of n colored points in the plane so that for any axes-parallel query rectangle $q = [a, b] \times [c, d]$, we can report efficiently the distinct colors of the points in q and moreover perform insertions in S. This is the *generalized 2-dimensional range reporting* problem supporting insertions.

Our solution is based on the quadrant reporting structure of Lemma 1.2. We first show how to solve the problem for query rectangles $q' = [a, b] \times [c, \infty)$. We store the points of S in sorted order by x-coordinate at the leaves of a $BB(\alpha)$ tree T'. At each internal node v, we store an instance of the structure of Lemma 1.2 for *NE*-queries (resp., NW-queries) built on the points in v's left (resp., right) subtree. Let $X(v)$ denote the average of the x-coordinate in the rightmost leaf in v's left subtree and the x-coordinate in the leftmost leaf of v's right subtree; for a leaf v, we take $X(v)$ to be the x-coordinate of the point stored at v.

To answer a query q', we do a binary search down T', using $[a, b]$, until either the search runs off T' or a (highest) node v is reached such that $[a, b]$ intersects $X(v)$. In the former case, we stop. In the latter case, if v is a leaf, then if v's point is in q' we report its color. If v is a non-leaf, then we query the structures at v using the *NE*-quadrant and the NW-quadrant derived from q' (i.e., the quadrants with corners at (a, c) and (b, c), respectively), and then combine the answers. Updates on T' are performed using the amortized-case updating strategy for $BB(\alpha)$ trees [54]. The correctness of the method should be clear. The space and query time bounds follow from Lemma 1.2. Since the amortized insertion time of the quadrant searching structure is $O(\log n)$, the insertion in the $BB(\alpha)$ tree takes amortized time $O(\log^2 n)$ [54].

To solve the problem for general query rectangles $q = [a, b] \times [c, d]$, we use the above approach again, except that we store the points in the tree by sorted y-coordinates. At each internal node v, we store an instance of the data structure above to answer queries of the form $[a, b] \times [c, \infty)$ (resp., $[a, b] \times (-\infty, d]$) on the points in v's left (resp., right) subtree. The query strategy is similar to the previous one, except that we use the interval $[c, d]$ to search in the tree. The query time is as before, while the space and update times increase by a logarithmic factor.

LEMMA 1.3 Let S be a set of n colored points in the plane. There exists a data structure of size $O(n \log^2 n)$ such that for any query rectangle $[a, b] \times [c, d]$, we can report the i distinct colors of the points that are contained in it in $O(\log^2 n + i)$ time. Moreover, points can be inserted into this data structure in $O(\log^3 n)$ amortized time.

Generalized 3-dimensional range searching

The semi-dynamic structure of Lemma 1.3 coupled with persistence allows us to go up one dimension and solve the original problem of interest: Preprocess a set S of n colored points in $\mathbb{R}^3$ so that for any query box $q = [a, b] \times [c, d] \times [e, f]$ the i distinct colors of the points inside q can be reported efficiently.

First consider queries of the form $q' = [a, b] \times [c, d] \times [e, \infty)$. We sort the points of S by non-increasing z-coordinates, and insert them in this order into a partially persistent version of the structure of Lemma 1.3, taking only the first two coordinates into account. To answer q', we access the version corresponding to the smallest z-coordinate greater than or equal to e and query it with $[a, b] \times [c, d]$.

To see that the query algorithm is correct, observe that the version accessed contains the projections on the xy-plane of exactly those points of S whose z-coordinate is at least

e. Lemma 1.3 then guarantees that among these only the distinct colors of the ones in $[a, b] \times [c, d]$ are reported. These are precisely the distinct colors of the points contained in $[a, b] \times [c, d] \times [e, \infty)$. The query time follows from Lemma 1.3. To analyze the space requirement, we note that the structure of Lemma 1.3 satisfies the conditions given in [22]. Specifically, it is a pointer-based structure, where each node is pointed to by only $O(1)$ other nodes. As shown in [22], any modification made by a persistent update operation on such a structure adds only $O(1)$ amortized space to the resulting persistent structure. By Lemma 1.3, the total time for creating the persistent structure, via insertions, is $O(n \log^3 n)$. This implies the same bound for the number of modifications in the structure, so the total space is $O(n \log^3 n)$.

To solve the problem for general query boxes $q = [a, b] \times [c, d] \times [e, f]$, we follow an approach similar to that described for the 2-dimensional case: We store the points in a balanced binary search tree, sorted by z-coordinates. We associate with each internal node v in the tree the auxiliary structure described above for answering queries of the form $[a, b] \times [c, d] \times [e, \infty)$ (resp., $[a, b] \times [c, d] \times (-\infty, f]$) on the points in v's left (resp., right) subtree. (Note that since we do not need to do updates here the tree need not be a $BB(\alpha)$ tree.) Queries are done by searching down the tree using the interval $[e, f]$. The query time is as before, but the space increases by a logarithmic factor.

THEOREM 1.7 *Let S be a set of n colored points in 3-space. S can be stored in a data structure of size $O(n \log^4 n)$ such that for any query box $[a, b] \times [c, d] \times [e, f]$, we can report the i distinct colors of the points that are contained in it in $O(\log^2 n + i)$ time.*

Additional applications of the persistence-based approach to generalized intersection problems can be found in [27, 28, 30, 50].

1.3.4 A general approach for reporting problems

We describe a general method from [30] for solving any generalized reporting problem given a data structure for a "related" standard decision problem.

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

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

THEOREM 1.8 *Assume that a set of n geometric objects can be stored in a data structure of size $M(n)$ such that it can be decided in $f(n)$ time whether or not a query object intersects any of the n objects. Assume that $M(n)/n$ and $f(n)$ are non-decreasing functions for non-negative values of n. Then a set S of n colored geometric objects can be preprocessed into a data structure of size $O(M(n) \log n)$ such that the i distinct colors of the objects in S that are intersected by a query object q can be reported in time $O(f(n) + i \cdot f(n) \log n)$.*

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

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

If $v_1, v_2, \ldots, v_r$ are the nodes at any level, then the total space used by CT at that level is $\sum_{i=1}^{r} M(|S(v_i)|) = \sum_{i=1}^{r} |S(v_i)| \cdot (M(|S(v_i)|)/|S(v_i)|) \leq \sum_{i=1}^{r} |S(v_i)| \cdot (M(n)/n) = M(n)$, since $\sum_{i=1}^{r} |S(v_i)| = n$ and since $|S(v_i)| \leq n$ implies that $M(|S(v_i)|)/|S(v_i)| \leq M(n)/n$. Now since there are $O(\log n)$ levels, the overall space is $O(M(n)\log n)$. The query time can be upper-bounded as follows: If $i = 0$, then the query on $DEC(\text{root})$ returns "false" and we abandon the search at the root itself; in this case, the query time is just $O(f(n))$. Suppose that $i \neq 0$. Call a visited node v *fruitful* if the query on $DEC(v)$ returns "true" and *fruitless* otherwise. Each fruitful node can be charged to some color in its subtree that gets reported. Since the number of times any reported color can be charged is $O(\log n)$ (the height of CT) and since i colors are reported, the number of fruitful nodes is $O(i \log n)$. Since each fruitless node has a fruitful parent and CT is a binary tree, it follows that there are only $O(i \log n)$ fruitless nodes. Hence the number of nodes visited by the search is $O(i \log n)$. At each such node, v, we spend time $f(|S(v)|)$, which is $O(f(n))$ since $|S(v)| \leq n$ and f is non-decreasing. Thus the total time spent in doing queries at the visited nodes is $O(i \cdot f(n) \log n)$. The claimed query time follows. ∎

As an application of this method, consider the generalized halfspace range searching problem in $\mathbb{R}^d$, for any fixed $d \geq 2$. For $d = 2, 3$, we discussed a solution for this problem in Section 1.3.2. For $d > 3$, the problem can be solved by extending (significantly) the ray-envelope intersection algorithm outlined in Section 1.3.2. However, the bounds are not very satisfactory—$O(n^{d\lfloor d/2 \rfloor + \epsilon})$ space and logarithmic query time or near-linear space and superlinear query time. The solution we give below has more desirable bounds.

The colored objects for this problem are points in $\mathbb{R}^d$ and the query is a closed halfspace in $\mathbb{R}^d$. We store the objects in CT, as described previously. The standard decision problem that we need to solve at each node v of CT is "Does a query halfspace contain any point of $S(v)$." The answer to this query is "true" if and only if the query halfspace is non-empty. We take the data structure, $DEC(v)$, for this problem to be the one given in [41]. If $|S_v| = n_v$, then $DEC(v)$ uses $O(n_v^{\lfloor d/2 \rfloor}/(\log n_v)^{\lfloor d/2 \rfloor - \epsilon})$ space and has query time $O(\log n_v)$ [41]. The conditions in Theorem 1.8 hold, so applying it gives the following result.

THEOREM 1.9 *For any fixed $d \geq 2$, a set S of n colored points in $\mathbb{R}^d$ 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.*

Other applications of the general method may be found in [30].

1.3.5 Adding range restrictions

We describe the general technique of [29] that adds a range restriction to a generalized intersection searching problem.

Let PR be a generalized intersection searching problem on a set S of n colored objects and query objects q belonging to a class Q. We denote the answer to a query by $PR(q, S)$. To add a *range restriction*, we associate with each element $p \in S$ a real number k_p. In a range-restricted generalized intersection searching problem, denoted by TPR, a query consists of an element $q \in Q$ and an interval $[l, r]$, and

$$TPR(q, [l, r], S) := PR(q, \{p \in S : l \leq k_p \leq r\}).$$

For example, if PR is the generalized $(d-1)$-dimensional range searching problem, then TPR is the generalized d-dimensional version of this problem, obtained by adding a range restriction to the dth dimension.

Assume that we have a data structure DS that solves PR with $O((\log n)^u + i)$ query time using $O(n^{1+\epsilon})$ space and a data structure TDS that solves TPR for generalized (semi-infinite) queries of the form $TPR(q, [l, \infty), S)$ with $O((\log n)^v + i)$ query time using $O(n^w)$ space. (Here u and v are positive constants, $w > 1$ is a constant, and $\epsilon > 0$ is an arbitrarily small constant.) We will show how to transform DS and TDS into a data structure that solves generalized queries $TPR(q, [l, r], S)$ in $O((\log n)^{\max(u,v,1)} + i)$ time, using $O((n^{1+\epsilon})$ space.

Let $S = \{p_1, p_2, \ldots, p_n\}$, where $k_{p_1} \geq k_{p_2} \geq \ldots \geq k_{p_n}$. Let m be an arbitrary parameter with $1 \leq m \leq n$. We assume for simplicity that n/m is an integer. Let $S_j = \{p_1, p_2, \ldots, p_{jm}\}$ and $S'_j = \{p_{jm+1}, p_{jm+2}, \ldots, p_{(j+1)m}\}$ for $0 \leq j < n/m$.

The transformed data structure consists of the following. For each j with $0 \leq j < n/m$, there is a data structure DS_j (of type DS) storing S_j for solving generalized queries of the form $PR(q, S_j)$, and a data structure TDS_j (of type TDS) storing S'_j for solving generalized queries of the form $TPR(q, [l, \infty), S'_j)$.

To answer a query $TPR(q, [l, \infty), S)$, we do the following. Compute the index j such that $k_{p_{(j+1)m}} < l \leq k_{p_{jm}}$. Solve the query $PR(q, S_j)$ using DS_j, solve the query $TPR(q, [l, \infty), S'_j)$ using TDS_j, and output the union of the colors reported by these two queries. It is easy to see that the query algorithm is correct. The following lemma gives the complexity of the transformed data structure.

LEMMA 1.4 The transformed data structure uses $O(n^{2+\epsilon}/m + n\, m^{w-1})$ space and can be used to answer generalized queries $TPR(q, [l, \infty), S)$ in $O((\log n)^{\max(u,v,1)} + i)$ time.

THEOREM 1.10 *Let S, DS and TDS be as above. There exists a data structure of size $O(n^{1+\epsilon})$ that solves generalized queries $TPR(q, [l, r], S)$ in $O((\log n)^{\max(u,v,1)} + i)$ time.*

Proof We will use Lemma 1.4 to establish the claimed bounds for answering generalized queries $TPR(q, [l, \infty), S)$. The result for queries $TPR(q, [l, r], S)$ then follows from a technique, based on $BB(\alpha)$ trees, that we used in Section 1.3.3.

If $w > 2$, then we apply Lemma 1.4 with $m = n^{1/w}$. This gives a data structure having size $O(n^2)$ that answers queries $TPR(q, [l, \infty), S)$ in $O((\log n)^{\max(u,v,1)} + i)$ time. Hence, we may assume that $w = 2$.

By applying Lemma 1.4 repeatedly, we obtain, for each integer constant $a \geq 1$, a data structure of size $O(n^{1+\epsilon+1/a})$ that answers queries $TPR(q, [l, \infty), S)$ in $O((\log n)^{\max(u,v,1)} +$

i) time. This claim follows by induction on a; in the inductive step from a to $a+1$, we apply Lemma 1.4 with $m = n^{a/(a+1)}$. ∎

Using Theorem 1.10, we can solve efficiently, for instance, the generalized orthogonal range searching problem in $\mathbb{R}^d$. (Examples of other problems solvable via this method may be found in [29].)

THEOREM 1.11 *Let S be a set of n colored points in $\mathbb{R}^d$, where $d \geq 1$ is a constant. There exists a data structure of size $O(n^{1+\epsilon})$ such that for any query box in $\mathbb{R}^d$, we can report the i distinct colors of the points that are contained in it in $O(\log n + i)$ time.*

Proof The proof is by induction on d. For $d = 1$, the claim follows from Theorem 1.1. Let $d \geq 2$, and let *DS* be a data structure of size $O(n^{1+\epsilon})$ that answers generalized $(d-1)$-dimensional range queries in $O(\log n + i)$ time. Observe that for the generalized d-dimensional range searching problem, there are only polynomially many distinct semi-infinite queries. Hence, there exists a data structure *TDS* of polynomial size that answers generalized d-dimensional semi-infinite range queries in $O(\log n + i)$ time. Applying Theorem 1.10 to *DS* and *TDS* proves the claim. ∎

1.3.6 Exploiting the output size

In this approach, the design of the data structure and the query algorithm is based on the (unknown) size of the output (i.e., i). It involves building and querying two structures, one to handle "large" output size and the other to handle "small" output size. The definition of "large" and "small" depends on the problem at hand and will become clear in the discussion below.

Suppose that we have designed a low-space data structure, $\mathcal{D}_L$, to handle the case where the output size is large; let the query time of this be $O(f(n)+i)$. Then the crucial observation is that if i is $\Omega(f(n))$ (i.e., i is large), then the query time is $O(i)$, which is the best one can hope for since it takes this much time to merely output the query results. However, if i is $O(f(n))$ (i.e., i is small), then the query time is $O(f(n))$ which is undesirable if $f(n)$ is large. The challenge now is to design a separate data structure, $\mathcal{D}_S$, that can handle efficiently the case where the output size is small, by taking advantage of this very fact. Intuitively, this step involves "precomputing" and storing the answers to certain carefully chosen queries; the space used for this is kept small by exploiting the fact that the output size is small. Note that an additional challenge is that one does not know *a priori* whether i is small or large for a given query instance. Therefore, the overall query algorithm proceeds as follows: First $\mathcal{D}_S$ is queried with the given query object. If this succeeds in returning all of the query results (i.e., the output size is small) then the query algorithm terminates. Otherwise, the query on $\mathcal{D}_S$ aborts and the algorithm proceeds to reissue the query on $\mathcal{D}_L$ to compute the query result.

We illustrate this approach by presenting an optimal algorithm for generalized grounded range reporting in $\mathbb{R}^2$, where the input is a set of n colored points and the query is a rectangle $q = [x_l, x_r] \times [y_0, \infty)$. This algorithm uses $O(n)$ space and answers queries in $O(\log n + i)$ time. The following description is based on ideas from [46, 47], where the problem is solved in external memory.

For $\mathcal{D}_L$, we use a data structure presented in [33] which, for a set of n colored points in $\mathbb{R}^2$, occupies $O(n)$ space and answers a generalized grounded range query in $O(\log^2 n + i)$ time. Thus, $f(n) = \log^2 n$ and we take the output size to be large if $i \geq \log^2 n$. We build an instance of $\mathcal{D}_L$ on the given set S.

We design the structure $\mathcal{D}_S$ as follows: We sort the points of S in non-decreasing order of their x-coordinates and partition them into groups consisting of $\log^3 n$ consecutive points each. For each group we build an instance of $\mathcal{D}_L$. Next, we build a balanced binary search tree, $\mathcal{T}$, based on the left-to-right ordering of the groups and associate a leaf with each group. Let v be a proper ancestor of a leaf u and let $\Pi(u,v)$ be the path from u to v (excluding u and v). Let $S_l(u,v)$ (resp., $S_r(u,v)$) be the union of the sets of points in the subtrees rooted at nodes that are left (resp., right) children of nodes on $\Pi(u,v)$ but not themselves on the path. For each distinct color in $S_l(u,v)$, we select from among the points of that color in $S_l(u,v)$ the one with the highest y-coordinate. Let $S'_l(u,v)$ be the set of such points selected. Let $S''_l(u,v)$ be the subset of $S'_l(u,v)$ consisting of the $\log^2 n$ points with largest y-coordinate (ties broken arbitrarily). If fewer than $\log^2 n$ points are in $S'_l(u,v)$, then we include all of them in $S''_l(u,v)$. A symmetric discussion for $S_r(u,v)$ yields a set $S''_r(u,v)$. For each pair (u,v), we store $S''_l(u,v)$ and $S''_r(u,v)$ in linked lists, in non-increasing order of the y-coordinates of their points (ties broken arbitrarily). The number of (u,v)-pairs stored is $O((n/\log^3 n) \times (\log n)) = O(n/\log^2 n)$, so the space occupied by all sets $S''_l(u,v)$ and $S''_r(u,v)$ is $O(n)$. The space occupied by $\mathcal{T}$ and the $\mathcal{D}_L$ structures at all leaves is also $O(n)$.

To answer a query $q = [x_l, x_r] \times [y_0, \infty)$, we first determine the leaf u_l (resp., u_r) of $\mathcal{T}$ whose range of x-coordinates contains x_l (resp., x_r). If $u_l = u_r$, then we query the $\mathcal{D}_L$ structure of that leaf and stop. Otherwise, we find the lowest common ancestor, v, of u_l and u_r and do the following:

- First, we report the distinct colors of the points in the groups associated with u_l and u_r that lie in q. This is done by querying the $\mathcal{D}_L$ structure of u_l (resp., u_r) with $[x_l, \infty) \times [y_0, \infty)$ (resp., $(-\infty, x_r] \times [y_0, \infty)$).
- Next, we scan the list for $S''_r(u_l,v)$ and report the colors in it until either (a) we find a point with y-coordinate less than y_0, or (b) all the points in the list have been scanned. If case (a) holds, then the distinct colors of the points of $S_r(u_l,v)$ that lie in q have been reported. If case (b) holds, then we check the size of $S''_r(u_l,v)$. If $|S''_r(u_l,v)| < \log^2 n$, then again the distinct colors of the points of $S_r(u_l,v)$ that lie in q have been reported. (Recall that, by construction, if $S_r(u_l,v)$ has fewer than $\log^2 n$ distinctly-colored points, then all of them are included in $S''_r(u_l,v)$.) However, if $|S''_r(u_l,v)| \geq \log^2 n$, then we conclude that i is $\Omega(\log^2 n)$. Similarly, we also scan the list for $S''_l(u_r,v)$ and either report the distinct colors of the points of $S_l(u_r,v)$ that lie in q or conclude that i is $\Omega(\log^2 n)$. If we conclude i is $\Omega(\log^2 n)$ at least once in the above process, then we discard the reported colors and proceed to query the structure $\mathcal{D}_L$ built on the entire set S with q.

The time to find u_l, u_r, and v is $O(\log n)$. If $u_l = u_r$, then querying the $\mathcal{D}_L$ structure at that leaf node takes $O((\log(\log^3 n))^2 + i) = O(\log n + i)$ time. If $u_l \neq u_r$, then, as above, querying the $\mathcal{D}_L$ structure at the two leaves takes $O(\log n + i)$ time. The time to scan $S''_r(u_l,v)$ and $S''_l(u_r,v)$ is $O(i)$. Finally, the time to query the $\mathcal{D}_L$ structure for S is $O(\log^2 n + i) = O(i)$, since at this point we know that i is $\Omega(\log^2 n)$. Therefore, the overall query time is $O(\log n + i)$.

THEOREM 1.12 *Let S be a set of n colored points in $\mathbb{R}^2$. S can be preprocessed into a data structure of size $O(n)$ such that the i distinct colors of the points of S lying in any grounded query rectangle can be reported in $O(\log n + i)$ time.*

We remark that the idea of exploiting the output size to gain efficiency can be traced back to [15], where it was used within the framework of filtering search. It has manifested itself in various forms in several subsequent papers on standard and generalized range search, including [1, 2, 3, 4, 10, 37, 46, 47, 49, 51] among others.

1.3.7 A reverse transformation

In Section 1.3.1, we discussed how the generalized range counting problem in $\mathbb{R}^1$ can be transformed to a standard range counting problem in $\mathbb{R}^2$ (with a grounded query rectangle). In this section, we show a reverse transformation which maps a standard range counting problem in $\mathbb{R}^2$ to a generalized range counting problem in $\mathbb{R}^1$. The approach, which is based on [37], shows that the two problems (i.e., generalized range counting in $\mathbb{R}^1$ and standard range counting in $\mathbb{R}^2$) are in fact equivalent and it also yields a lower bound for the former problem based on a known lower bound for the latter (in the word-RAM model). This reverse transformation is not a solution technique for generalized problems *per se*, but it does reveal an interesting connection between a generalized problem and a standard problem and prompts the question of whether other pairs of generalized and standard problems might share a similar connection.

We begin by noting that a standard range counting query in $\mathbb{R}^2$ with a four-sided (i.e., non-grounded) query rectangle can be answered by doing four standard quadrant counting queries, where each quadrant is defined by one of the vertices of the rectangle, and then adding and subtracting the counts returned using the principle of inclusion/exclusion. Therefore, in what follows, it suffices to focus on standard quadrant counting queries. Specifically, we wish to preprocess a set S of n points in $\mathbb{R}^2$ so that given any query quadrant $q = (-\infty, a] \times (-\infty, b]$, we can count efficiently the number of points of S that are in q. We assume, without loss of generality, that all coordinates are positive.

We map each point $p = (x(p), y(p)) \in S$ to two points in $\mathbb{R}^1$, one with coordinate $-x(p)$ and the other with coordinate $y(p)$. We give each of these points the color p (any unique identifier associated with p). Let S_1 be the set of these newly constructed $2n$ points in $\mathbb{R}^1$. Next we make a copy, S_2, of S_1 and recolor the points so that all of them have distinct colors. For each of S_1 and S_2, we build a data structure for answering generalized range counting queries in $\mathbb{R}^1$. (Note that the structure on S_2 is actually a standard structure since all colors in S_2 are distinct, but it is convenient for our purposes to view it as a generalized structure.)

Given q, we query the two data structures with $q' = [-a, b]$ to obtain two integers t_1 and t_2, where t_1 is the the number of distinct colors among the points of S_1 in q' and t_2 is the number of distinct colors among the points of S_2 in q' (hence also the number of points of S_2 in q', since the points in S_2 have distinct colors). We report $t_2 - t_1$ as the answer to the query q on S, i.e., the number of points of S that are in q.

The correctness of this can be seen as follows: If p is in q, then we have $x(p) \leq a$ and $y(p) \leq b$, i.e., $-a \leq -x(p)$ and $y(p) \leq b$, i.e., $-a \leq -x(p) < b$ and $-a < y(p) \leq b$ (since all coordinates are positive). Hence the points $-x(p)$ and $y(p)$ are both in $q' = [-a, b]$. Thus, this pair contributes 2 to t_2 and 1 to t_1, and it follows that $t_2 - t_1$ correctly returns the count of the points of S that are in q. On the other hand, if p is not in q, then we have $-a > -x(p)$ or $y(p) > b$ (or both). In this case, the pair contributes the same amount to both t_2 and t_1 (1, if exactly one inequality holds, and 0 if both hold), so $t_2 - t_1$ again returns the correct overall count.

1.4 Conclusion and future directions

We have reviewed research on a class of geometric query-retrieval problems, where the objects to be queried come aggregated in disjoint groups and of interest are questions concerning the intersection of the query object with the groups (rather than with the individual objects). These problems include the well-studied standard intersection problems as a special case and have many applications. We have described several general techniques that have been identified for these problems and have illustrated them with examples.

Some potential directions for future work include: (i) extending the transformation-based approach to higher dimensions; (ii) improving the time bounds for some of the problems discussed here—for instance, can the generalized orthogonal range searching problem in $\mathbb{R}^d$, for $d \geq 4$, be solved with $O(\text{polylog}(n) + i)$ query time and $O(n(\log n)^{O(1)}n)$ space; (iii) developing general dynamization techniques for generalized problems, along the lines of, for instance, [11] for standard problems; (iv) developing efficient solutions to generalized problems where the objects may be in time-dependent motion; (v) identifying other pairs of standard and generalized problems (beyond the pair discussed in Section 1.3.7) that are equivalent; and (vi) implementing and testing experimentally some of the solutions presented here.

1.5 Acknowledgement

Parts of the material in this chapter are drawn from the publications of some of the chapter's authors: References [27, 28, 29], with permission from Elsevier (`http://www.elsevier.com`), and Reference [30], with permission from Taylor & Francis (`http://www.tandf.co.uk`). The work of one of the authors (S. Rahul) is supported in part by a Doctoral Dissertation Fellowship from the Graduate School of the University of Minnesota.

References

[1] P. Afshani, L. Arge, and K.G. Larsen. Higher-dimensional orthogonal range reporting and rectangle stabbing in the pointer machine model. In *Proceedings of the ACM Symposium on Computational Geometry*, pages 323–332, 2012.

[2] P. Afshani and T.M. Chan. Optimal halfspace range reporting in three dimensions. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 180–186, 2009.

[3] P. K. Agarwal, L. Arge, and K. Yi. An optimal dynamic interval stabbing-max data structure? In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 803–812, 2005.

[4] P. K. Agarwal, S. W Cheng, Y. Tao, and K. Yi. Indexing uncertain data. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 137–146, 2009.

[5] P. K. Agarwal, M. de Berg, S. Har-Peled, M. H. Overmars, M. Sharir, and J. Vahrenhold. Reporting intersecting pairs of convex polytopes in two and three dimensions. *Computational Geometry: Theory and Applications*, 23:195–207, 2002.

[6] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, volume 223 of *Contemporary Mathematics*, pages 1–56. American Mathematical Society, Providence, RI, 1999.

[7] P. K. Agarwal, S. Govindarajan, and S. Muthukrishnan. Range searching in categorical data: Colored range searching on grid. In *Proceedings of the 10th European Sym-*

posium on Algorithms, volume 2461 of *Lecture Notes in Computer Science*, pages 17–28, Berlin, 2002. Springer-Verlag.

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

[9] A. Aggarwal and J. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.

[10] A. Agrawal, S. Rahul, Y. Li, and R. Janardan. Range search on tuples of points. *Journal of Discrete Algorithms*, 2014. Published online. DOI: 10.1016/j.jda.2014.10.006.

[11] J. L. Bentley and J. B. Saxe. Decomposable searching problems I: Static-to-dynamic transformations. *Journal of Algorithms*, 1:301–358, 1980.

[12] P. Bozanis, N. Kitsios, C. Makris, and A. Tsakalidis. New upper bounds for generalized intersection searching problems. In *Proceedings of the 22nd International Colloqium on Automata, Languages and Programming*, volume 944 of *Lecture Notes in Computer Science*, pages 464–475, Berlin, 1995. Springer-Verlag.

[13] P. Bozanis, N. Kitsios, C. Makris, and A. Tsakalidis. Red-blue intersection reporting for objects of non-constant size. *The Computer Journal*, 39:541–546, 1996.

[14] P. Bozanis, N. Kitsios, C. Makris, and A. Tsakalidis. New results on intersection query problems. *The Computer Journal*, 40:22–29, 1997.

[15] B. Chazelle. Filtering search: A new approach to query-answering. *SIAM Journal on Computing*, 15(3):703–724, 1986.

[16] B. Chazelle and L. J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1:133–162, 1986.

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

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

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

[20] M. de Berg and H. Haverkort. Significant-presence range queries in categorical data. In *Proceedings of the 8th International Workshop on Algorithms and Data Structures*, pages 462–473. Springer, 2003.

[21] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd edition, 2000.

[22] 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.

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

[24] A. Elmasry, M. He, J. Munro, and P. Nicholson. Dynamic range majority data structures. In *Proceedings of the 22nd International Symposium on Algorithms and Computation*, pages 150–159. Springer, 2011.

[25] P. Ferragina, N. Koudas, S. Muthukrishnan, and D. Srivastava. Two-dimensional substring indexing. In *Proceedings of the 20th ACM Symposium on Principles of Database Systems*, pages 282–288, 2001.

[26] P. Gupta. *Efficient algorithms and data structures for geometric intersec tion problems*. Ph.D. dissertation, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1995.

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

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

[29] P. Gupta, R. Janardan, and M. Smid. A technique for adding range restrictions to generalized searching problems. *Information Processing Letters*, 64:263–269, 1997.

[30] P. Gupta, R. Janardan, and M. Smid. Algorithms for some intersection searching problems involving circular objects. *International Journal of Mathematical Algorithms*, 1:35–52, 1999.

[31] P. Gupta, R. Janardan, and M. Smid. Efficient algorithms for counting and reporting pairwise intersections between convex polygons. *Information Processing Letters*, 69:7–13, 1999.

[32] T. Hagerup. Sorting and searching on the word RAM. In *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science*, pages 366–398. Springer, 1998.

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

[34] H. Kaplan, N. Rubin, M. Sharir, and E. Verbin. Counting colors in boxes. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 785–794, 2007.

[35] H. Kaplan, M. Sharir, and E. Verbin. Colored intersection searching via sparse rectangular matrix multiplication. In *Proceedings of the ACM Symposium on Computational Geometry*, pages 52–60, 2006.

[36] Y. Lai, C. Poon, and B. Shi. Approximate colored range and point enclosure queries. *Journal of Discrete Algorithms*, 6(3):420–432, 2008.

[37] K.G. Larsen and F. van Walderveen. Near-optimal range reporting structures for categorical data. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 265–276, 2013.

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

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

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

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

[42] E. M. McCreight. Priority search trees. *SIAM Journal on Computing*, 14:257–276, 1985.

[43] S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 657–666, 2002.

[44] G. Navarro. Spaces, trees, and colors: The algorithmic landscape of document retrieval on sequences. *ACM Computing Surveys*, 46(4):52, 2014.

[45] G. Navarro and Y. Nekrich. Top-k document retrieval in optimal time and linear space. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1066–1077, 2012.

[46] Y. Nekrich. Space-efficient range reporting for categorical data. In *Proceedings of the 31st symposium on Principles of Database Systems*, pages 113–120, 2012.

[47] Y. Nekrich. Efficient range searching for categorical and plain data. *ACM Transactions on Database Systems*, 39(1):9, 2014.

[48] M. Patil, S. V. Thankachan, R. Shah, Y. Nekrich, and J.S. Vitter. Categorical range

maxima queries. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 266–277, 2014.

[49] S. Rahul. Improved bounds for orthogonal point enclosure query and point location in orthogonal subdivisions in R^3. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2015. To appear.

[50] Q. Shi and J. JáJá. Optimal and near-optimal algorithms for generalized intersection reporting on pointer machines. *Information Processing Letters*, 95(3):382–388, 2005.

[51] Y. Tao. Stabbing horizontal segments with rays. In *Proceedings of the ACM Symposium on Computational Geometry*, pages 313–322, 2012.

[52] M. J. van Kreveld. *New Results on Data Structures in Computational Geometry*. Ph.D. dissertation, Department of Computer Science, Utrecht University, The Netherlands, 1992.

[53] J. S. Vitter. External memory algorithms and data structures: dealing with massive data. *ACM Computing Surveys*, 33:209–271, 2001.

[54] D.E. Willard and G.S. Lueker. Adding range restriction capability to dynamic data structures. *Journal of the ACM*, 32:597–617, 1985.