Categorical Range Reporting with Frequencies

Arnab Ganguly
Dept. of Computer Science, University of Wisconsin, Whitewater, USA
gangulya@uww.edu

J. Ian Munro
Cheriton School of Computer Science, University of Waterloo, Canada
imunro@uwaterloo.ca

Yakov Nekrich
Cheriton School of Computer Science, University of Waterloo, Canada
yakov.nekrich@googlemail.com

Rahul Shah
Dept. of Computer Science, Baton Rouge, USA
rahul@csc.lsu.edu

Sharma V. Thankachan
Dept. of Computer Science, University of Central Florida
sharma.thankachan@ucf.edu

Abstract

In this paper, we consider a variant of the color range reporting problem called *color reporting with frequencies*. Our goal is to pre-process a set of colored points into a data structure, so that given a query range Q, we can report all colors that appear in Q, along with their respective frequencies. In other words, for each reported color, we also output the number of times it occurs in Q. We describe an external-memory data structure that uses $O(N(1+\log^2 D/\log N))$ words and answers one-dimensional queries in $O(1+K/B)$ I/Os, where N is the total number of points in the data structure, D is the total number of colors in the data structure, K is the number of reported colors, and B is the block size.

Next we turn to an approximate version of this problem: report all colors σ that appear in the query range; for every reported color, we provide a constant-factor approximation on its frequency. We consider color reporting with approximate frequencies in two dimensions. Our data structure uses $O(N)$ space and answers two-dimensional queries in $O(\log_B N + \log^* B + K/B)$ I/Os in the special case when the query range is bounded on two sides. As a corollary, we can also answer one-dimensional approximate queries within the same time and space bounds.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Data Structures, Range Reporting, Range Counting, Categorical Range Reporting, Orthogonal Range Query

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.9

1 Introduction

Orthogonal range query is a fundamental problem in computational geometry and data structures, with a large number of applications in database theory, information retrieval, and text mining. In this problem, we want to pre-process a set of points so that later, given a query rectangle Q, we can report some desired information about the points contained in Q. Typical examples of this information are: the *maximum* (or *minimum*) value of a point in the range (in this case we associate a numeric value to each point), the most-frequent value in the range (or *mode*), the *median* value, the number of points in the range, etc.

In the *colored (or categorical) orthogonal range reporting problem*, every point is assigned a color (category). Given a query range Q, we must report distinct categories of points in the query range. Typically, we want to avoid reporting the same category many times. For this reason, colored variants of the range reporting problem are considered to be more difficult. In the *orthogonal range counting problem*, we keep the set of points in a data structure so that, given a query range Q, we can count the number of points in Q.

In this paper, we consider a variant of color range reporting queries that combines color reporting with counting: *given a query range Q, we report all colors that appear in Q; additionally, for each color within Q, we also report the number of its occurrences in Q.* Henceforth, such queries are called *color reporting with frequencies.* We also consider an approximate variant: *all colors in a query range Q are reported; for each color σ, which appears K_σ times in Q, we find a constant factor approximation for K_σ.*

We describe two data structures – one for color reporting with exact frequencies in one dimension and the other for color reporting with approximate frequencies in two dimensions, which are the first of their kind in the external memory model.

1.1 Applications to Databases

Situations where we need to report distinct categories of objects in a query range arise frequently in database applications. Suppose we have a database of employees, and we want to identify all employees whose salary lies in the range \$40,000 – \$80,000. This problem corresponds to one-dimensional point reporting queries. Now suppose that we want to report different job positions, such that at least one employee holding this position earns between \$40,000 and \$80,000. This problem corresponds to one-dimensional color reporting queries. In many cases, we may be interested in a more involved query: *group all employees who earn between \$40,000 and \$80,000 according to their job positions, then list all groups and the number of employees in each group.* This problem can be modeled by (one-dimensional) color reporting queries with frequencies. Furthermore we may be satisfied if we obtain approximate numbers of employees for every group. In this case, it is sufficient to answer the approximate variant of the color reporting query with frequencies.

Another prominent application of color range queries is in databases containing a collection of texts (documents). Given a query pattern Q, we want to report all documents that contain Q at least once; for each such document, we want to estimate how many occurrences of Q it contains. Such a query can be reduced to a color reporting query with frequencies.

1.2 Model of Computation

We study the color reporting with (approximate) frequencies problems in the external memory model. Here, the storage is divided into a main (or internal) memory of limited size and a large (potentially unbounded) external disk. We can perform arithmetic operations on data that is stored in the main memory. The data structure, however, does not fit into main memory and must be stored on the external disk. Using one I/O operation, we can read a contiguous block of B words from disk into the main memory or write a block of B words from main memory into the external disk. The main memory is much faster than the external memory; hence, all main memory operations are considered free and the algorithm complexity is measured in the number of I/O operations.

We remark that when we answer a reporting query in the external memory model, it is preferable to report $\Theta(B)$ objects (points, colors) with one I/O operation. Design of methods that fully exploit this property of the external memory model poses an additional challenge.

1.3 Notation

Throughout this paper, N is the total number of points, where each point is associated with a color. We denote by D the number of distinct colors. The number of colors in the answer to a query is denoted by K. The number of points with color σ in the answer to a query (or the frequency of σ) is denoted by K_σ. We assume that a word consists of $\Theta(\log N)$ bits. Unless specified otherwise, the space usage is measured in words of $\Theta(\log N)$ bits.

1.4 Previous and Related Work

Due to its importance, both in theory and in practice, the problem of reporting distinct colors of points in a range has received considerable attention. Both one-dimensional and multi-dimensional color (or categorical) range reporting were considered in the literature [13, 7, 11, 12, 8, 18, 19, 14, 23]. Other variants of this problem, such as top-k color reporting [14, 20], color maxima queries [25], color stabbing queries [10] were also studied.

Gupta et al. [11] present a main memory data structure that uses linear-space (i.e., $O(N)$-word) and answers one-dimensional colored range reporting queries in $O(\log N + K)$ time. Muthukrishnan [18] showed that the query time can be improved to $O(1 + K)$ if points are in the rank space, i.e., point coordinates are integers $1, 2, \ldots, N$. Later, Nekrich and Vitter [24] removed the rank space assumption, without any penalty in space or time.

In the external memory model, one-dimensional color reporting is usually solved by reducing it to three-sided point reporting (i.e., to a special case of two-dimensional point reporting when the query range is bounded on three sides). Arge et al. [3] described a linear-space solution that supports queries in $O(\log_B N + K/B)$ I/Os. Nekrich [21] described an $O(N)$-word data structure that supports queries in $O(\log \log_B U + K/B)$ I/Os when point coordinates are bounded by U. If point coordinates are bounded by N, another $O(N)$-space data structure from [21] supports queries in $O(\log_2^{(h)} N + K/B)$ I/Os, where h is a positive constant and $\log^{(h)} N$ denotes the logarithm function iterated h times.[1] Larsen and Pagh [15] showed that one-dimensional queries can be answered in $O(1 + K/B)$ I/Os using $O(N)$ words if point coordinates are bounded by N. They also studied the related problem of reporting the top-k colors, where each color is associated with a static weight and the k highest weighted colors within the query range have to be reported. Nekrich and Vitter [24] demonstrated that one-dimensional color reporting queries can be answered in $O(1 + K/B)$ I/Os even if point coordinates are arbitrary integers. Patil et al. [25] considered another variant of color reporting in one-dimension, called *colored range maxima queries*; here, one has to report the points corresponding to k distinct colors within the query range having the highest y-coordinate values. They presented multiple data structures with different space-and-time trade-offs, such as an $O(N)$-word data structure with $O(\log^* N + k/B)$ query time[2] and an $O(N \log^* N)$-word data structure with optimal $O(1 + k/B)$ query time.

Although this problem has been studied extensively and optimal results for many colored range reporting problems were achieved, limited progress has been made for the *color range reporting with frequencies* problem. Gupta, Janardan, and Smid [11] presented an internal memory data structure that answers a query in $O(\log N + K)$ time and uses $O(N \log N)$ space; see the type-2 counting problem in [11]. If we combine a result of Muthukrishnan [18] with an idea of Sadakane [29], we can obtain an $O(N)$-word data structure for reporting colored points and their frequencies in the rank space in $O(1 + K)$ time. However, this result

[1] $\log^{(h)} N = \log(\log^{(h-1)} N)$ for $h > 1$ and $\log^{(1)} N = \log N$

[2] $\log^* N$ is the smallest number i such that $\log^{(i)} N \leq 2$

Table 1 Our Results.

Query Type	Space	Time
1D/2D Two-sided with Constant Appx	$O(N)$ words	$O(\log^* B + \log_B N + K/B)$ I/Os
1D One-sided Exact	$O(N)$ words	$O(1 + K/B)$ I/Os
1D Two-sided Exact	$O(N(1 + \log^2 D/ \log N))$ words	$O(1 + K/B)$ I/Os

is not efficient in the external memory model, where the desired dependency on the output size is $O(f(N) + K/B)$ for some small function $f(N)$. To the best of our knowledge, there is no previous solution that exploits the properties of the external memory model. In contrast, the standard color reporting (without frequencies) can be solved in optimal $O(1 + K/B)$ time and $O(N)$ space in external memory [15, 24]. In this paper, we bridge this gap.

In the related approximate range counting problem, we must approximate the number of point within the query range. Chan and Wilkinson [9] presented an $O(N \log\log N)$-word data structure that answers approximate two-dimensional queries in $O(\log\log N)$ time. The previous paper by Nekrich [22] achieved a better approximation factor, compared to [9], at the cost of higher space usage. In another paper, Nekrich [23] presented an $O(N)$-space data structure that supports approximate point counting queries on an $N \times N$ grid in $O(1)$ I/Os, when the query range is bounded on three sides.

The second problem that we study combines color reporting with approximate point counting problem: we report all colors within the query range and provide a constant factor approximation for the frequency of each color. We primarily study the problem from the perspective of dominance reporting in two-dimensions, i.e., the query range is bounded on two sides; see e.g., [1] for a catalog on dominance queries. Straightforward reductions from this two-dimensional dominance problem can be used to answer a (two-sided) color reporting query with approximate frequencies in one dimension.

1.5 Our Results

Unless specified otherwise, we assume that points are in the rank space, i.e., point coordinates are integers $1, 2, \ldots, N$. More general scenarios can be reduced to this special case at the cost of increasing the query time by a small additive factor: if points are integers $1, \ldots, U$ for a parameter U, then the query cost increases by $O(\log_2 \log_B U)$; if point coordinates can assume arbitrary values, the query cost increases by $O(\log_B N)$.

We first consider the problem of reporting colors and their frequencies for a two-sided query in 2D, which is commonly referred to as dominance query [1] in 2D. We present a linear-space data structure that answers a constant-factor 2-sided approximate-frequency query: given a query range $[-\infty, \alpha] \times [-\infty, \beta]$, we report all colors in the query range, and for every reported color σ, we also provide an estimate k_σ on the number of its occurrences in the range. Let $\delta > 1$ be a constant that is fixed at the construction time and can be arbitrarily close to 1. For every color σ in the range, we obtain an integer index i, such that $\delta^i \leq k_\sigma \leq \delta^{i+1}$. The total cost of a query is $O(\log^2(N/B) + K/B)$ I/Os. Our data structure is based on a reduction to the problem of identifying all two-dimensional rectangles that contain a query point [4, 27] (rectangle stabbing problem), and uses $O(N)$ space. Then, we improve the time to $O(\log_B N + \log^* B + K/B)$ I/Os, still using linear space. As a straightforward corollary, we answer one-dimensional 2-sided queries with the same space-time trade-offs.

Additionally, we present an $O(N\lceil\frac{\log^2 D}{\log N}\rceil)$-word data structure that answers 1-dimensional 2-sided color reporting queries with exact frequencies in optimal $O(1+K/B)$ I/Os. We remark that the space is $O(N)$ words if the number of colors is not too large, i.e., when $D \le 2^{\sqrt{\log N}}$. This result is based on an $O(N)$-space data structure that answers 1-dimensional 1-sided color reporting queries with exact frequencies in optimal $O(1+K/B)$ I/Os.

Our results are presented in Table 1.

1.6 Map

We begin in Section 2 by reducing 2-sided approximate-frequency queries in 2D to rectangle stabbing queries (also known as orthogonal point enclosure). This leads to a simple linear space data structure. In Section 3, we improve this result to achieve our claimed I/O complexity without increasing the space usage. Section 4 explains how to use this data structure for answering approximate-frequency queries in one dimension. In Section 5, we present our data structure for reporting colors with exact frequencies in one dimension. Finally, we conclude in Section 6 with a couple of unanswered problems.

2 2D Color Dominance with Approximate Frequencies

In this section, we focus on answering two-dimensional colored dominance queries with approximate frequencies. Specifically, we consider the following:

▶ **Definition 1.** *Given a collection of two-dimensional colored points, we define* FREQ(α,β,σ) *to be the number of 2d points $p_i = (x_i, y_i)$ satisfying $x_i \le \alpha$, $y_i \le \beta$, and the color of p_i is σ.*

▶ **Problem 2.** *Given N colored points in 2d, answer an* APPX2D(α,β) *query: report all distinct colors $\sigma_1, \sigma_2, \ldots, \sigma_K$ with values $f_1, f_2, \ldots, f_K$ such that for each reported color σ_i,*

- FREQ$(\alpha,\beta,\sigma_i) \ne 0$, *and*
- FREQ$(\alpha,\beta,\sigma_i) \le f_i \le \delta \cdot$ FREQ(α,β,σ_i)

We prove the following theorem in this section.

▶ **Theorem 3.** *There exists an $O(N)$-word data structure that answers* APPX2D(α,β) *queries in $O(\log^2 \frac{N}{B} + K/B)$ I/Os.*

2.1 Shallow Cutting

We heavily rely on the concept of shallow cutting that has been used extensively in range reporting problems [1, 2, 17, 27, 28]. Consider a color σ and let N_σ be the total number of points of color σ. We call a point a σ-point if it is of color σ.

An h-shallow cutting for a parameter h can be constructed as follows. We start sweeping a vertical line V from $x = -\infty$ until we reach a σ-point such that the total number of σ-points encountered so far is δ^h. Now start sweeping a horizontal line H from $y = \infty$ until we reach the highest σ-point among these δ^h σ-points. Let c_1 be the point of intersection of the sweeping lines at this point. Let $\mathbb{R}_1^h$ be the region bounded by the sweeping lines; note that $\mathbb{R}_1^h$ is two-sided and has δ^h number of σ-points.

Now again start sweeping V to the right, until the region, say $\mathbb{R}_1^{h'}$, bounded by V and H contains δ^{h+1} points. Let c_1' be the point of intersection of the sweeping lines at this point. Start sweeping H downwards until the region, say $\mathbb{R}_2^h$, bounded by V and H again contains

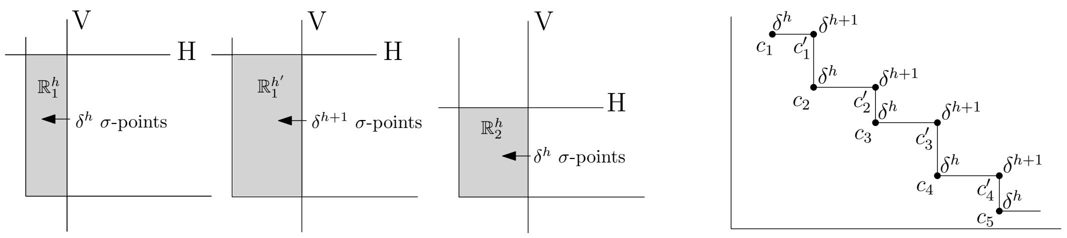
V
H
$\mathbb{R}_1^h$
δ^h σ-points
$\mathbb{R}_1^{h'}$
δ^{h+1} σ-points
$\mathbb{R}_2^h$
δ^h σ-points
c_1 c_1' c_2 c_2' c_3 c_3' c_4 c_4' c_5
δ^h δ^{h+1}

Figure 1 Illustration of Shallow Cutting and the resultant staircase structure.

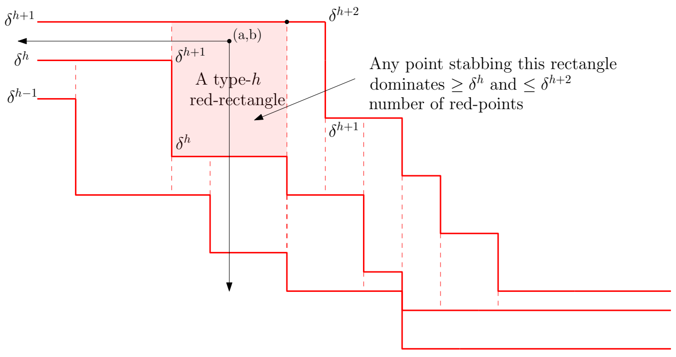
δ^{h+1}
δ^h
δ^{h-1}
δ^{h+2}
(a,b)
A type-h red-rectangle
Any point stabbing this rectangle dominates $\geq \delta^h$ and $\leq \delta^{h+2}$ number of red-points

Figure 2 Illustration of Rectangular Tessellation.

δ^h points. Let c_2 be the new point of intersection of V and H. We repeat this process until V reaches the rightmost σ-point within the region bounded by V and H, following which H is swept down so that the last two-sided region contains δ^h many σ-points.

We denote by $\mathbb{R}_1^h, \mathbb{R}_2^h, \ldots$, type-$h$ σ-regions; likewise, type-h regions refers to the collection of type-h σ-regions over all possible colors σ. The points $c_1, c_1', c_2, c_2', \ldots$ are referred to as the corner points of this "*staircase structure*". We also refer to this staircase structure as the skyline of the regions. See Figure 1.

The following lemma will play a crucial role in proving our space bounds.

▶ **Lemma 4.** *The number of type-h regions is $O(N/\delta^h)$.*

Proof. Refer to Figure 1. It is easy to see that every time a new region $\mathbb{R}_i^h$ is created from $\mathbb{R}_{i-1}^h$, we read $\delta^{h+1} - \delta^h$ many new σ-points. Hence, we create at most $\frac{N_\sigma}{(\delta^{h+1}-\delta^h)} = \frac{N_\sigma}{(\delta-1)\delta^h}$ σ-regions, which is $O(N_\sigma/\delta^h)$. Thus, the number of type-h σ-regions is $O(N_\sigma/\delta^h)$. The lemma follows trivially. ◀

2.2 Tessellation Using Rectangles

Let us consider the shallow cutting outlined in the previous section. For each color σ, we carry out the shallow cutting for every $h \in \left[1, \lceil \log_\delta N \rceil\right]$, thereby breaking the xy-plane into various σ-regions – type-1 σ-regions, type-2 σ-regions, $\cdots$, type-$(\log_\delta N)$ σ-regions.

We now present the following important lemma (see Figure 3).

▶ **Lemma 5.** *A type-h σ-region is contained within a type-$(h+1)$ σ-region.*

Proof. Refer to Figure 3. It shows the situation when a type-h σ-region is not contained in a type-$(h+1)$ σ-region. Since the number of σ-points dominated by both B and D is δ^{h+1}, there are no σ-points either to the left of the segment AD, or below the segment CD, or within the rectangle $ABCD$. But then our construction is incorrect and the points D and C must have coincided with A and B respectively. The claim follows by contradiction. ◀

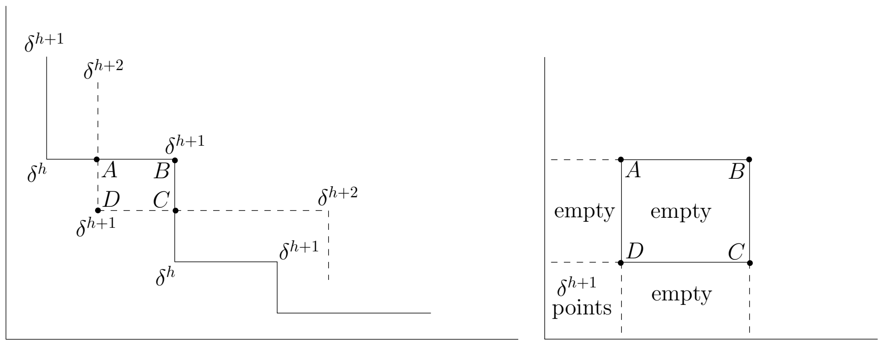

Figure 3 Illustration of Lemma 5. A type-h region is contained withing a type-$(h+1)$ region.

Based on the above lemma, the "*staircase structure*" of the $(h+1)$-shallow cutting lies to the right and above of the staircase structure for the h-shallow cutting.

Let us consider a point p in the plane such that it is contained in a type-$(h+1)$ σ-region but not contained in a type-h σ-region. This, as we shall see in more details in the next subsection, gives an estimate of the number of σ-points that are dominated by p. Our main idea is to associate such a point p with an approximate count of the σ-points that are dominated by it. Obviously, we cannot associate each point separately; hence, we break the xy-plane between a type-h and a type-$(h+1)$ region into several (yet appropriately bounded) rectangles. Specifically, consider the corner points of a type-h σ-region. We draw vertical lines from these points onto the skyline of the type-$(h-1)$ and type-$(h+1)$ σ-regions, thereby breaking the plane into several rectangles. See Figure 2.

We call each rectangle a type-h σ-rectangle if it is contained in a type-$(h+1)$ σ-region but not within a type-(h) σ-region. As before, type-h rectangles refers to the collection of type-h σ-rectangles over all possible colors σ. Using Lemma 4, we can show that the number of type-h rectangles for a fixed h is $O(N/\delta^h)$. The following lemma is obtained by summing over $h = 1, 2, \ldots, \lceil \log_\delta N \rceil$.

▶ **Lemma 6.** *The total number of rectangles is $O(N)$.*

2.3 Rectangle Stabbing

A rectangle is said to be stabbed by a point if it contains the point within its boundaries. The rectangle tessellation structure described in the previous section immediately leads to the following important lemma.

▶ **Lemma 7.** *If a type-h σ-rectangle is stabbed by (α, β), then the following inequality holds:*

$$\delta^h \leq \mathsf{FREQ}(\alpha, \beta, \sigma) \leq \delta^{h+2}$$

Proof. Consider the shallow cutting in Figure 2. Since the point (α, β) stabs a type-h σ-rectangle, it is contained within a type-$(h+1)$ σ-region. Thus, the number of σ-points dominated by any point on the skyline of a type-$(h+1)$ σ-region is at most δ^{h+2}. Hence, $\mathsf{FREQ}(\alpha, \beta, \sigma) \leq \delta^{h+2}$. To prove the other inequality, we observe in Figure 2 that any point within a type-$(h+1)$ σ-region dominates at most δ^h σ-points of a type-h σ-region. Hence, $\mathsf{FREQ}(\alpha, \beta, \sigma) \geq \delta^h$, as required. ◀

Intuitively, looking at the lemma above, it follows that in order to answer an $\mathsf{APPX2D}(\alpha, \beta)$ query, it suffices to report all the type-h σ-rectangles that are stabbed by (α, β) for all possible choices of σ and appropriate choices of h in each case. This is because if a type-h σ-rectangle is stabbed by a point (α, β), we have the following:

- there exists a σ-point dominated by (α, β), which follows simply from the way in which the rectangles are constructed, and
- $\delta^h \leq \mathsf{FREQ}(\alpha, \beta, \sigma) \leq \delta^{h+2}$, using Lemma 7

We notice that for each particular color there is at most one stabbed rectangle of that color. Therefore for each color, our task is to find the rectangle of that color (if any) that is stabbed by (α, β); for each stabbed rectangle, report its color and an appropriate real value which gives an approximation of $\mathsf{FREQ}(\alpha, \beta, \sigma)$.

To this end, we collect all the rectangles of all possible colors and store them in the following external-memory rectangle-stabbing data structure in [4].

▶ **Fact 8** (Theorem 4, [4]). *Given M rectangles, there exists an $O(M)$ space data structure that can report all rectangles stabbed by an input point in $O(\log^2(M/B) + occ/B)$ I/Os, where occ is the output size.*

With each type-h σ-rectangle, say R^h_σ, we store its corresponding color σ, and the value $\mathsf{COUNT}(R^h_\sigma) = \delta^{h+2}$.

Using Fact 8 and Lemma 6, the total space is $O(N)$ words.

2.4 Wrapping Up

To answer $\mathsf{APPX2D}(\alpha, \beta)$, we query the data structure of Fact 8 and report all the stabbed rectangles along with their color and $\mathsf{COUNT}(\cdot)$ values. The query time is $O(\log^2(N/B) + K/B)$, as desired.

Note that each color is reported exactly once, because only one rectangle of a particular color is stabbed; hence, we are left to show that for a reported type-h σ-rectangle R^h_σ, $\mathsf{COUNT}(R^h_\sigma)$ gives us a desired approximation of $\mathsf{FREQ}(\alpha, \beta, \sigma)$.

First we note that $\mathsf{COUNT}(R^h_\sigma) \geq \mathsf{FREQ}(\alpha, \beta, \sigma)$

Revisiting Lemma 7, we see that

$$\mathsf{FREQ}(\alpha, \beta, \sigma) \geq \delta^h = \frac{\mathsf{COUNT}(R^h_\sigma)}{\delta^2} \implies \mathsf{COUNT}(R^h_\sigma) \leq \delta^2 \cdot \mathsf{FREQ}(\alpha, \beta, \sigma)$$

By choosing $\sqrt{\delta}$ (instead of δ) during construction, we obtain

$$\mathsf{COUNT}(R^h_\sigma) \leq \delta \cdot \mathsf{FREQ}(\alpha, \beta, \sigma)$$

which leads to our desired approximation ratio. (Note that space and time bounds are only affected by a constant factor.) This completes the proof of Theorem 3.

3 Improvement via Bootstrapping

In this section, we set out to improve the query time of Theorem 3, still using linear space. More specifically, we prove the following theorem.

▶ **Theorem 9.** *There exists an $O(N)$-word data structure that answers $\mathsf{APPX2D}(\alpha, \beta)$ queries in $O(\log_B N + \log^* B + K/B)$ I/Os.*

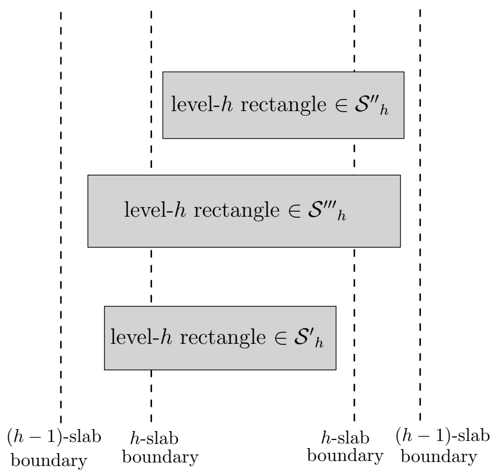
level-h rectangle $\in \mathcal{S}''_h$
level-h rectangle $\in \mathcal{S}'''_h$
level-h rectangle $\in \mathcal{S}'_h$
$(h-1)$-slab boundary
h-slab boundary
h-slab boundary
$(h-1)$-slab boundary

Figure 4 Illustration of type-h rectangles and the sets $\mathcal{S}'_h$, $\mathcal{S}''_h$, and S'''_h.

3.1 Framework and Overview

In order to prove Theorem 9, we need to improve the $\log^2(N/B)$ factor in the I/O complexity of Theorem 3. Unfortunately, as shown by Arge, Samoladas, and Yi [4], it is not possible to achieve $o(\log_2(N/B))$ query complexity for the rectangle stabbing problem in two dimensions if the data structure uses linear space. However, in our case, we are able to circumvent the lower bound of [4] because rectangles have some additional special properties. Our approach is based on a hierarchical subdivison of the plane into vertical slabs. The subdivision is based on two parameters:

$$\Delta_h = B(\log^{(h)}(N/B))^4 \text{ and } K_h = B(\log^{(h)}(N/B))^2$$

A slab boundary is a vertical line through $x = i + 0.5$ for some integer $i \in [1, N]$ and slab is the region between two slab-boundaries. The entire xy-plane is a level-0 slab. For $h = 1, 2, 3, \ldots, \log^* N$, we divide each level-$(h-1)$ slab into level h-slabs, so that

- the number of level-h slabs within any level-$(h-1)$ slab is $O(\Delta_{h-1}/\Delta_h)$
- the number of rectangles completely covered by any level-h slab is $O(\Delta_h)$.

A rectangle is a level-h rectangle if it crosses at least one boundary of a level-h slab, but is completely contained within a level-$(h-1)$ slab. See Figure 4.

Throughout this section we will denote by $\mathcal{S}_i$ the i-slab that contains the query point (α, β) for $i = 1, 2, \ldots, \log^* n$. Consider a particular h. Suppose that a point (α, β) stabs K rectangles, where $K \geq K_h$. All the level-h rectangles are contained in the $(h-1)$-slab $\mathcal{S}_{h-1}$. If we keep all level-h rectangles contained in $\mathcal{S}_{h-1}$ in the data structure of Fact 8, then all level-h rectangles stabbed by (α, β) are reported in $O(\log^2(\Delta_{h-1}/B) + K/B) = O(K_h/B + K/B) = O(K/B)$ I/Os. This case is considered in detail in Section 3.2.

The main difficulty is in reporting level-h rectangles for values of h, such that $K < K_h$. In this case, described in Section 3.3, we follow a different strategy. We keep all level-h rectangles that span the slab $\mathcal{S}_h$ (i.e., rectangles that intersect $\mathcal{S}_h$ but have all four corners outside of $\mathcal{S}_h$) in a separate data structure and use it to report all level-h rectangles that are stabbed by (α, β). Since (α, β) is within $\mathcal{S}_h$, we only need to look for rectangles whose y-projection contains β. In the general case, this approach would require too much space: a level-h rectangle can span a large number of h-slabs and we would have to keep it in a data structure for every slab. Fortunately in our case, rectangles satisfy a so-called *monotonic property*, defined in Observation 15. Due to this property, we only need to store $o(\Delta_h)$ "lowest" spanning rectangles for every h-slab. Any point above those "lowest" rectangles is guaranteed to stab at least K_h rectangles. Additionally, we also need to classify rectangles into heavy and light ones, according to the frequencies of regions represented by these rectangles. A detailed description is provided in Section 3.3.2. We consider separately the case of the level-h rectangles that intersect but do not span $\mathcal{S}_h$. Queries on such rectangles can be reduced to three-dimensional dominance queries; see Section 3.3.1.

In the description above, we assumed that the value of K is known. Computing the value of K is rather straightforward – simply report the colors of all points dominated by (α, β). Using the result of Patil et al. [25], we can report all colors (and compute K) in $O(\log^* N + K/B)$ I/Os using an $O(N)$ space data structure.

3.2 Handling Case 1: $K \geq K_h$

For $h = 1, 2, 3, \ldots$ and for each level-$(h-1)$ slab $\mathbb{D}$, we create a rectangle stabbing structure of Fact 8 over all level-h rectangles that are completely within $\mathbb{D}$. By our choice of parameters, the number of such rectangles is $O(\Delta_{h-1})$. By definition of level-h rectangles, every rectangle is stored in one data structure; hence, the total space is $O(N)$ words.

To report all level-h rectangles stabbed by (α, β), we first identify the level-$(h-1)$ slab $\mathbb{D}_{h-1}$ that is stabbed by the point (α, β) using the following lemma.

▶ **Lemma 10.** *For any $h \in [1, \log^* N]$, we can identify the level-$(h-1)$ slab that is stabbed by a point in $O(1)$ I/Os by using an $O(N)$ space data structure.*

Proof. For each h, maintain a bit vector $B_h[1, N]$, such that $B_h[i] = 1$ iff $i + 0.5$ is a level-h slab boundary. Additionally, associate an $o(N)$-bit structure for constant rank/select support on B_h [26]. Clearly, the level-h slab stabbed by (α, β) is the one that starts at $t + 0.5$, where t is the rightmost position $\leq \beta$, such that $B_h[t] = 1$. The number of I/Os required is $O(1)$. The total space is $O(N \log^* N)$ bits, which is $O(N)$ words. ◀

Now, we consider a query over the rectangle stabbing structure associated with $\mathbb{D}_{h-1}$. Suppose the number of rectangles reported is occ_{h-1}. Then, the number of I/Os required is

$$O(\log^2(\Delta_{h-1}/B) + occ_{h-1}/B) \text{ i.e., } O((K_h + occ_{h-1})/B)$$

Therefore the total number of I/Os over all values of h, where $K \geq K_h$, is

$$O\left(\frac{1}{B} \sum_{h, K \geq K_h} (K_h + occ_{h-1})\right) \text{ i.e., } O(\log^* N + K/B)$$

In summary, we have the following lemma.

▶ **Lemma 11.** *If $K_h \leq K$, then all level-h rectangles that are stabbed by a point (α, β) can be retrieved in $O(\log^* N + K/B)$ I/Os by using an $O(N)$ space data structure.*

3.3 Handling Case 2: $K < K_h$

Let $\mathcal{S}_h$ be the set of all level-h rectangles intersecting with a level-h slab, say $\mathbb{D}_h$. Note that by definition, they all are completely within a level-$(h-1)$ slab, say $\mathbb{D}_{h-1}$, hence $|\mathcal{S}_h| = O(\Delta_{h-1})$. We divide $\mathcal{S}_h$ into three disjoint sets, $\mathcal{S}_h = \mathcal{S}'_h \cup \mathcal{S}''_h \cup \mathcal{S}'''_h$, defined as follows.

1. $\mathcal{S}'_h$: rectangles in $\mathcal{S}_h$ with both right corners inside $\mathbb{D}_h$ and both left corners outside $\mathbb{D}_h$.
2. $\mathcal{S}''_h$: rectangles in $\mathcal{S}_h$ with both left corners inside $\mathbb{D}_h$ and both right corners outside $\mathbb{D}_h$.
3. $\mathcal{S}'''_h$: rectangles in $\mathcal{S}_h$ with all 4 corners outside $\mathbb{D}_h$.

See Figure 4. We will answer stabbing queries for every subset of $\mathcal{S}_h$ separately.

3.3.1 Handling $\mathcal{S}'_h$ and $\mathcal{S}''_h$

We show how to find the rectangles that belong to $\mathcal{S}'_h$ and are stabbed by (α, β); the rectangles in $\mathcal{S}''_h$ can be found in a symmetric way.

Observe that since the right corners of any rectangle in $\mathcal{S}'_h$ are within the slab, a rectangle $[x_l, x_r] \times [y_b, y_t]$ is stabbed when $\alpha \leq x_r$ and $y_b \leq \beta \leq y_t$. Based on this intuition, we recall the following three-dimensional dominance reporting structure of Afshani [1].

▶ **Fact 12.** *Given M three-dimensional points, there exists an $O(M)$ space data structure that can report all the occ points dominated by a query point q in $O(\log_B M + occ/B)$ I/Os.*

Using the above result, we prove the following lemma.

▶ **Lemma 13.** *We can report all rectangles from sets $\mathcal{S}'_h$ and $\mathcal{S}''_h$, $1 \leq h \leq \log^* N$, stabbed by a query point (α, β) in $O(\log^* N + \log_B N + K/B)$ I/Os using an $O(N)$ space data structure.*

Proof. Based on the data structure in Fact 12, we represent a rectangle $R = [x_l, x_r] \times [y_b, y_t]$ in $\mathcal{S}'_h$ as a three-dimensional point $(-x_r, y_b, -y_t)$. The rectangle is stabbed by (α, β) iff $(-\alpha, \beta, -\beta)$ dominates $(-x_r, y_b, -y_t)$. For each $h \in [1, \log^* N]$, we maintain two dominance structures – one for the rectangles in $\mathcal{S}'_h$ and another for the rectangles in $\mathcal{S}''_h$. The total space over all levels is $O(N)$. Since $|\mathcal{S}'_h \cup \mathcal{S}''_h| = O(\Delta_h)$, if the number of rectangles reported at level h is occ_h, the number of I/Os is

$$O\left(\log_B(\Delta_h) + \frac{occ_h}{B}\right) \text{ i.e., } O\left(1 + \log_B(\log^{(h)} N) + \frac{occ_h}{B}\right)$$

The total number of I/Os for $h = 1, 2, \ldots, \log^* N$ is $O(\log^* N + \log_B N + \frac{K}{B})$. ◀

3.3.2 Handling $\mathcal{S}'''_h$

What remains is to show how to obtain the rectangles in $\mathcal{S}'''_h$ stabbed by the query. The idea is to divide $\mathcal{S}'''_h$ into two carefully selected subsets:

- a set $\mathcal{H}_h$ of heavy rectangles
- a set $\mathcal{L}_h$ of light rectangles

Heavy Rectangles. The set $\mathcal{H}_h$ contains all rectangles R in $\mathcal{S}'''_h$ with $\mathsf{COUNT}(R) \geq \dfrac{\Delta_{h-1}}{B}$, where $\mathsf{COUNT}(R)$ is the frequency associated with R. We have the following lemma.

▶ **Lemma 14.** *For any $h \in [1, \log^* N]$, the number of heavy rectangles in $\mathcal{H}_h$ is $O(B)$.*

Proof. Consider a level-h σ-rectangle $R = [x', x''] \times [y', y'']$. Let $\mathsf{COUNT}(R) = f$. Then, the number of σ-points in $[x', x'']$ is $\Theta(f)$, which follows from our shallow cutting construction. Therefore within a level-$(h-1)$ slab $\mathbb{D}$, the number of points is Δ_{h-1} and the number of level-h heavy rectangles completely within $\mathbb{D}$ is bounded by $\Delta_{h-1}/f = O(B)$. ◀

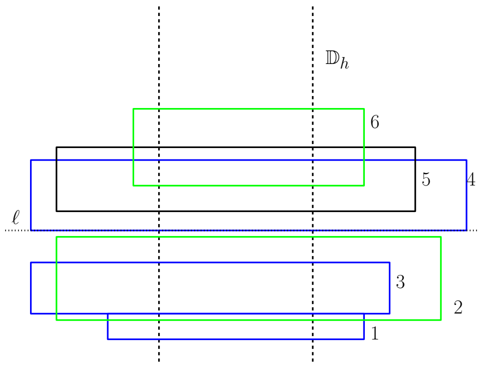

Figure 5 Example of $\mathcal{L}_h$ for $K_h = 3$. Rectangles spanning $\mathbb{D}_h$ with bottom segment below ℓ (i.e., 1, 2, 3, and 4) are included in $\mathcal{L}_h$. Every point within $\mathbb{D}_h$ and above ℓ stabs at least K_h many rectangles.

We can store these rectangles in space $\sum_h \left(\frac{N}{\Delta_h} \cdot B\right) = O(N)$. When a query is answered, we load all the rectangles in $\mathcal{H}_h$ into the internal memory using $O(1)$ I/Os, and then examine them in the internal memory to find the ones stabbed by (α, β). Hence, the total number of I/Os over all h is bounded by $O(\log^* N)$.

Light Rectangles. To create the set $\mathcal{L}_h \subseteq S_h'''$ of light rectangles, we sweep a horizontal line ℓ from $-\infty$ in the positive y-direction. Every time when ℓ crosses the bottom line of a rectangle from S_h''', we add its color to the set of colors, say $\mathcal{C}_h$. We stop when $\mathcal{C}_h$ contains K_h distinct colors (or when ℓ reaches the highest rectangle in S_h'''). See Figure 5. Then $\mathcal{L}_h$ is the set of all rectangles R in S_h''' with color $c \in \mathcal{C}_h$ and the $\mathsf{COUNT}(R) < \Delta_{h-1}/B$. Recall that $\mathsf{COUNT}(R)$ is the frequency associated with a rectangle R, and they are of the form $\delta, \delta^2, \ldots$. Therefore, the number of rectangles having a color from $\mathcal{C}_h$, which is associated with a particular $\mathsf{COUNT}(\cdot)$ is $\lceil \log_\delta(\Delta_{h-1}/B) \rceil$. Hence, the size of $\mathcal{L}_h$ is $|\mathcal{L}_h| = K_h \cdot \log_\delta(\Delta_{h-1}/B) = O(B(\log^{(h)} n)^3)$.

Since every rectangle in $\mathcal{L}_h$ spans $\mathbb{D}_h$, for our purposes they can be represented by their projections on the y-axis, i.e., x-coordinates can be ignored. To check whether a rectangle $[x_l, x_r] \times [y_b, y_t]$ in $\mathcal{L}_h$ is stabbed by (α, β), it suffices to check whether $y_b \le \beta \le y_t$. We can find all $[y_b, y_t]$ stabbed by β in $O(\log_B |\mathcal{L}_h|)$ I/Os [5].

Hence, all rectangles in $\mathcal{L}_h$ that are stabbed by (α, β) can be obtained in

$$O\left(\log_B |\mathcal{L}_h| + \frac{occ_h}{B}\right) = O\left(1 + \log_B(\log^{(h)} n) + \frac{occ_h}{B}\right)$$

I/Os, where occ_h is the number of such rectangles. Therefore the total number of I/Os summed over $h \in [1, \log^* N]$ is $O(\log^* N + \log_B N + K/B)$ as desired.

The total space needed to store all sets $\mathcal{L}_h$ is

$$O\left(\sum_h \frac{nK_h}{\Delta_h} \cdot \log \frac{\Delta_{h-1}}{B}\right) = O\left(\sum_h \frac{N}{\log^{(h)} \frac{N}{B}}\right) = O(N)$$

Now we will show that it is sufficient to examine rectangles in $\mathcal{L}_h$ if $K < K_h$.

▶ **Observation 15** (Monotonic Stabbing). *If a point (x, y) stabs a rectangle with color c, then any point (x, y'), where $y' > y$, stabs a rectangle with color c. Hence, the output (colors) of a stabbing query (x, y) is a subset of the output of a stabbing query (x, y') for $y' > y$.*

Let y_h denote the final y-coordinate of ℓ (i.e., the position of ℓ when $\mathcal{C}_h$ contains K_h colors, see Fig. 5). If a rectangle $R \in S_h'''$ stabs a point $(x, y) \in \mathbb{D}_h$ and $y \le y_h$, then $R \in \mathcal{L}_h$. Consider an arbitrary point $(x, y) \in \mathbb{D}_h$ with $y \ge y_h$. For every color in $c \in \mathcal{C}_h$, there is a point $(x, y') \in \mathbb{D}_h$ such that $y' \le y_h$ and (x, y') stabs a rectangle of color c. Hence, by Observation 15, (x, y) also stabs a rectangle of color c. Therefore any point $(x, y) \in \mathbb{D}_h$, such that $y \ge y_h$, stabs at least K_h different rectangles. We obtain the following lemma.

▶ **Lemma 16.** *We can report all rectangles from sets S_h''', $1 \le h \le \log^* N$, that are stabbed by a query point (α, β) in $O(\log^* N + \log_B N + K/B)$ I/Os using an $O(N)$ space data structure.*

3.4 Wrapping Up

Combining Lemmas 11, 13 and 16, the total space is $O(N)$ words and the total number of I/Os needed is $O(\log^* N + \log_B N + K/B)$, which is $O(\log^* B + \log_B N + K/B)$.[3] This completes the proof of Theorem 9.

4 1D Approximate Color Reporting

We now consider the following query: *reporting colors in one dimension with their approximate frequencies*, i.e., we report all the colors within the query range $[\alpha, \beta]$; however, instead of reporting the exact frequency f_{exact} for a color σ that lies in the query range, we report a real value f_σ such that $f_{exact} \le f_\sigma \le \delta \cdot f_{exact}$, where $\delta > 1$ is an arbitrary small constant. We reduce these queries to APPX2D(α, β) queries in 2D discussed in Sections 2 and 3: represent a 1D point x as $(-x, x)$ in 2D; an approximate color frequency query in one dimension, denoted by APPX1D(α, β), is answered using the query APPX2D$(-\alpha, \beta)$. Hence, the following is an immediate consequence of Theorem 9.

▶ **Theorem 17.** *Given N one-dimensional colored points, we can answer an* APPX1D(α, β) *query in $O(\log^* B + \log_B N + K/B)$ I/Os using an $O(N)$ space data structure.*

5 1D Exact Color Reporting

In this section, we answer one-dimensional color reporting queries with exact frequencies.

▶ **Problem 18.** *Given N one-dimensional colored points, answer an* EXACT1D(α, β) *query defined as: report all distinct colors $\sigma_1, \sigma_2, \ldots, \sigma_K$, with associated values* FREQ$(\alpha, \beta, \sigma_1)$, FREQ$(\alpha, \beta, \sigma_2)$, *…,* FREQ$(\alpha, \beta, \sigma_K)$*, such that for each σ_i, there exists a σ_i-point x_i satisfying $\alpha \le x_i \le \beta$. Here,* FREQ$(\alpha, \beta, \sigma_i)$ *is the number of σ_i-points in $[\alpha, \beta]$.*

First, we describe a data structure for storing partially persistent lists, which will be heavily used for answering these queries. Then, we present a data structure that answers one-sided queries, i.e., either $\alpha = 1$ or $\beta = N$. This result is based on storing selected points and their colors in a persistent list; we achieve optimal space usage and query cost for this special case. Then, we show how the same data structure can be used to answer general queries; however the query cost is proportional to the total number of colors. Next, we show how the optimal query cost for two-sided queries can be achieved by a data structure that needs $O(N \log D)$ space. We then show how to reduce the space usage by presenting an $O(N\lceil \frac{\log^2 D}{\log N} \rceil)$-space data structure that answers an EXACT1D(α, β) query in optimal $O(1 + K/B)$ I/Os; see Theorem 23.

[3] If $\log N > B$, then $\frac{\log N}{\log B} > \frac{\log N}{\log \log N} > \log^* N > \log^* B$. If $\log N < B$, then $\log^* N = O(\log^* B)$.

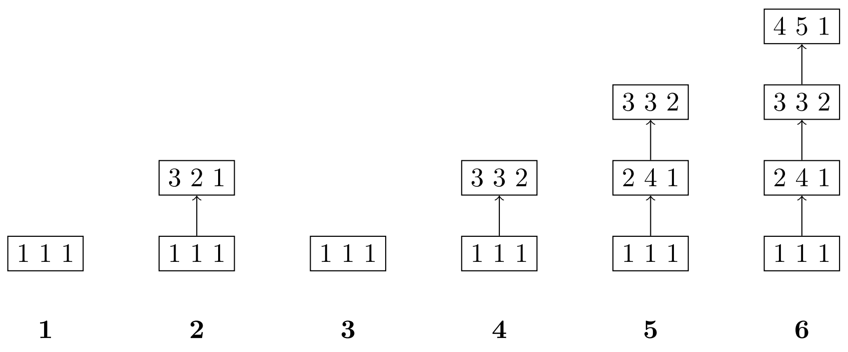

Figure 6 An example of a persistent list L for a set of five points such that $\mathsf{color}(1) = 1$, $\mathsf{color}(2) = 3$, $\mathsf{color}(3) = 3$, $\mathsf{color}(4) = 2$, and $\mathsf{color}(5) = 4$. Each version of the list is shown separately. Every entry of L contains a color σ, a point x, and its rank $\mathsf{rank}(x)$ (in this order). Versions $1, 2, 4, 5, 6$ correspond to points $1, 2, 3, 4, 5$. We remark that the total space used by L is $6 \cdot \text{const}$.

5.1 Partially Persistent Lists

Our solution makes use of the data structure for *offline partially persistent linked list* problem. In this problem, we maintain a linked list L under insertions and deletions: an arbitrary list entry can be deleted and a new entry can be inserted at any position of the list. We assume that L is originally empty and the sequence of updates is known in advance. The t-th update creates a new version of the list with timestamp t.

We can traverse the list (or a prefix of the list) for any timestamp t: given t, we can report all list entries that were in the list at time t. Also, given t and a value a, we can report all list entries that are smaller than a and that were in the list at time t. Such queries can be supported in $O(1 + K/B)$ I/Os, where K is the number of reported entries. The space needed to store a persistent list is $O(r)$ words or $O(r \log r)$ bits, where r is the total number of updates. The persistent list can be implemented using e.g., the partially persistent B-tree [6]; we refer to [16], Section 4, for a more detailed description.

5.2 1-sided Queries

A *one-sided* query is of the form $[1, \beta]$ or $[\alpha, N]$. Define the rank of a point p as the number of points that are smaller than or equal to p and have the same color:

$$\mathsf{rank}(p) = |\{p' \mid p' \leq p \text{ and } \mathsf{color}(p') = \mathsf{color}(p)\}|$$

We use a persistent list L in order to answer one-sided queries. (See Figure 6.) Every entry of L stores a color and entries are sorted by their colors in increasing order. An entry of L also contains some point p and its rank, $\mathsf{rank}(p)$. The list L is organized as follows. Originally L is empty. All points from a set X are traversed in the left-to-right order. When a point x is reached, we update the list L as follows. If L already contains an entry e for $\mathsf{color}(x)$, we remove e from L. Then we insert a new entry e' of color $\mathsf{color}(x)$ into an appropriate position (i.e., in sorted order) in L. The entry e' contains the point x and $\mathsf{rank}(x)$. Thus, the list L is updated one or two times for every point in X and the total number of entries in L is $O(N)$. In addition to L, we keep a table $V[1 \ldots N]$; its i-th entry $V[i]$ contains the index of the version of L after the point i was inserted.

Suppose that we want to report all colors that occur in a range $[1, \alpha]$ and output the number of occurrences for each color. We traverse the j-th version L_j of L where $j = V[\alpha]$. For every entry e of L_j, we report its color and the rank of the point p stored in e.

This brings us to the following result.

▶ **Theorem 19.** *There exists an $O(N)$ space data structure, using which we can answer* EXACT1D$(1, \beta)$ *in* $O(1 + \frac{K}{B})$ *I/Os.*

We can use the same persistent list L in order to answer two-sided queries. Unfortunately the query cost is significantly higher.

▶ **Lemma 20.** *There exists an $O(N)$-space data structure, using which we can answer* EXACT1D(α, β) *in* $O(1 + D/B)$ *I/Os.*

Proof. We use the persistent list L and the array V defined earlier. Suppose we want to answer a query on a range $[\alpha, \beta]$. We identify the versions f and l of L that correspond to $\alpha - 1$ and β, i.e., $f = V[\alpha - 1]$ and $l = V[\beta]$. By definition, both L_f and L_l contain exactly one entry for every color σ that occurs in $[1, \alpha - 1]$ and $[1, \beta]$ respectively. Let e_f and e_l denote entries of color σ in L_f and L_l respectively. If e_f and e_l contain the same point x, then $x < \alpha$ and σ does not occur in $[\alpha, \beta]$. If e_f and e_l contain two different points, p_f and p_l respectively, then σ occurs $\mathsf{rank}(p_l) - \mathsf{rank}(p_f)$ times in $[\alpha, \beta]$. Finally if there is no entry e_f in L_f, then the leftmost occurrence of σ lies in $[\alpha, \beta]$. In this case, σ occurs $\mathsf{rank}(p_l)$ times.

The values of $\mathsf{rank}(p_l)$ and $\mathsf{rank}(p_f)$ for all relevant colors can be computed as follows. We simultaneously generate versions L_f and L_l of L and merge them into a new (non-persistent) list T. Entries of T are sorted by color and T contains at most two entries of the same color. When T is available, we can retrieve values of p_l and p_f (resp. the value of p_l is e_f does not exist) for at least $B/2$ colors σ with one I/O operation. Hence, we can compute the number of occurrences for at least $B/2$ colors with $O(1)$ I/Os. The total cost of our procedure is $O(q/B)$, where q denotes the number of entries in T. Since $q \le D$, our algorithm answers an EXACT1D(α, β) query in $O(1 + D/B)$ I/Os. ◀

5.3 Answering in Optimal I/Os

We use the data structure of Lemma 20 as a building block. For $i = 1, 2, \ldots, \log(D/B)$, we divide the set of points into i-chunks so that each chunk, except the last one, contains $B \cdot 2^i$ distinct colors. The last chunk contains at least $B \cdot 2^i$ and at most $B \cdot 2^{i+1}$ distinct colors. For any particular i, we use $C_{i,j}$ to denote the j^{th} i-chunk. For every i-chunk, we keep the data structure of Lemma 20 that answers queries in $O(2^i)$ I/Os. We also store the data structure of Lemma 20 for every two consecutive i-chunks $C_{i,j} \cup C_{i,j+1}$.

Consider a query $[\alpha, \beta]$. We find the smallest i such that there is no i-chunk $C_{i,j}$ that is entirely contained in $[\alpha, \beta]$. Then, the interval $[\alpha, \beta]$ intersects at most two i-chunks. Suppose that $[\alpha, \beta]$ is contained in one chunk $C_{i,j}$. Then we report all colors in $[\alpha, \beta]$ and the number of their occurrences in $O((B \cdot 2^i)/B)$ I/Os using the result of Lemma 20. If $i = 1$, the query cost is $O((B \cdot 2)/B) = O(1)$ I/Os. If $i > 1$, the interval $[\alpha, \beta]$ contains at least one $(i-1)$-chunk and the number of distinct colors in $[\alpha, \beta]$ is $K \ge B \cdot 2^{i-1}$. Hence the query cost is $O(2^i) = O(K/B)$ I/Os. Now suppose that $[\alpha, \beta]$ intersects two chunks, $C_{i,j}$ and $C_{i,j+1}$. In this case, we answer a query using the data structure for the chunk pair $C_{i,j} \cup C_{i,j+1}$. By the same argument as above, the total query cost is $O(K/B)$ I/Os.

It remains to show how we can find the desired minimum i and the chunks $C_{i,j}$ and $C_{i,j+1}$. For $i = 1, 2, \ldots, \log^* n$, we keep boundaries of i-chunks in a bit vector of size $O(n)$. Using B_i and the data structure from [26], we can find the i-chunks that contain α and β in $O(1)$ time. In order to find, $C_{i,j}$ and $C_{i,j+1}$ we proceed as follows. Using B_r we find r-chunks C_{r,f_r} and C_{r,l_r} that contain α and β respectively for $r = 1, 2, \ldots, i$; we stop when $f_i = l_i$ or $f_i + 1 = l_i$. The total cost of finding the chunks is $O(i) = O(K/B + 1)$.

We get the following result.

▶ **Theorem 21.** *There exists an $O(N \log D)$-space data structure that answers a query* $\mathsf{EXACT1D}(\alpha, \beta)$ *in optimal $O(1 + K/B)$ I/Os.*

5.4 Reducing Space Usage

▶ **Lemma 22.** *Consider a subset of the input points containing d distinct colors. If α and β lie within the boundaries of this subset, by using an $O\big(N(\log D + \frac{\log N}{D})\big)$-bit data structure, we can answer an* $\mathsf{EXACT1D}(\alpha, \beta)$ *query in $O(1 + d/B)$ I/Os.*

Proof. Let P' be the subset of the set of points. We divide the set of points into sub-chunks C_i of size D^2 each (except for the last sub-chunk that can contain up to $2D^2$ points). If P' contain less than $2D^2$ points, all points are in the same sub-chunk. For every sub-chunk we keep the data structure of Lemma 20. Since each sub-chunk contains D^2 points and the color of every point can be specified with $\log D$ bits, the space usage of a sub-chunk data structure is $O(D^2 \log D)$ bits. If there is more than one sub-chunk, we keep a global data structure G that contains $O(N/D)$ elements. We implement G as a persistent list that is similar to the structure of Lemma 20. However our modified list contains at most D elements for each sub-chunk. We traverse sub-chunks of P' in the left-to-right order. For each sub-chunk C_i, we perform $O(D)$ updates on G. For every color σ that occurred in C_i, the following updates are performed. If G already contains an entry e_o of color σ we remove e_o from G. Then, we insert a new entry e of color σ into G; e contains the rightmost point x_σ of color σ in C_i and the rank of x_σ, $\mathsf{rank}(x_\sigma)$.

Consider a query $[\alpha', \beta']$ such that α' and β' are boundaries of sub-chunks C_f and C_l. Let $i_\alpha = V_g[f-1]$ and $i_\beta = V_g[l]$ denote the versions of G that correspond to sub-chunks C_{f-1} and C_l respectively. We can answer a query on $[\alpha', \beta']$ by essentially the same method that is used in Lemma 20. We generate versions G_{i_α} and G_{i_β} and merge them into a list T_G. Since G_{i_α} contain at most one entry of every color, T_G contains at most two entries of the same color. Let $p_f(\sigma)$ denote the rightmost occurrence of σ in $[1, \alpha' - 1]$ (i.e., $p_f(\sigma)$ is the rightmost occurrence of σ in the sub-chunk C_{f-1} or in some sub-chunk to the left of C_{f-1}) and $p_l(\sigma)$ denotes the rightmost occurrence of σ in $[1, \beta']$ (i.e., $p_l(\sigma)$ is the rightmost occurrence of σ in sub-chunks $C_1, C_2, \ldots, C_l$). The color σ occurs $\mathsf{rank}(p_l(\sigma)) - \mathsf{rank}(p_f(\sigma))$ times in $[\alpha', \beta']$ (or $\mathsf{rank}(p_l(\sigma))$ times if $p_f(\sigma)$ does not exist). We can retrieve at least $B/2$ values of $\mathsf{rank}(p_f(\sigma))$ and $\mathsf{rank}(p_l(\sigma))$ with one I/O operation. The total cost of a query is $O(1 + d/B)$ because the list T_G contains at most $2d$ entries.

Now we consider an $\mathsf{EXACT1D}(\alpha, \beta)$ query restricted to the boundaries of P'. Suppose that $\alpha \in C_f$ and $\beta \in C_l$. If α and β are in the same sub-chunk, we use the data structure for that sub-chunk to answer the query. Otherwise we decompose $[\alpha, \beta]$ into at most three parts, $[\alpha, \beta] = [\alpha, \alpha' - 1] \cup [\alpha', \beta'] \cup [\beta' + 1, \beta]$ so that $[\alpha', \beta'] = \cup_{i=f+1}^{l-1} C_i$. The intervals $[\alpha, \alpha' - 1]$ and $[\beta' + 1, \beta]$ are in sub-chunks C_f and C_l respectively. We answer extended color reporting queries on $[\alpha, \alpha' - 1]$, $[\alpha', \beta']$, and $[\beta' + 1, \beta]$. An answer to each query is stored in a list of colors T_i, $1 \le i \le 3$. The entries of T_i are sorted by color in increasing order. For an entry of color σ in a list T_i, we store the frequency of σ in the corresponding interval. We merge lists T_i into a global list T in $O(1 + d/B)$ I/Os. Then we traverse the resulting list and obtain the frequencies of all colors in $[\alpha, \beta]$ in $O(1 + d/B)$ I/Os.

The persistent list G uses $O(N\lceil \frac{\log N}{D} \rceil)$ bits of space. All chunk data structures require $O(N \log D)$ bits. Hence the total space is $O\big(N(\log D + \frac{\log N}{D})\big)$ bits. ◀

▶ **Theorem 23.** *There exists an* $O(N\lceil\frac{\log^2 D}{\log N}\rceil)$*-space data structure that answers an* EXACT1D(α,β) *query in optimal* $O(1+K/B)$ *I/Os.*

Proof. We divide the set of points into chunks as described in Theorem 21, and then use Lemma 22 for each chunk. Since each point is contained in $\log D$ chunks, the total space occupied is $O(N(\log D+\lceil\frac{\log N}{D}\rceil)\log D) = O(N(\log^2 D+\log N))$ bits or $O(N\lceil\frac{\log^2 D}{\log N}\rceil)$ words. A query is answered in $O(1+K/B)$ I/Os as explained in the proof of Theorem 21. ◀

6 Conclusion

We leave the following questions unanswered. First, *can we obtain a data structure that answers one-dimensional color reporting queries with approximate frequencies in optimal* $O(1+K/B)$ *I/Os?* Although such a data structure with $O(N\log^* N)$ space seems feasible, techniques do not seem to generalize to the more general case of two-dimensional dominance queries. Second, an interesting question is whether the approximation factor δ can be provided during query time, as opposed to the current approach of having to provide it during construction time. Additionally, we leave the question of reporting colors with exact frequency in case of 2D dominance unanswered. Finally, it would be interesting if one could generalize 2D dominance to the more general 3-sided/4-sided queries.

References

1 Peyman Afshani. On Dominance Reporting in 3D. In *ESA*, pages 41–51, 2008.

2 Pankaj K. Agarwal, Alon Efrat, and Micha Sharir. Vertical Decomposition of Shallow Levels in 3-Dimensional Arrangements and Its Applications. *SIAM J. Comput.*, 29(3):912–953, 1999. doi:10.1137/S0097539795295936.

3 Lars Arge, Vasilis Samoladas, and Jeffrey Scott Vitter. On Two-Dimensional Indexability and Optimal Range Search Indexing. In *Proc. 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 346–357, 1999. doi:10.1145/303976.304010.

4 Lars Arge, Vasilis Samoladas, and Ke Yi. Optimal External Memory Planar Point Enclosure. *Algorithmica*, 54(3):337–352, 2009. doi:10.1007/s00453-007-9126-2.

5 Lars Arge and Jeffrey Scott Vitter. Optimal External Memory Interval Management. *SIAM J. Comput.*, 32(6):1488–1508, 2003. doi:10.1137/S009753970240481X.

6 Bruno Becker, Stephan Gschwind, Thomas Ohler, Bernhard Seeger, and Peter Widmayer. An Asymptotically Optimal Multiversion B-Tree. *VLDB J.*, 5(4):264–275, 1996. doi:10.1007/s007780050028.

7 Panayiotis Bozanis, Nectarios Kitsios, Christos Makris, and Athanasios K. Tsakalidis. New Upper Bounds for Generalized Intersection Searching Problems. In *Proc. 22nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 464–474, 1995. doi:10.1007/3-540-60084-1_97.

8 Panayiotis Bozanis, Nectarios Kitsios, Christos Makris, and Athanasios K. Tsakalidis. New Results on Intersection Query Problems. *Comput. J.*, 40(1):22–29, 1997. doi:10.1093/comjnl/40.1.22.

9 Timothy M. Chan and Bryan T. Wilkinson. Adaptive and Approximate Orthogonal Range Counting. In *Proc. 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 241–251, 2013. doi:10.1137/1.9781611973105.18.

10 Arnab Ganguly, Wing-Kai Hon, and Rahul Shah. Stabbing Colors in One Dimension. In *2017 Data Compression Conference, DCC 2017, Snowbird, UT, USA, April 4-7, 2017*, pages 280–289, 2017. doi:10.1109/DCC.2017.44.

11 Prosenjit Gupta, Ravi Janardan, and Michiel H. M. Smid. Further Results on Generalized Intersection Searching Problems: counting, Reporting, and Dynamization. *Journal of Algorithms*, 19(2):282–317, 1995. doi:10.1006/jagm.1995.1038.

12 Prosenjit Gupta, Ravi Janardan, and Michiel H. M. Smid. Algorithms for Generalized Halfspace Range Searching and Other Intersection Searching Problems. *Comput. Geom.*, 6:1–19, 1996. doi:10.1016/0925-7721(95)00012-7.

13 Ravi Janardan and Mario A. Lopez. Generalized intersection searching problems. *International Journal of Computational Geometry and Applications*, 3(1):39–69, 1993.

14 Marek Karpinski and Yakov Nekrich. Top-K Color Queries for Document Retrieval. In *Proc. 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 401–411, 2011. URL: http://www.siam.org/proceedings/soda/2011/SODA11_032_karpinskim.pdf.

15 Kasper Green Larsen and Rasmus Pagh. I/O-efficient data structures for colored range and prefix reporting. In *Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 583–592, 2012. URL: http://portal.acm.org/citation.cfm?id=2095165{&}CFID=63838676{&}CFTOKEN=79617016.

16 Kasper Green Larsen and Freek van Walderveen. Near-Optimal Range Reporting Structures for Categorical Data. In *Proc. 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 256–276, 2013.

17 Jiří Matoušek. Reporting Points in Halfspaces. In *Proc. 32nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 207–215, 1991. doi:10.1109/SFCS.1991.185370.

18 S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 657–666, 2002. doi:10.1145/545381.545469.

19 Alexandros Nanopoulos and Panayiotis Bozanis. Categorical Range Queires in Large Databases. In *Proc. 8th International Symposium on Advances in Spatial and Temporal Databases, (SSTD)*, pages 122–139, 2003. doi:10.1007/978-3-540-45072-6_8.

20 Gonzalo Navarro and Yakov Nekrich. Top-k document retrieval in optimal time and linear space. In *Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1066–1077, 2012. URL: http://portal.acm.org/citation.cfm?id=2095200{&}CFID=63838676{&}CFTOKEN=79617016.

21 Yakov Nekrich. External Memory Range Reporting on a Grid. In *Proc. 18th International Symposium on Algorithms and Computation (ISAAC)*, pages 525–535, 2007. doi:10.1007/978-3-540-77120-3_46.

22 Yakov Nekrich. Data Structures for Approximate Orthogonal Range Counting. In *Proc. 20th International Symposium on Algorithms and Computation (ISAAC)*, pages 183–192, 2009. doi:10.1007/978-3-642-10631-6_20.

23 Yakov Nekrich. Efficient range searching for categorical and plain data. *ACM Trans. Database Syst.*, 39(1):9, 2014. doi:10.1145/2543924.

24 Yakov Nekrich and Jeffrey Scott Vitter. Optimal Color Range Reporting in One Dimension. In *Proc. 21st Annual European Symposium Algorithms (ESA)*, pages 743–754, 2013. doi:10.1007/978-3-642-40450-4_63.

25 Manish Patil, Sharma V. Thankachan, Rahul Shah, Yakov Nekrich, and Jeffrey Scott Vitter. Categorical range maxima queries. In *Proc. 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 266–277, 2014. doi:10.1145/2594538.2594557.

26 Mihai Patrascu. Succincter. In *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 305–313, 2008. doi:10.1109/FOCS.2008.83.

27 Saladi Rahul. Improved Bounds for Orthogonal Point Enclosure Query and Point Location in Orthogonal Subdivisions in $\mathbb{R}^3$. In *Proc.26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 200–211, 2015. doi:10.1137/1.9781611973730.15.

28 Saladi Rahul. Approximate Range Counting Revisited. In *Proc. 33rd International Symposium on Computational Geometry (SoCG)*, pages 55:1–55:15, 2017. `doi:10.4230/LIPIcs.SoCG.2017.55`.

29 Kunihiko Sadakane. Succinct data structures for flexible text retrieval systems. *J. Discrete Algorithms*, 5(1):12–22, 2007. `doi:10.1016/j.jda.2006.03.011`.