Near-Optimal Range Reporting Structures for Categorical Data

Kasper Green Larsen* Freek van Walderveen*

Abstract

Range reporting on categorical (or colored) data is a well-studied generalization of the classical range reporting problem in which each of the N input points has an associated color (category). A query then asks to report the set of colors of the points in a given rectangular query range, which may be far smaller than the set of all points in the query range.

We study two-dimensional categorical range reporting in both the word-RAM and I/O-model. For the I/O-model, we present two alternative data structures for three-sided queries. The first answers queries in optimal $O(\lg_B N + K/B)$ I/Os using $O(N \lg^* N)$ space, where K is the number of distinct colors in the output, B is the disk block size, and $\lg^* N$ is the iterated logarithm of N. Our second data structure uses linear space and answers queries in $O(\lg_B N + \lg^{(h)} N + K/B)$ I/Os for any constant integer $h \geq 1$. Here $\lg^{(1)} N = \lg N$ and $\lg^{(h)} N = \lg(\lg^{(h-1)} N)$ when $h > 1$. Both solutions use only comparisons on the coordinates. We also show that the $\lg_B N$ terms in the query costs can be reduced to optimal $\lg \lg_B U$ when the input points lie on a $U \times U$ grid and we allow word-level manipulations of the coordinates. We further reduce the query time to just $O(1)$ if the points are given on an $N \times N$ grid. Both solutions also lead to improved data structures for four-sided queries. For the word-RAM, we obtain optimal data structures for three-sided range reporting, as well as improved upper bounds for four-sided range reporting.

Finally, we show a tight lower bound on one-dimensional categorical range *counting* using an elegant reduction from (standard) two-dimensional range counting.

1 Introduction

Orthogonal range reporting is one of the classic and most fundamental data structure problems in the fields of computational geometry and spatial databases. Given a set of N points in d-dimensional space, the goal is to represent the points in a data structure, such that given an axis-aligned query rectangle, one can efficiently list all points contained therein. Any introductory textbook on computational geometry is sure to contain a motivating example such as: "Give me a list of all employees hired between 2001 and 2004 who earn less than 80.000 per year". By representing each employee as the point (year hired, salary), this query translates into the orthogonal range query specified by the (unbounded) axis-aligned rectangle $[2001 : 2004] \times (-\infty : 80.000]$. Given its key role, this problem has been extensively studied in almost all models of computation, including the word-RAM and the I/O-model. The two-dimensional variant of the problem is particularly well understood, with optimal I/O-model solutions dating back to the 1990s [4].

However, in many natural applications, the standard orthogonal range reporting problem does not suffice. Consider as an example the following generalization of the above query: "Give me the different job positions of those employees hired between 2001 and 2004 who earn less than 80.000 per year". This problem can obviously be solved by augmenting the above points with the job position of the corresponding employee and then scanning through the output list. However, this solution may be very inefficient as the number of employees with the same job position could be huge. In range searching terminology, the above problem is an instance of *categorical range reporting* (a.k.a. *colored range reporting* or *generalized range reporting*). For categorical range reporting, we assume the input consists of a set of N points in d-dimensional space, each assigned a color. The goal is to represent the points in a data structure, such that given an axis-aligned query rectangle, one can efficiently report the set of distinct colors assigned to points contained in the query rectangle. While the categorical range reporting problem has numerous applications (e.g. in document retrieval, IP package routing, gis systems and VLSI layout, see [2, 13, 16]), determining the exact complexity of the problem remains open, even in the two-dimensional case.

In this paper, we present significant improvements for the *three-sided* categorical range reporting problem. More specifically, we provide optimal word-RAM and near-optimal I/O-model data structures for categorical range reporting when the query rectangles are

*MADALGO Center for Massive Data Algorithmics, Department of Computer Science, Aarhus University, Denmark. Email: larsen@cs.au.dk, freek@cs.au.dk.

unbounded in one direction, i.e. they are of the form $[x_0 : x_1] \times (-\infty : y]$. In the literature, three-sided range queries have received perhaps even more attention than the general problem. This is owed mainly to two things: It is the "hardest" range searching problem that still admits linear space solutions with less than polynomial query time, and secondly, almost all solutions for the general problem use data structures for three-sided queries as building blocks. In fact, we also obtain improved word-RAM and I/O-model data structures for the general problem using our three-sided data structures as black-boxes. We finally conclude by establishing a tight word-RAM lower bound for one-dimensional categorical range counting (i.e. counting the number of distinct colors in a query interval).

1.1 Models of Computation. As mentioned, we present new solutions in the word-RAM and I/O-model. We briefly summarize the two models below. We also give a short overview of the pointer machine model, as many previous data structures have been designed in this model.

Word-RAM. In the unit-cost word-RAM model, a data structure consists of a memory divided into words of w bits each. For range reporting problems, we make the standard assumption that the number of bits in a word is proportional to the number of bits needed to represent an input point and to address any input point, i.e. if the input points have integer coordinates from a universe $[U] = \{0, \ldots, U-1\}$, then we assume $w = \Theta(\lg N + \lg U)$. The memory words may represent arbitrary information about the input. When answering a query, a data structure may use indirect addressing and any standard operation on words in constant time, including for instance addition, subtraction, multiplication, division and bitwise operations.

I/O-model. In many real-life applications, the input to a range searching problem is too large to fit in the main memory of a machine. In such settings, the number of accesses to the disk becomes the performance bottleneck. To cope with this, the I/O-model defines a machine to consist of a main memory of limited size M words and an infinite sized disk, partitioned into disk blocks of size B words. Again, a word consists of $\Theta(\lg N + \lg U)$ bits. Computation can only take place on data sitting in main memory. A data structure in this model resides on disk and relevant parts are moved to and from main memory in blocks of B words when needed, which we refer to as an I/O. In this model, we measure the performance of a data structure in the number of I/Os performed, i.e. all computation on data in main memory is free of charge.

Range reporting data structures in the I/O-model are typically designed under the indivisibility assumption, i.e. an input point has to be stored "uncompressed" in a word in order to be reported. Other than that, words can store arbitrary information and a data structure is allowed to perform random accesses to disk blocks. Making the indivisibility assumption typically allows for very high and near-tight lower bounds using the indexability framework of Hellerstein et al. [14]. The I/O-model data structures we present all satisfy the indivisibility assumption.

Pointer Machine. The pointer machine model is a constrained variant of the word-RAM where the memory words of the data structure can only be accessed through pointers. Furthermore, the input points to a range searching problem are assumed *indivisible* and we assume coordinates can only be *compared*. This means that any memory word may store one input point plus a constant number of pointers to other memory words. The main motivating factor for studying the pointer machine is that we can prove polynomially high and often tight lower bounds for range reporting problems using the framework of Chazelle [11]. This stands in sharp contrast to the highest query time lower bound proved for *any* static data structure problem in the word-RAM, a mere $\Omega(\lg N)$, even for linear space data structures [18]. Note that since any memory cell stores a constant number of pointers, one cannot generally hope for a query time below $\Theta(\lg N)$.

1.2 Previous Results. The categorical range reporting problem was first introduced by Janardan and Lopez in 1993 [16]. Their solutions are for the pointer machine model and they presented an optimal $O(N)$ space and $O(\lg N + K)$ query time data structure for one-dimensional categorical range reporting. Here, and throughout the paper, K denotes the number of distinct colors in the output of a query, and space is measured in number of words. For the (four-sided) two-dimensional problem, they presented two different tradeoffs: One data structure using $O(N \lg N)$ space and answering queries in $O(\lg^2 N + K)$ time and one using $O(N \lg^2 N)$ space and answering queries in optimal $O(\lg N + K)$ time.

The next results on the two-dimensional problem were due to Agarwal et al. [2]. They studied the problem in the word-RAM model and assumed the coordinates of the input points lie on a $U \times U$ integer grid. For such inputs, they presented a data structure using $O(N \lg^2 U)$ space and answering queries in $O(\lg \lg U + K)$ time. This query time is optimal by reduction from predecessor search [23]. Agarwal et al. also consider the case where the query is three-sided. For this problem, they present a data structure using $O(N \lg N)$ space

and answering queries in $O(\lg\lg U + K)$ time. Again the query time is optimal.

Subsequently, Shi and JaJa [24] presented a linear space pointer machine data structure that answers three-sided queries in optimal $O(\lg N + K)$ time. By a standard reduction, this also gave an $O(N \lg N)$ space data structure for four-sided queries, answering queries in optimal $O(\lg N + K)$ time. Other variants of categorical range reporting were investigated by Gupta et al. [13] and Bozanis et al. [8, 9], with a main focus on dynamic data structures and other types of query ranges.

Very recently, several results were presented for categorical range reporting in the I/O-model [19, 21]. In [21], Nekrich presents a number of tradeoffs for three-sided queries: His first data structure uses linear space and answers queries in $O((N/B)^\varepsilon + K/B)$ time, where $\varepsilon > 0$ is an arbitrarily small constant. Note that the desired output term in the I/O-model is $O(K/B)$ and not $O(K)$, since writing K output points to disk costs $O(K/B)$ I/Os. For input points with coordinates on an $N \times N$ grid (rank space), he presented two additional data structures for three-sided queries, one answering queries in optimal $O(1 + K/B)$ I/Os using $O(N \lg\lg N \lg\lg\lg N)$ space, and one answering queries in $O(\lg^\varepsilon \lg N + K/B)$ I/Os with $O(N \lg\lg N)$ space, where $\varepsilon > 0$ is an arbitrarily small constant. Finally, he presented a data structure for three-sided queries when the coordinates are on a $U \times U$ grid. This data structure answers queries in $O(\lg\lg_B U + K/B)$ I/Os and uses $O(N \lg\lg N \lg\lg\lg N)$ space. The query time is optimal by reduction from predecessor search [23]. Using these data structures as building blocks, he also obtained an improved data structure for the general problem on a $U \times U$ grid, using $O(N \lg N \lg\lg N \lg\lg\lg N)$ space and answering queries in optimal $O(\lg\lg_B U + K/B)$ I/Os. We note that all the data structures of Nekrich can be implemented using only comparisons, at the cost of increasing the search cost to $O(\lg_B N)$. They can also be implemented in the word-RAM at the cost of removing the B's from the above bounds. His results for input points with coordinates on a $U \times U$ grid thus improve over the previous best word-RAM results of Agarwal et al. [2]

For one-dimensional categorical range counting, Gupta et al. [13] showed a reduction to two-dimensional range counting *without* colors. Thus using the two-dimensional range counting word-RAM data structure of JaJa et al. [15], one can solve the problem in linear space and $O(\lg N/\lg\lg N)$ time. This bound is optimal for two-dimensional range counting [22], but is not known to be optimal for the one-dimensional categorical range counting problem. For two-dimensional categorical range counting, Kaplan et al. [17] showed a connection to matrix multiplication, implying in particular that answering N two-dimensional categorical range counting queries in $O(N^{\omega/2})$ time yields an $O(N^\omega)$ time algorithm for multiplying two $N \times N$ Boolean matrices. For $\omega < 2.373$ this would improve the best known upper bound for Boolean matrix multiplication [25]. Thus solving two-dimensional categorical range counting in polylogarithmic time would be a major breakthrough.

Unconditional Lower Bounds. Since standard range reporting is a special case of categorical range reporting, we can conclude that any I/O-model data structure with $O(f(N,B) + K/B)$ query time for two-dimensional categorical range reporting (four-sided queries) must use $\Omega(N \lg N/\lg f(N,B))$ space [4]. When coordinates can only be compared, the optimal bound on $f(N,B)$ is $O(\lg_B N)$, so the space lower bound is $\Omega(N \lg N/\lg\lg_B N)$. We note that $O(\lg_B N)$ is also the best achievable query time for two- and three-sided queries when coordinates can only be compared. When coordinates are integers on a $U \times U$ grid, where $U = N^{1+\Omega(1)}$, the optimal bound on $f(N,B)$ is $O(\lg\lg_B U)$ for all three variants [23]. Finally, if points are on an $N \times N$ grid, only the four-sided problem has an $\Omega(\lg\lg_B N)$ lower bound [23], whereas two- and three-sided queries can be solved with $f(N,B) = O(1)$.

In the word-RAM, the only lower bounds known are obtained by reduction from predecessor search. This allows us to conclude that for any $N \lg^{O(1)} N$ space, the query time must be $\Omega(\lg\lg U + K)$ for the four-sided problem on points with coordinates on a $U \times U$ grid. For the three-sided problem on a $U \times U$ grid we can similarly conclude that the query time must be $\Omega(\lg\lg U + K)$ for any $N \lg^{O(1)} N$ space, provided that $U = N^{1+\Omega(1)}$. We note that on an $N \times N$ grid, the standard three-sided problem can be solved in $O(1 + K)$ time [3], while the four-sided problem still has an $\Omega(\lg\lg N + K)$ lower bound [23].

Since these word-RAM lower bounds do not specify precisely what space we can achieve in optimal query time, we mention that the best known solution for standard range reporting (four-sided) in the word-RAM answers queries in optimal $O(\lg\lg U + K)$ time using $O(N \lg^\varepsilon N)$ space, where $\varepsilon > 0$ is an arbitrarily small constant [10].

1.3 Our Results.

Our main results are optimal data structures for three-sided categorical range reporting in the word-RAM and near-optimal data structures for the I/O-model. More specifically, we present an optimal $O(N)$ space and $O(\lg\lg U + K)$ query time data structure in the word-RAM when points have coordinates on a $U \times U$ grid. We improve the query time

further to $O(1+K)$ when the points are on an $N \times N$ grid. The best previous linear space data structure in the word-RAM is in fact the pointer machine result with $O(\lg N + K)$ query time. The best previous result with optimal $O(\lg\lg U + K)$ query time is the $O(N \lg\lg N \lg\lg\lg N)$ space data structure of Nekrich.

For the I/O-model, we show that for any integer $h \geq 1$ (not necessarily constant), one can obtain a data structure for three-sided queries that uses $O(Nh)$ space and answers queries in $O(\lg_B N + \lg^{(h)} N + K/B)$ I/Os. Here $\lg^{(h)} N$ is the logarithm iterated h times, i.e. $\lg^{(1)} N = \max\{1, \lg N\}$ and $\lg^{(i)} N = \max\{1, \lg(\lg^{(i-1)} N)\}$. At one extreme of the tradeoff, we get that for any constant integer $h \geq 1$, there exists a linear space data structure answering queries in $O(\lg_B N + \lg^{(h)} N + K/B)$ I/Os. At the other extreme, we get a data structure using $O(N \lg^* N)$ space and answering queries in $O(\lg_B N + K/B)$ I/Os. Here $\lg^* N$ denotes the iterated logarithm, i.e. the smallest value of h such that $\lg^{(h)} N$ is constant. Note that the previous fastest I/O-model data structure with linear space usage answers queries in $O((N/B)^\varepsilon + K/B)$ I/Os! The best previous data structure with optimal query time uses $O(N \lg\lg N \lg\lg\lg N)$ space.

We also extend our I/O-model results to the case where points have coordinates on a $U \times U$ grid. In this case, we reduce the $\lg_B N$ term in the query bounds to an optimal $\lg\lg_B U$ term. Finally, for points on an $N \times N$ grid, we reduce it all the way to $O(1)$. This again improves over all the previous best tradeoffs.

Using a standard reduction, all these results (except for the $N \times N$ grid) extend to four-sided queries at the cost of increasing the space by a $\lg N$ factor. This improves over all the previous tradeoffs in both models.

Finally, in Section 6, we establish a tight lower bound for one-dimensional categorical range counting. This lower bound is established by giving an elegant reduction from (standard) two-dimensional range counting to one-dimensional categorical range counting. Not only does this establish a tight $\Omega(\lg N / \lg\lg N)$ lower bound on the query time of any $N \lg^{O(1)} N$ space word-RAM data structure [22], but combined with the previous reduction from one-dimensional categorical range counting to two-dimensional standard range counting [13], it leads to the very surprising conclusion that the two problems are identical!

1.4 Note on Duplication of Colors. In previous work, categorical range reporting data structures were allowed to report the same color from within a query range a constant number of times, and not necessarily just once (note that since it is a constant number of times, the $O(K)$ or $O(K/B)$ term in the reporting time remains unchanged). This assumption can easily be removed in the word-RAM by removing duplicates in the output either by using an array indexed by color or hashing. The assumption is a little more questionable in the I/O-model, but we still make it and note that this was also assumed in the previous I/O-model paper by Nekrich [21] (except for the linear space and $O((N/B)^\varepsilon + K/B)$ query cost solution). Whether it is possible to design data structures matching ours, but where each color is reported exactly once, remains an interesting open problem.

2 Three-Sided Categorical Range Reporting

In this section we present a linear space data structure for answering three-sided categorical range queries in $O(\lg(N/B) + K/B)$ I/Os based on ideas from the internal-memory data structures by Mortensen [20] and Shi and JaJa [24]. This is not yet the query time we are aiming for, but the data structure will be used as a building block for our final data structure in the next section.

In Section 2.1 we first describe a standard reduction from three-sided categorical range reporting to offline partially-persistent (insertion-only) *one-dimensional* categorical range reporting. Then, we show how to solve this one-dimensional problem by dividing the points into buckets of size B and building a binary tree of height $O(\lg(N/B))$ over the buckets. Following [20] and [24], we then assign a y-coordinate representing a level in the binary tree to each one-dimensional point. We show that a one-dimensional range query on the original points can be answered using two three-sided non-colored range queries on two transformed point sets that have only $O(\lg(N/B))$ different y-coordinates.

In Section 2.2 we first construct a simple partially-persistent one-dimensional data structure, which can be used for each of the $O(\lg(N/B))$ y-coordinates in the above reduction. This data structure supports range queries in $O(1 + K/B)$ I/Os when given a pointer to the disk block storing the leftmost point inside a query range. To find these leftmost points for each of the $O(\lg(N/B))$ one-dimensional data structures, we construct a rectangular subdivision of the plane for each of them, and show that a point location query can be used to find the desired leftmost points. We overlay all $O(\lg(N/B))$ rectangular subdivisions and use a known rectangle-stabbing data structure with query complexity $O(\lg(N/B) + K/B)$ I/Os to perform all $O(\lg(N/B))$ point locations in *parallel*. Note that the output size is only $O(\lg(N/B))$, so the total cost of this rectangle-stabbing query is $O(\lg(N/B))$ I/Os.

2.1 Reduction to Partially-Persistent Three-Sided Non-Colored Range Reporting. The three-sided categorical range reporting problem can be solved using a data structure for the *offline insertion-only partially-persistent one-dimensional categorical range reporting problem* [13]: In this problem, we are given a sequence of N insertions of one-dimensional colored points. The insertions have been assigned increasing timestamps, which we think of as the insertion times. Given a query interval $[x_0 : x_1]$ and a timestamp t, we must support reporting all colors contained in the interval $[x_0 : x_1]$ as if only updates with timestamp at most t had been performed on the data structure.

The three-sided categorical range reporting problem can be reduced to this problem by the following procedure: Given the N colored input points in $\mathbb{R}^2$, sweep a horizontal line from $y = -\infty$ towards ∞ and insert points (x, y) into the offline one-dimensional categorical range reporting data structure when they are hit by the sweep line, i.e. we map each point (x, y) to the one-dimensional point with coordinate x and insertion timestamp y. A query $[x_0 : x_1] \times (-\infty : y_0]$ now translates into the one-dimensional query $[x_0 : x_1]$ with the query timestamp set to y.

One-Dimensional Insertion-Only Partially-Persistent Categorical Range Reporting. We now show how to solve one-dimensional insertion-only partially-persistent categorical range reporting using a data structure for partially-persistent two-dimensional three-sided non-colored range reporting where the number of different y-coordinates is $O(\lg(N/B))$. As mentioned, our approach follows [20, 24] but with the crucial difference that we construct a number of buckets to obtain $O(\lg(N/B))$ instead of $O(\lg N)$ different y-coordinates. This small difference is the key to our later improvements in Section 3.

We first divide the set of x-coordinates of all input points into N/B buckets of size B and the points in each bucket are stored together with their insertion timestamps in $O(1)$ disk blocks. Then we conceptually build a balanced binary tree of height $O(\lg(N/B))$ with these buckets as leaves. Let $b(p)$ denote the bucket containing a point p, and $h(v)$ the height of node v in the tree, counting from the leaves up, so the leaves have height 0.

Now process the input points in order of insertion time, and let p with color χ be any inserted point for which there is no χ-colored point right of p in $b(p)$ (at the time of the insertion). We maintain a value $y_l(p)$, which is the height $h(a)$ of the lowest ancestor a of $b(p)$ for which there are no inserted points with color χ right of p in the range covered by the subtree rooted at a (see Figure 1(a)). At any step during the insertions, we let P_l denote the two-dimensional point set consisting of the set of currently inserted one-dimensional points that are the rightmost in their bucket. A point p is represented in P_l as the two-dimensional point with the same x-coordinate as p, and with y-coordinate $y_l(p)$.

When inserting a point p, it might be that there was already at least one point with color χ in the subtree rooted at a. If this is the case, we find the rightmost point p' among all points in the subtree rooted at a with color χ. We then find the lowest common ancestor a' of p and p', delete p' from P_l and in case $b(p') \neq b(p)$ we re-insert p' with y-coordinate $h(a') - 1$. This operation re-establishes the representation of each point in P_l, i.e. any point p which is the rightmost of its color inside $b(p)$, is represented by $(x, y_l(p))$ in P_l, where x is the x-coordinate of p.

The point set P_l can thus be maintained with at most one insertion and one deletion for each one-dimensional point that is inserted. We now store P_l in a partially-persistent data structure for answering three-sided range queries (no colors, but insertions and deletions), but where the number of different y-coordinates is only $O(\lg(N/B))$. The partial persistence allows us to query P_l as it was after processing only the one-dimensional insertions with timestamp at most t, for any t. This partially-persistent structure is described in Section 2.2.

To answer a query $[x_0 : x_1]$ at timestamp t, we first find the lowest common ancestor a of x_0 and x_1. In case $b(x_0) = b(x_1)$ we spend O(1) I/Os to read this bucket from disk and simply solve the range reporting problem in internal memory at no I/O costs. Otherwise, we answer the query in two parts (so we may report colors twice). Let c_0 be the child of a containing $b(x_0)$ in its subtree and let c_1 be the child containing $b(x_1)$ in its subtree. A *left query* is used for finding the relevant colors in the subtree rooted c_0 and a *right query* is used for finding the relevant colors in the subtree rooted at child c_1 (see Figure 1(b)).

For answering left queries, we let x_{m} be the largest x-coordinate of a point in the subtree rooted at c_0, where we ignore insertion time (this value can be stored at c_0 as a preprocessing step). We then use a data structure for partially-persistent (insertion and deletion) three-sided range reporting to report all points (x, y) that are in P_l at timestamp t and where furthermore $x_0 \leq x \leq x_{\mathrm{m}}$ and $y \geq h(a) - 1$. Note that in case a point with color χ is present in the range $[x_0 : x_{\mathrm{m}}]$, either the point is the rightmost with color χ in the subtree of c_0, in which case its height is at least $h(c_0) = h(a) - 1$, or otherwise, there is a representative of the same color with a higher x-coordinate and height at least $h(a) - 1$. Moreover, no two points with the

same color in the subtree of c_0 can have height at least $h(a)-1$, so for all colors in the range $[x_0 : x_{\mathrm{m}}]$, there is exactly one point of that color which is reported by the query on P_l.

For answering right queries we define $y_r(p)$ analogously to $y_l(p)$, that is, $y_r(p)$ is the height of the lowest ancestor such that p is the leftmost point of color χ in the subtree. Similarly, P_r is defined as P_l, with rightmost replaced by leftmost and $y_l(p)$ replaced by $y_r(p)$. The query is also analogous, we find all points $(x, y) \in P_r$ for which $x_{\mathrm{m}} < x \leq x_1$ and $y \geq h(a) - 1$ at timestamp t.

2.2 Offline Partially-Persistent Three-Sided Non-Colored Range Reporting. As the number of different y-coordinates is $O(\lg(N/B))$, we can afford to do an I/O for each of them. Thus, if we can construct a one-dimensional partially-persistent data structure with query time $O(1 + K/B)$ for each y-coordinate, we have solved the problem (we just query the data structures stored for all y-coordinates inside the query range).

For the one-dimensional reporting problems, we simply construct a partially-persistent B-tree [6] over the points. This partially-persistent B-tree stores points in sorted order in the leaves, and we simply augment the leaves with pointers between them. These pointers can easily be maintained partially-persistent with only a constant factor overhead. Thus if we are given a pointer to the leaf containing x_0 at time t, we can report the K points in a range $[x_0 : x_1]$ in $O(1 + K/B)$ I/Os simply by scanning leaves until a point with x-coordinate greater than x_1 is encountered. Our goal is therefore to find such a pointer for all $O(\lg(N/B))$ different y-coordinates, using only $O(\lg(N/B))$ I/Os.

Reduction to Partially-Persistent Rectangle Stabbing. Consider one of the partially-persistent B-trees stored for a particular y-coordinate. Each insertion and deletion into such a tree changes $O(1)$ leaves amortized. We thus represent each version of each leaf as a rectangle $[x_0 : x_1] \times [t_0 : t_1]$, where $[x_0 : x_1]$ is the range of coordinates represented by the leaf after the update (starting from but excluding the last point in the previous leaf, or $-\infty$ for the first leaf), t_0 is the timestamp of the update and t_1 is the timestamp of the next update to the leaf. To find which leaf contains x at timestamp t, we need to find the rectangle containing the point (x, t), that is, we need to answer a *point location query* in a set of disjoint axis-aligned rectangles. The total number of rectangles is equal to the total number of leaf changes in the partially-persistent B-tree, which is linear in the total number of updates, N.

We superimpose the rectangular subdivisions for all $O(\lg(N/B))$ different partially-persistent B-trees and store them in a linear space rectangle-stabbing data structure of Arge et al. [5], in which each rectangle is augmented with a pointer to the leaf it represents. This data structure answers rectangle-stabbing queries (report all K rectangles containing a query point) in $O(\lg(N/B) + K/B)$ I/Os. Note that we report at most $O(\lg(N/B))$ rectangles, so we have the following theorem. We note that the idea of solving a number of point location queries in parallel using a rectangle-stabbing query has appeared before in the work of Afshani et al. [1].

THEOREM 2.1. *There exists a data structure for three-sided categorical range reporting that answers queries in $O(\lg(N/B) + K/B)$ I/Os using linear space.*

3 Final Data Structure

In this section, we show that using the data structure from Theorem 2.1 as a building block, we can obtain a data structure that uses $O(Nh)$ space and answers queries in $O(\lg_B N + \lg^{(h)}(N/B) + K/B)$ I/Os for any integer $h > 1$ (not necessarily constant). Before presenting our solution, we need the following easy result for two-sided categorical range reporting:

LEMMA 3.1. *There exists a data structure for two-sided categorical range reporting, using linear space and answering queries in optimal $O(\lg_B N + K/B)$ I/Os.*

Proof. Assume the queries are unbounded towards $-\infty$ in both the x- and y-direction. To solve the problem, we maintain a partially persistent B-tree $\mathcal{T}$, which is initially empty. Now conceptually sweep a vertical line from $x = -\infty$ to ∞, and whenever a point (x, y) from the input intersects the sweep line, check if it has the lowest y-coordinate among all points of the same color which have been encountered so far. If so, we first delete the previous lowest point of the same color (if any) from $\mathcal{T}$ and set the timestamp of the deletion to $2x - 1$. We then insert the newly encountered point with key y and timestamp $2x$. To answer a query $(-\infty, x_0] \times (-\infty, y_0]$ we simply ask the (non-colored) range reporting query $(-\infty, y_0]$ on $\mathcal{T}$ with the timestamp being $2x_0$. The correctness follows immediately. □

With Theorem 2.1 and Lemma 3.1 established, we are ready to bootstrap with these two solutions to obtain an even faster solution for three-sided queries.

3.1 Bootstrapping. In the following, we use the data structure of Theorem 2.1 to reduce the query time even further. Our key idea is to only query the data structure from Theorem 2.1 in case we know we have

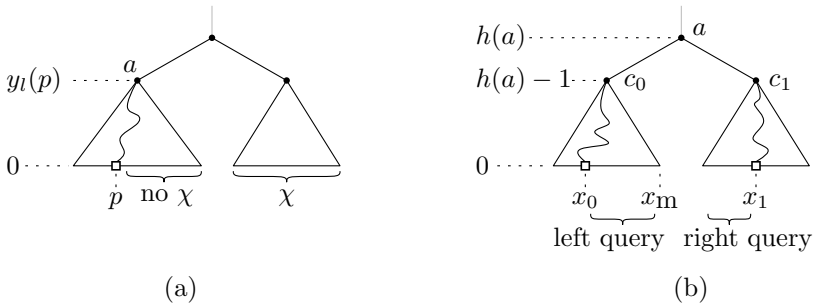

Figure 1: (a) Assignment of y coordinates. (b) Answering queries.

$\Omega(B \lg(N/B))$ points to report. If we have fewer points to report, then we can reduce the problem size we are working on essentially by an exponential factor. The details follow below.

Our final data structure takes as input an integer parameter $h \geq 1$ (not necessarily constant). If $h = 1$, we simply store the data structure of Theorem 2.1 which has space usage $O(Nh)$ and query time $O(\lg^{(1)}(N/B) + K/B)$. In this case, we say that the data structure is a *leaf* data structure.

If $h > 1$, we do the following: First of all, we store all input points in the data structure of Theorem 2.1, which we will query in case we are sure we have $\Omega(B \lg(N/B))$ colors to report. Second, we sort the input points by x-coordinate and divide them into buckets of $B \lg^2(N/B)$ consecutive points each. For each bucket, we implement the data structure from Lemma 3.1 such that we can answer two-sided queries on the points in a bucket. This uses linear space.

We then conceptually construct a complete binary search tree on the buckets. For a bucket b, we now store for each ancestor v in the binary search tree two lists of points, $S_\ell(b, v)$ and $S_r(b, v)$, as defined below. Intuitively, $S_\ell(b, v)$ contains a set of $O(B \lg(N/B))$ points with unique colors in the subtrees hanging off to the left of the path from b to v (see Figure 2), and similarly $S_r(b, v)$ contains points from the subtrees hanging off to the right of the path from b to v.

More formally, let v be an ancestor of bucket b and $p(b, v)$ the path from b to v (excluding b and v). Now construct the set $S'_\ell(b, v)$ of all points in subtrees hanging off to the left of $p(b, v)$, and similarly $S'_r(b, v)$ the set of points in subtrees hanging off to the right of $p(b, v)$ (see Figure 2). For each b and v, let $S''_\ell(b, v)$ be the set of points that each have the smallest y-coordinate among the points of the same color in $S'_\ell(b, v)$, i.e. $S''_\ell(b, v) := \{q \in S'_\ell(b, v) \mid \forall q' \in S'_\ell(b, v) : \chi(q') = \chi(q) \Rightarrow y(q') > y(q)\}$. Define $S''_r(b, v)$ symmetrically, with l replaced by r. Then, select the $B \lg(N/B)$ points with lowest y-coordinate in $S''_\ell(b, v)$ to form the set $S_\ell(b, v)$ (do the same for $S_r(b, v)$). For a bucket b, we store $S_\ell(b, v)$ and $S_r(b, v)$ for each ancestor v as a list of points ordered by y-coordinate. As we have $O(N/B \lg^2(N/B))$ buckets and $O(B \lg(N/B))$ points for each of the $O(\lg(N/B))$ ancestors of a bucket, we need $O(N/B \lg^2(N/B) \cdot B \lg^2(N/B)) = O(N)$ space for storing these lists. We note that lists similar in spirit were used in the solution of [21].

Finally, we store the same data structure recursively on each bucket, but with h decreased by 1. Note that the ratio between the number of points and disk block size goes from N/B to $\lg^2(N/B)$ in the recursively constructed data structures, and when the recursion bottoms out (h decreases to 1), the ratio has decreased to $O((\lg^{(h-1)}(N/B))^2)$.

Answering Queries. To answer a query $[x_0 : x_1] \times (-\infty : y_0]$, first determine the successor of x_0, denoted $s(x_0)$, and the predecessor of x_1, denoted $p(x_1)$, among the x-coordinates of the input points, as well as their respective ranks. This can be done by storing one global B-tree on the x-coordinates of all input points. We first determine whether $s(x_0)$ and $p(x_1)$ lie in the same leaf data structure. Since our recursive data structures are constructed as complete binary trees on the x-coordinates of the points, this can be determined solely from the ranks of $s(x_0)$ and $p(x_1)$, i.e. no I/Os are needed (if we want a solution that is strictly comparison based, we can determine this as described below for the case when they are not in the same leaf structure). If they are in the same leaf data structure, we simply answer the query using the data structure of Theorem 2.1 stored at the leaf.

If $s(x_0)$ and $p(x_1)$ are not in the same leaf data structure, we determine the first node, v, in all the recursively constructed binary trees, for which $s(x_0)$ and $p(x_1)$ lie in different subtrees. Again, since the entire data structure (including all the recursive data structures) is a complete binary tree with auxiliary data structures stored at h different levels, we can store the binary tree blocked on disk such that the path from the

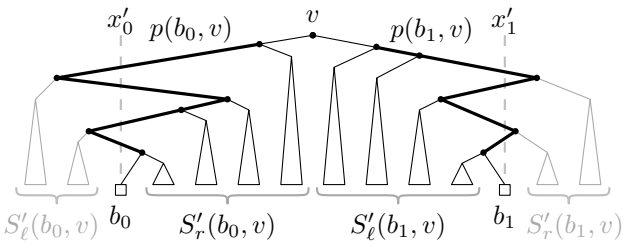

Figure 2: The answer to a query with x-range $[x_0 : x_1]$ is contained in $S'_r(b_0, v)$,$S'_\ell(b_1, v)$, b_0 and b_1.

root to v can be traversed in $O(\lg_B N)$ I/Os (cut the tree at every $\lg_B N$ levels and store each micro tree in a disk block). Let $\mathcal{T}_v$ be the complete binary tree containing v and let $1 < h' \leq h$ denote the value of h used when constructing $\mathcal{T}_v$. The number of points in $\mathcal{T}_v$ is $O((\lg^{(h'-1)}(N/B))^2 B)$. We now find the bucket $b_0 \in \mathcal{T}_v$ containing $s(x_0)$ and $b_1 \in \mathcal{T}_v$ containing $p(x_1)$. Since $s(x_0)$ and $p(x_1)$ lie in different subtrees of v, we have $b_0 \neq b_1$. We now answer the query in at most four parts, so each color may be reported up to four times. We first report the colors of the points in b_0 and b_1 that intersect the query range using the two-sided data structures of Lemma 3.1 stored over all points in those buckets, where in b_0 we use the query range $[x_0, \infty) \times (-\infty, y_0]$ and in b_1 we use the query range $(-\infty, x_1] \times (-\infty, y_0]$. Then, we scan $S_r(b_0, v)$ and $S_\ell(b_1, v)$, starting from the point with smallest y-coordinate and reporting the colors of points until their y-coordinate is greater than y_0. In case the two lists both return less than all of their points, we are sure to have reported all colors in the range since any other colors either do not have points in the subtrees hanging off to the right (left) of the path from v to b_0 (v to b_1), or the points in there have y-coordinates greater than y_0 (otherwise they would have been represented in $S_r(b_0, v)$ or $S_\ell(b_1, v)$). In case all points are reported from one of the two lists, we discard the output so far and instead query the data structure of Theorem 2.1 containing all points in T_v. Since the output size is $K = \Omega(\min\{|S_r(b_0, v)|, |S_\ell(b_1, v)|\}) = \Omega(B \lg((\lg^{(h'-1)}(N/B))^2)) = \Omega(B \lg^{(h')}(N/B))$ in this case, this costs $O(\lg((\lg^{(h'-1)}(N/B))^2) + K/B) = O(\lg^{(h')}(N/B) + K/B) = O(K/B)$ I/Os.

Analysis. We first analyze the query time. Determining the successor, $s(x_0)$, of x_0 and predecessor, $p(x_1)$, of x_1, including their ranks, costs $O(\lg_B N)$ I/Os. If $s(x_0)$ and $p(x_1)$ are in the same leaf data structure, we query the data structure of Theorem 2.1 on a set of $O((\lg^{(h-1)}(N/B))^2 B)$ points, costing $O(\lg^{(h)}(N/B) + K/B)$ I/Os. If $s(x_0)$ and $p(x_1)$ are not in the same leaf data structure, we query two two-sided data structures of Lemma 3.1, both stored on $O(N)$ points, costing $O(\lg_B N + K/B)$ I/Os. Scanning the two lists $S_r(b_0, v)$ and $S_\ell(b_1, v)$ costs $O(K/B)$ I/Os since we only continue scanning while we report new colors. In case we report all points from one list, we spend another $O(K/B)$ I/Os querying the three-sided data structure stored on T_v. Thus the total query cost is $O(\lg_B N + \lg^{(h)}(N/B) + K/B)$ I/Os as claimed. For the space usage, observe that each recursive level uses linear space, thus the space is bounded by $O(Nh)$.

THEOREM 3.1. *For any integer parameter $h \geq 1$ (not necessarily constant), there exists an $O(Nh)$ space data structure answering three-sided categorical range queries in $O(\lg_B N + \lg^{(h)}(N/B) + K/B)$ I/Os, where K is the number of colors reported.*

Four-sided queries. The three-sided data structure above also provides an improved data structure for four-sided queries: Simply use the binary range trees [7] to obtain a data structure for four-sided queries, at the cost of increasing the space by a $\lg N$ factor.

4 Input Points on a Grid

In this section, we show how to reduce the query time of our I/O-model data structures when the input points are given on a grid. Our aim is to reduce the $\lg_B N$ term and from the analysis in Section 3, we see that this term arises from two places:

- Finding the successor of x_0, predecessor of x_1, their ranks, and the lowest common ancestor node.
- Querying a two-sided data structure from Lemma 3.1 on a bucket.

The first part is easily dealt with since predecessor search in an integer universe of size U can be done in $O(\lg \lg_B U)$ I/Os when we allow manipulation of integers [23] and in $O(1)$ I/Os in a universe of size N (simply store an array). Furthermore, our data structure is a complete binary tree, so the lowest common ancestor can be determined directly from the ranks. For the two-sided structure from Lemma 3.1, recall that we are solving this problem by asking one-sided queries on a partially-persistent B-tree, i.e. queries

of the form $(-\infty, y_0]$ at some timestamp x_0. It is easily seen that a partially-persistent B-tree can be implemented such that if we are given a pointer to the disk block containing the smallest element stored in a partially-persistent tree at a given timestamp x_0, then we can immediately jump to that disk block and scan points in the tree until the keys exceed y_0, i.e. we can answer the query in $O(1 + K/B)$ I/Os if given such a pointer. Now if we augment the partially-persistent B-tree structure with a sorted list of pointers, one for each operation performed on the structure, where each pointer points to the disk block containing the smallest element in the tree after the corresponding update operation, then we only need to determine where the update operation immediately preceding the query timestamp x_0 is located in this sorted array. This is a predecessor search problem. Now observe that the timestamps used in our solution are in $[2U]$ when the points are on a $U \times U$ grid, hence we conclude

COROLLARY 4.1. *For any integer parameter $h \geq 1$ (not necessarily constant), there exists an $O(Nh)$ space data structure answering three-sided categorical range queries in $O(\lg \lg_B U + \lg^{(h)}(N/B) + K/B)$ I/Os when the N input points are given on a $U \times U$ grid, where K is the number of colors reported.*

COROLLARY 4.2. *For any integer parameter $h \geq 1$ (not necessarily constant), there exists an $O(Nh)$ space data structure answering three-sided categorical range queries in $O(\lg^{(h)}(N/B) + K/B)$ I/Os when the N input points are given on an $N \times N$ grid, where K is the number of colors reported.*

5 Word-RAM Data Structures

In the following, we show how to extend our I/O-model solutions to obtain optimal solutions in the word-RAM. We first note that the I/O-model data structure of Theorem 2.1 translates directly to the following word-RAM result:

COROLLARY 5.1. *There exists a word-RAM data structure for three-sided categorical range reporting, using linear space and answering queries in $O(\lg N + K)$ I/Os.*

$U \times U$ Grid in the Word-RAM. To obtain optimal query times in the word-RAM, we bootstrap with the data structure from Corollary 5.1 by using the same layout as Section 3, i.e. we partition the points into buckets of size $\lg^2 N$ and construct a complete binary tree on the buckets. For the buckets, we do not recurse but instead store the word-RAM data structure of Corollary 5.1 directly. Since the number of points in a bucket is $\lg^2 N$, the cost of querying such a data structure is $O(\lg(\lg^2 N) + K) = O(\lg \lg N + K)$. For the lists $S_r(b, v)$ and $S_\ell(b, v)$, we use the topmost $\lg N$ points and hence the space usage is linear.

When coordinates lie on a $U \times U$ grid, we can find the successor (and rank) of x_0 and the predecessor (and rank) of x_1 in $O(\lg \lg U)$ time using a predecessor-search data structure [23]. The lowest common ancestor v and the lists $S_r(b_0, v)$ and $S_\ell(b_1, v)$ can be found in constant time by word-operations and indexing in the word-RAM, so the total query time is $O(\lg \lg U + K)$.

THEOREM 5.1. *There exists a linear space data structure answering three-sided categorical range queries on a set of points in $[U] \times [U]$ in $O(\lg \lg U + K)$ time in the word-RAM, where K is the number of colors reported.*

$N \times N$ Grid in the Word-RAM. For input points on an $N \times N$ grid (i.e. rank space), we bootstrap with the data structure of Theorem 5.1, which we query in case we have $\Omega(\lg \lg N)$ colors to report. We again construct the layout of Section 3, i.e. we partition into buckets and construct a complete binary tree on them. This time the buckets contain $\lg N \lg \lg N$ points each and the size of each $S_\ell(b, v)$ and $S_r(b, v)$ is $\lg \lg N$, so we use $O(N/(\lg N \lg \lg N) \cdot \lg N \cdot \lg \lg N) = O(N)$ space for storing the $S_\ell(b, v)$ and $S_r(b, v)$ lists. We do not recurse on the buckets, but instead we store a data structure that answers three-sided queries on $O(\lg N \lg \lg N)$ points in $O(1 + K)$ time and with linear space. This data structure is described in Section 5.1. This data structure also uses a lookup table of size $O(N^\varepsilon)$, where $\varepsilon > 0$ is an arbitrarily small constant, which can be shared among all buckets.

We query the data structure as in Section 3, except we use the $O(1 + K)$ query time data structure for handling the buckets. The elements $s(x_0)$ and $p(x_1)$ and their ranks can be found in constant time by storing an array over all x-coordinates. Hence, in this case the query time is $O(1 + K)$.

THEOREM 5.2. *There exists a linear space data structure answering three-sided categorical range queries on a set of N points in $[N] \times [N]$ in $O(1 + K)$ time in the word-RAM, where K is the number of colors reported.*

5.1 Word-RAM Queries on Few Points.

In this section, we show how to answer three-sided categorical range reporting queries on a set P of $m = O(\lg N \lg \lg N)$ points with coordinates on an $N \times N$ grid. Our solution uses a lookup table of size $O(N^\varepsilon)$, where $\varepsilon > 0$ is an arbitrarily small constant. This lookup table is independent of the input set and hence can be shared by any number of data structures. In addition, our solution uses linear space (i.e. $O(m \lg N)$

bits) and answers queries in $O(1+K)$ time. The techniques used are standard.

First we store a Fusion tree [12] on the m input points. This Fusion tree uses linear space and allows us to answer predecessor queries over the input points in constant time. Thus we can map a query into rank space coordinates w.r.t. P in constant time, i.e. we can consider all points in P as lying on an $m \times m$ grid and we can assume that the cornerpoints of a query are also represented by points on the grid. Furthermore, we replace the colors of the input points by integers in $[m]$, such that two points with the same color are assigned the same new color in $[m]$ (i.e. a rank space reduction on the colors). Note that we can trivially recover the old coordinates and colors from our new representations of points using lookup arrays of total size $O(m \lg N)$ bits. We let P^* denote our transformed set of input points.

We now partition the points in P^* into m/Δ buckets of $\Delta = \delta \lg N / \lg\lg N$ points each, where $\delta > 0$ is a sufficiently small constant. The ith bucket consists of the Δ points in P^* with x-coordinates $\{i\Delta, \ldots, (i+1)\Delta - 1\}$. Now observe that a single bucket can be completely described in $\Delta 2 \lg m < 3\Delta \lg\lg N = 3\delta \lg N$ bits, simply by writing down the y-coordinate and color of each point, one after the other. Our shared lookup table has one entry for every possible combination of a bucket (i.e. bit representation of a bucket) and a query inside that bucket. Since a three-sided query inside a bucket can be described in $3 \lg m$ bits, the total number of entries in the lookup table is bounded by $2^{3\delta \lg N + 4 \lg\lg N} = O(N^\varepsilon)$ for an arbitrarily small constant $\varepsilon > 0$. Each entry of the lookup table simply stores the answer of the corresponding query on the corresponding bucket (note that things are in rank space, but we can recover the original points and colors using the arrays mentioned earlier). The total size of the lookup table can still be bounded by $O(N^\varepsilon)$ for any constant $\varepsilon > 0$.

Thus for any query range $[x_0 : x_1] \times (-\infty : y]$ (in rank space), we can partition the query into three disjoint queries: Let i_0 be the index of the bucket containing x_0 and let i_1 be the index of the bucket containing x_1. The query is partitioned into the following parts:

1. $[x_0 : (i_0+1)\Delta - 1] \times (-\infty : y]$. This part of the query is answered using the lookup table. This takes $O(1+K)$ time.

2. $[(i_0+1)\Delta : i_1\Delta - 1] \times (-\infty : y]$. We show how to handle this part below.

3. $[i_1\Delta : x_1] \times (-\infty : y]$. This part is also answered in $O(1+K)$ time using the lookup table.

Note that if a query lies completely within a bucket, we can immediately answer the query using the lookup table. Thus all that remains is to show how we answer the query $[(i_0+1)\Delta : i_1\Delta - 1] \times (-\infty : y]$. Note that the x-range of this query completely spans a number of consecutive buckets and stops at the border between two buckets on each side. The last part of our data structure therefore consists of a sorted list for every range of buckets. For a range of buckets, we first collect all points contained in those buckets. From this set of points, we select for each color the point with lowest y-coordinate. We store this subset of points in sorted order of y-coordinate, packed into words. Now given the above query range $[(i_0+1)\Delta : i_1\Delta - 1] \times (-\infty : y]$ we start by examining the first point in the sorted list stored for the range of buckets $i_0 + 1, \ldots, i_1 - 1$. If the y-coordinate of the point is at most y, we report it and examine the next point in the list. This continues until a point with y-coordinate greater than y is encountered, or until the end of the list is reached. Clearly this correctly reports the colors in the query range and the time needed is $O(1+K)$. Since any point is stored in at most $(m/\Delta)^2 = O(\lg^4 \lg N)$ such lists, the total space usage is bounded by $O(m \lg^4 \lg N \lg m) = O(m \lg^5 \lg N) = o(m \lg N)$ bits. We conclude that the space usage of our data structure is dominated by the Fusion tree and the arrays mapping points and colors from rank space to their original coordinates. Thus the space usage is linear.

Four-sided queries. The three-sided data structure above also provides an improved data structure for four-sided queries in the word-RAM. Again we simply use range trees [7] to get the following result:

THEOREM 5.3. *There exists an $O(N \lg N)$ space data structure answering four-sided categorical range queries in $O(\lg\lg U + K)$ time in the word-RAM when the input points lie on a $U \times U$ grid. Here K is the number of colors reported.*

6 Categorical Range Counting

In this section, we present a reduction from two-dimensional range counting to one-dimensional categorical range counting. This establishes a tight $\Omega(\lg N / \lg\lg N)$ query time lower bound for any categorical range counting data structure in the word-RAM that uses $N \lg^{O(1)} N$ space. Combined with the previous reduction in the other direction [13], this also shows that the two problems are identical.

We reduce from two-dimensional *dominance* counting, i.e. counting where the query rectangles are of the form $(-\infty : x'] \times (-\infty : y']$. Note that dominance counting solves counting with four-sided ranges by adding and

subtracting the answers to a constant number of queries.

Let P be the N input points to a two-dimensional dominance range counting problem and assume all coordinates are positive (this can easily be achieved through a translation of the points). We map each input point to two one-dimensional colored points: If an input point p has coordinates (x, y), then it is mapped to the two points with coordinates $-x$ and y, respectively. The color of the two constructed points is set to p (or some value uniquely identifying p). Letting P' denote the constructed set of $2N$ one-dimensional points, we construct a one-dimensional categorical range counting data structure on P'. Finally, we construct the set P^* consisting of all points in P', but where the colors are changed such that every point has a unique color. We also construct a one-dimensional categorical range counting data structure on P^*. Note that this corresponds to a standard one-dimensional range counting data structure since all colors are distinct.

When presented with a dominance counting query $(-\infty : x'] \times (-\infty : y']$, we ask the one-dimensional categorical range counting query $[-x' : y']$ on the two data structures constructed for P' and P^*. Letting t' and t^* denote the answers to these queries, we return as our result $t^* - t'$.

To see that we have correctly answered the dominance counting query $(-\infty : x'] \times (-\infty : y']$, let p be an input point with coordinates (x, y). We have four cases:

1. $x \leq x'$ and $y \leq y'$: The point p is inside the dominance query. In this case, observe that the one-dimensional query range $[-x' : y']$ contains both one-dimensional points representing p. Therefore, p contributes 2 to the value t^* but only 1 to the value t' (the two representatives have the same color), i.e. it contributes 1 to the returned answer.

2. $x \leq x'$ and $y > y'$: The point p is outside the dominance query. In this case, the one-dimensional query range $[-x' : y']$ contains one of the two points representing p. Thus p contributes 1 to both t^* and t' and hence 0 to the returned answer.

3. $x > x'$ and $y \leq y'$: Symmetric to case 2.

4. $x > x'$ and $y > y'$: The point p is outside the dominance query. In this case, the one-dimensional query range $[-x' : y']$ contains none of the two points representing p. Thus p contributes 0 to both t^*, t' and to the returned answer.

It follows immediately from the above case analysis that we have reduced two-dimensional dominance counting to one-dimensional categorical range counting.

References

[1] P. Afshani, L. Arge, and K. D. Larsen. Orthogonal range reporting: Query lower bounds, optimal structures in 3d, and higher dimensional improvements. In *Proc. 26th ACM Symposium on Computational Geometry*, pages 240–246, 2010.

[2] P. K. Agarwal, S. Govindarajan, and S. Muthukrishnan. Range searching in categorical data: Colored range searching on grid. In *Proc. 10th European Symposium on Algorithms*, pages 17–28, 2002.

[3] S. Alstrup, G. S. Brodal, and T. Rauhe. New data structures for orthogonal range searching. In *Proc. 41st IEEE Symposium on Foundations of Computer Science*, pages 198–207, 2000.

[4] L. Arge, V. Samoladas, and J. S. Vitter. On two-dimensional indexability and optimal range search indexing. In *Proc. 18th ACM Symposium on Principles of Database Systems*, pages 346–357, 1999.

[5] L. Arge, V. Samoladas, and K. Yi. Optimal external memory planar point enclosure. *Algorithmica*, 54:337–352, 2009.

[6] B. Becker, S. Gschwind, T. Ohler, B. Seeger, and P. Widmayer. An asymptotically optimal multiversion B-tree. *VLDB Journal*, 5:264–275, 1996.

[7] J. L. Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, 1980.

[8] P. Bozanis, N. Kitsios, C. Makris, and A. Tsakalidis. New upper bounds for generalized intersection searching problems. In *Proc. 22nd International Colloquium on Automata, Languages, and Programming*, pages 464–474, 1995.

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

[10] T. M. Chan, K. G. Larsen, and M. Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proc. 27th ACM Symposium on Computational Geometry*, pages 1–10, 2011. See also arXiv:1011.5200.

[11] B. Chazelle. Lower bounds for orthogonal range searching: I. The reporting case. *Journal of the ACM*, 37(2):200–212, 1990.

[12] M. L. Fredman and D. E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47(3):424–436, 1993. See also STOC'90.

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

[14] J. M. Hellerstein, E. Koutsoupias, D. P. Miranker, C. H. Papadimitriou, and V. Samoladas. On a model of indexability and its bounds for range queries. *Journal of the ACM*, 49(1):35–55, 2002.

[15] J. JaJa, C. W. Mortensen, and Q. Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *Proc. 15th International*

Symposium on Algorithms and Computation, pages 558–568, 2004.

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

[17] H. Kaplan, N. Rubin, M. Sharir, and E. Verbin. Counting colors in boxes. In *Proc. 18th ACM/SIAM Symposium on Discrete Algorithms*, pages 785–794, 2007.

[18] K. G. Larsen. Higher cell probe lower bounds for evaluating polynomials. In *Proc. 53rd IEEE Symposium on Foundations of Computer Science*, 2012. To appear.

[19] K. G. Larsen and R. Pagh. I/O-efficient data structures for colored range and prefix reporting. In *Proc. 22nd ACM/SIAM Symposium on Discrete Algorithms*, pages 583–592, 2012.

[20] C. W. Mortensen. *Data structures for orthogonal intersection searching and other problems.* PhD thesis, 2006. Chapter 6.

[21] Y. Nekrich. Space-efficient range reporting for categorical data. In *Proc. 31st ACM Symposium on Principles of Database Systems*, pages 113–120, 2012.

[22] M. Pătraşcu. Lower bounds for 2-dimensional range counting. In *Proc. 39th ACM Symposium on Theory of Computation*, pages 40–46, 2007.

[23] M. Pătraşcu and M. Thorup. Time-space trade-offs for predecessor search. In *Proc. 38th ACM Symposium on Theory of Computation*, pages 232–240, 2006.

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

[25] V. V. Williams. Multiplying matrices faster than coppersmith-winograd. In *Proc. 44th ACM Symposium on Theory of Computation*, pages 887–898, 2012.