Dynamic Colored Orthogonal Range Searching

Timothy M. Chan
Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA

Zhengcheng Huang
Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA

Abstract

In the *colored orthogonal range reporting* problem, we want a data structure for storing n colored points so that given a query axis-aligned rectangle, we can report the distinct colors among the points inside the rectangle. This natural problem has been studied in a series of papers, but most prior work focused on the static case. In this paper, we give a dynamic data structure in the 2D case which can answer queries in $O(\log^{1+o(1)} n + k \log^{1/2+o(1)} n)$ time, where k denotes the output size (the number of distinct colors in the query range), and which can support insertions and deletions in $O(\log^{2+o(1)} n)$ time (amortized) in the standard RAM model. This is the first fully dynamic structure with polylogarithmic update time whose query cost per color reported is *sublogarithmic* (near $\sqrt{\log n}$). We also give an alternative data structure with $O(\log^{1+o(1)} n + k \log^{3/4+o(1)} n)$ query time and $O(\log^{3/2+o(1)} n)$ update time (amortized). We also mention extensions to higher constant dimensions.

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

Keywords and phrases Range searching, dynamic data structures, word RAM

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.28

Funding Supported by NSF Grant CCF-1814026.

Acknowledgements We thank Saladi Rahul for helpful discussions.

1 Introduction

Range searching is one of the most fundamental data structure problems studied in computational geometry. Motivated by applications in databases and information retrieval, many researchers [5, 6, 7, 12, 13, 14, 15, 17, 18, 19, 20, 21, 22, 23, 24, 25, 27, 29, 30, 31, 32, 34] have investigated a natural *colored* variant of range searching (also called "categorical", or "generalized", range searching):

> Given a set of n points where each point is assigned a color (or "category"), we want to quickly report the distinct colors of the points inside a query range.

The query time should depend on the output size k, i.e., the number of distinct colors in the range (which could be much smaller than the number of points in the range). In this paper, we focus on the basic case of 2D *orthogonal* query ranges, i.e., axis-aligned rectangles.

A long series of work have studied this colored orthogonal range reporting problem, which turns out to be more challenging than the traditional uncolored problem; see Table 1 for a quick summary. Throughout the paper, we assume the standard word RAM model of computation.

All these prior papers addressed primarily static data structures, and surprisingly, not much progress has been made on the equally fundamental *dynamic* problem, where insertions and deletions of points are allowed.

Table 1 Static data structures for colored 2D orthogonal range reporting. (Coordinates are assumed to be integers bounded by U.)

	space	query time
Janardan & Lopez ('93) [21]	$O(n \log n)$	$O(\log^2 n + k)$
	$O(n \log^2 n)$	$O(\log n + k)$
Gupta, Janardan, & Smid ('95) [17]	$O(n \log^2 n)$	$O(\log n + k)$
Agarwal et al. (ESA'02) [1]	$O(n \log^2 n)$	$O(\log \log U + k)$
Mortensen ('03) [27]	$O(n \log n \log \log n)$	$O(\log^2 \log U + k)$
	$O(n \log n)$	$O(\log n \log^2 \log n + k)$
Shi & JaJa ('05) [34]	$O(n \log n)$	$O(\log n + k)$
Larsen & van Walderveen (SODA'13) [25]	$O(n \log n)$	$O(\log \log U + k)$
Nekrich (PODS'12) [30]	$O(n \log n)$	$O(\log \log U + k)$
Chan and Nekrich (SODA'20) [7]	$O(n \log^{3/4+\varepsilon} n)$	$O(\log \log U + k)$

Known dynamic colored results. The 1D dynamic colored range reporting problem – in which query ranges are intervals – can be easily reduced to 2D uncolored 3-sided range reporting (e.g., see the survey [16, Section 1.3.1]), and so by "textbook" *range trees* [11, 33], the problem can be solved with $O(\log n + k)$ query time and $O(\log^2 n)$ update time, or by incorporating dynamic fractional cascading [26], $O(\log n \log \log n + k)$ query time and $O(\log n \log \log n)$ update time. By using the current best results of Chan and Tsakalidis [9] on dynamic 2D uncolored range searching, the query time improves to $O(\frac{\log n}{\log \log n} + k)$ and the update time improves to $O(\log^{1/2+\varepsilon} n)$ (amortized) for an arbitrarily small constant $\varepsilon > 0$. (For dynamic 2D uncolored range searching, a lower bound of $\Omega(\frac{\log n}{\log \log n} + k)$ query time is known [2] for data structures with polylogarithmic update time; and $\sqrt{\log n}$ update time has also been recognized as a barrier that would be difficult to break with existing techniques [9].)

Straightforwardly, the dynamic 2D colored range reporting can be reduced dynamic 1D colored range reporting, by using a range tree on the x-coordinates, with a logarithmic-factor increase in both the query and update time. This implies a 2D colored data structure with $O(\frac{\log^2 n}{\log \log n} + k \log n)$ query time and $O(\log^{3/2+\varepsilon} n)$ update time. Notice the extra logarithmic factor in the $O(k \log n)$ term of the query cost: as the query range is decomposed into $O(\log n)$ sub-ranges, the same color may discovered $O(\log n)$ times with this approach.

Alternatively, it is known (e.g., see [16, Section 3.4]) that a colored range reporting query can be reduced to $O(k \log n)$ number of uncolored range emptiness queries, by using a range tree on the colors. This simple reduction works in the dynamic setting, with update time increased by a logarithmic factor. By Chan and Tsakalidis's result [9] on dynamic uncolored range emptiness, this implies a different dynamic 2D colored data structure with $O(k \frac{\log^2 n}{\log \log n})$ query time and $O(\log^{3/2+\varepsilon} n)$ update time, which is no better than the above.

Known static methods managed to avoid extra logarithmic factors in the k term (as can be seen in Table 1), but these methods do not adapt well to the dynamic setting. Some results were known in the insertion-only case [17], but it is open whether there exists a fully dynamic data structure with $O(\text{polylog}\, n + k)$ query time and $O(\text{polylog}\, n)$ update time for 2D colored orthogonal range reporting.[1]

[1] However, it is not difficult to obtain $O(\frac{\log n}{\log \log n} + k)$ query time with $O(n^{\varepsilon})$ update time, by modifying the reduction from 2D colored to 1D colored to use a larger fan-out $n^{\varepsilon'}$.

New dynamic colored result. We make progress towards this open problem by presenting a new data structure for dynamic 2D colored orthogonal range reporting with $O(\log n + k \log^{1/2+o(1)} n)$ query time and $O(\text{polylog}\, n)$ update time. The query cost per reported color is thus sublogarithmic ($\log^{1/2+o(1)} n$).

Interestingly, this improved query time bound is obtained using an approach similar to that in Chan and Pătraşcu's $O(n\sqrt{\log n})$-time inversion-counting algorithm [8], or in other known data structures with "fractional-power-of-log" update times [8, 9, 10, 28, 36]. As reinterpreted in Chan and Tsakalidis' framework [9], the approach roughly involves two parts: (i) the design of *micro-structures* for small input (of size near $2^{\sqrt{\log n}}$), which uses bit packing tricks; and (ii) the design of *macro-structures*, which uses more traditional range trees but with larger fan-out (also near $2^{\sqrt{\log n}}$) to reduce the tree depth. What is novel in our application is that the $\sqrt{\log n}$ factor appears in the k term of the query cost – we will apply bit packing to the query's output list of colors in the micro-structures. In addition, we follow an idea from Chan and Nekrich's static data structure for colored range reporting [7]: first solve a "capped" version of the problem, where the number of reported colors k is promised to be upper-bounded by some parameter K_0, and then extend the solution to general k by building a range tree on the colors. The overall method is conceptually not complicated, as explained in Section 2.

Improving the update time. In its simplest version, our data structure has $O(\log^{3+o(1)} n)$ update time. In Section 3, by combining with the more sophisticated techniques in Chan and Tsakalidis' work on dynamic uncolored orthogonal range searching [9], we further reduce the update time to $O(\log^{2+o(1)} n)$, while keeping roughly the same query time $O(\log^{1+o(1)} n + k \log^{1/2+o(1)} n)$. These bounds are amortized.

Alternative result. In Section 4, by using a different micro-structure, we also describe an alternative data structure which further lowers the update time to $O(\log^{3/2+o(1)} n)$, although the query time is increased to $O(\log^{1+o(1)} n + k \log^{3/4+o(1)} n)$. Still, compared to the aforementioned prior result with $O(\frac{\log^2 n}{\log\log n} + k \log n)$ query time and $O(\log^{3/2+\varepsilon} n)$ update time, we get a strictly better query time bound while keeping the same update time.

Higher dimensions. Our approach can also be extended to higher dimensions. In Section 5, we show how to answer colored orthogonal range reporting queries in a constant dimension d in $O(\log^{d-1} n + k \log^{d-2+1/d+o(1)} n)$ time, while supporting updates in $O(\text{polylog}\, n)$ time. The $\log^{d-2+1/d} n$ factor is intriguingly similar to Chan and Pătraşcu's bound on d-dimensional (uncolored) orthogonal range counting [8].

Computational model. We work with the standard w-bit word RAM model of computation, with $w = \Omega(\log n)$. On occasion, we assume that certain nonstandard word operations take unit time. This assumption may be removed by simulating such operations via table lookup, after an initial preprocessing of the table in $2^{O(w)}$ time, the cost of which is negligible since we will eventually set $w = \delta \log n$ for a sufficiently small constant $\delta > 0$.

2 First Method

In this section, we present our first method for 2D colored orthogonal range reporting, achieving roughly $O(\log n + k\sqrt{\log n})$ query time with polylogarithmic update time.

We begin by solving the case of *3-sided* query ranges, i.e., rectangles that are unbounded on one side, w.l.o.g., from below. The colored 3-sided problem is already challenging (and, in fact, so is 2-sided). The general idea is to build the data structure in two stages. First, in Section 2.1 we design a *micro-structure* to solve the problem for small input size, in which case the colors can be encoded in much fewer bits than $\log n$ and we can apply bit packing tricks. In Sections 2.2–2.3 we then use this micro-structure to build a *macro-structure*, solving the 3-sided problem for large input size. Finally, in Section 2.4, we note how the original 4-sided problem can be reduced to the 3-sided problem, without increasing the query time (though the update time is increased by an extra logarithmic factor).

2.1 3-Sided Micro-Structure

Our micro-structure is obtained by modifying the binary range tree and incorporating bit packing techniques.

▶ **Lemma 1.** *Given a set of at most s points in 2D, there is a data structure for colored 3-sided range reporting with*

- *query time $Q_3(s,k) = O(\log s + (k \log k \log^{2+o(1)} s)/w + k)$, and*
- *amortized update time $U_3(s) = O(\log^{1+o(1)} s)$.*

Proof. Since there are at most s points and thus at most s colors, we can map each color to an integer in $\{1, \ldots, s\}$. This enables us to pack $w/\log s$ colors into a single word. The mapping can be stored in a "color translation" table of s entries, so that we can translate each mapped color back to its original color in constant time. In an insertion of a point with a new color, we can simply map its color to the next unused integer.

The micro-structure takes the form of a binary range tree. At each node, we divide the plane into two vertical slabs, each with roughly half the number of points, such that each child node stores a recursive data structure for one of the slabs. For each slab σ, we store a list L_σ of all points in σ, sorted by the y-coordinates. In addition, we store a sublist L'_σ that contains the lowest point of each color in L_σ, also sorted by y. Using a colored predecessor search structure [28, Theorem 14], which has $O(\log^2 \log s)$ update time, for any given point, we can find the predecessor in L_σ and L'_σ in $O(\log^2 \log s)$ time, given its predecessor in L_τ of the parent slab τ; we can also find the predecessor of a specific color in $O(\log^2 \log s)$ time.

When a point p is inserted to L_σ, if p has no predecessor of the same color, we insert it to L'_σ and delete its colored successor (if one exists) from L'_σ. When a point p is deleted from L_σ, if p is in L'_σ, we delete it and insert its colored successor to L'_σ.

By bit packing, we store the list of colors in L_σ in a sequence of words, each containing between $w/(2 \log s)$ and $w/\log s$ colors. During an insertion, when the number of elements in a word exceed $w/\log s$, we split the word into two. During a deletion, when the number of elements in a word drops below $w/(2 \log s)$, we merge the word with its successor in the sequence, and if the number of elements in the merged word exceeds $w/\log s$, we split it. This takes constant time (using standard word operations such as shifts).

We can ensure that the range tree has $O(\log s)$ height, for example, by known weight-balancing techniques [4]. The overall update time is thus $O(\log s \log^2 \log s)$.

For a given 3-sided query range q, if q intersects only one slab, then we recurse in that slab; otherwise, we make a recursive 2-sided query in each slab. Given a 2-sided query range q open to the left, if the vertical side of q intersects the left slab, then we recurse in the left slab; otherwise, we report all distinct colors in the left slab σ by scanning its sublist L'_σ from the bottom and extracting a prefix, and then recurse in the right slab. We can handle 2-sided query ranges open to the right symmetrically.

Each color is reported only once at each level, and therefore is reported $O(\log s)$ times over the recursion. Thus, the combined list of reported colors requires $O(k \log s \cdot \log s)$ bits and can be represented in $O(1 + (k \log^2 s)/w)$ words. We then remove duplicate colors by sorting and performing a linear scan. A bit-packed version of mergesort on m elements takes time $O(\log m)$ times the number of words (using possibly nonstandard word operations). Thus, the cost is $O(\log s + (1 + (k \log^2 s)/w) \cdot \log(k \log s)) = O(\log s + (k \log k \log^{2+o(1)} s)/w)$. Afterwards, we spend $O(k)$ time to translate the output colors. ◀

2.2 3-Sided Capped Macro-Structure

Next, we present our macro-structure for large input. We first solve the K_0-*capped* problem. Here, the number of colors in the query range is promised to be smaller than a given fixed value K_0, i.e., $k \le K_0$.

▶ **Lemma 2.** *Fix a value $K_0 \le 2^{\sqrt{w}}$. Given n points in 2D, there is a data structure for colored 3-sided K_0-capped range reporting with*

- *query time $O(\log n + (k \log K_0 \log^{1+o(1)} n)/\sqrt{w} + k) \le O(\log n + k \log K_0 \log^{1/2+o(1)} n)$;*
- *amortized update time $O(\log^{1+o(1)} n)$.*

Proof. We will still use a variant of the range tree, but with a much larger fan-out s to be set later. At each node, the plane is divided into s vertical slabs, each with about $1/s$ of the points. For each slab σ, among the lowest point of each color, we only take the K_0 lowest ones. We replace the x-coordinates of these points with that of the left side of σ. We then store all sK_0 such points across all slabs in the micro-structure from Lemma 1, with $U_3(sK_0)$ update time and $Q_3(sK_0, k)$ query time. We maintain colored predecessor search structures at each node as before. We can ensure that the range tree has $O(\log_s n)$ height, by known weight-balancing techniques [4].

For a given 3-sided query range q, we identify the child slabs σ_1 and σ_2 containing the two vertices of q, answer a query for the micro-structure for $q - \sigma_1 - \sigma_2$, and then recurse in σ_1 and σ_2. We make $O(\log_s n)$ queries on the micro-structures along two paths of the tree, so the query time is

$$Q_3'(n,k) \;=\; O(Q_3(sK_0,k)\log_s n) \;=\; O\left(\log(sK_0) + k\frac{\log k \log^{2+o(1)}(sK_0)}{w} + k\right)\cdot \log_s n.$$

For each update, we make $O(\log_s n)$ updates on the micro-structures along a path of the tree, so the update time is

$$U_3'(n) \;=\; O(U_3(sK_0)\log_s n + \log_s n \log^2 \log n) \;=\; O((\log^{1+o(1)}(sK_0) + \log^2 \log n)\log_s n).$$

Setting $s = 2^{\sqrt{w}}$ and recalling that $K_0 \le 2^{\sqrt{w}}$, we obtain $Q_3'(n,k) = O(\log n + (k \log k \log^{1+o(1)} n)/\sqrt{w} + k)$ and $U_3'(n) = O(\log^{1+o(1)} n)$. ◀

Remark. If $k \ge K_0$, the algorithm may or may not succeed in reporting all k distinct colors. It is actually possible to *pre-check* whether the algorithm will succeed, in $O(\log n)$ time (without paying the $O((k \log K_0 \log^{1+o(1)} n)/\sqrt{w} + k)$ cost): In the data structure from Lemma 2, recall that in each slab we store the lowest K_0 points of distinct colors in a micro-structure. We mark the K_0-th lowest point with a special color. In the micro-structure from Lemma 1, in each L_σ, we maintain the lowest special point. If the query algorithm wants to report a prefix of L'_σ, we check that the lowest special point is not in the query range (otherwise the algorithm would fail). We also check that the K_0-th lowest point of L'_σ is not in the query range. The cost of pre-checking is $O(\log_s n \cdot \log(sK_0)) = O(\log n)$.

2.3 From Capped to Uncapped

We now remove the assumption $k < K_0$, to obtain a complete solution to the 3-sided problem:

▶ **Theorem 3.** *Given n points in 2D, there is a data structure for colored 3-sided range reporting with*

- *query time $O(\log n + (k \log^{1+o(1)} n)/\sqrt{w} + k) \leq O(\log n + k \log^{1/2+o(1)} n)$, and*
- *amortized update time $O(\log^{2+o(1)} n)$.*

Proof. We use a range tree on the colors, as in Chan and Nekrich's static colored method [7, proof of Theorem 2.3], with fan-out f. At each node we store its point set in the capped data structure from Lemma 2. The color class is partitioned into f color subclasses, each with roughly equal number of colors. Each child of a node corresponds to one of these subclasses. We can ensure that the range tree has $O(\log_f n)$ height, by known weight-balancing techniques [4].

To answer a query, at each node we first pre-check whether the query will succeed, as in the Remark after Lemma 2.[2] If so, we query the capped structure from Lemma 2 at the node and finish the query. Otherwise, we recurse in all f children.

Whenever we recurse in the children of a node, the node must contain $\geq K_0$ colors in the query range. Thus, the number of recursive calls per level is $O(fk/K_0)$, so the total number of recursive calls is $O((fk/K_0) \log_f n)$, excluding the root. Thus, the total cost of pre-checking is $O((fk/K_0) \log_f n \cdot \log n)$. The total query time spent on capped structures is $O(((fk/K_0) \log_f n + 1) \cdot \log n + (k \log K_0 \log^{1+o(1)} n)/\sqrt{w} + k)$.

For each update, we make $O(\log_f n)$ updates to the capped structures along a path of the tree, so the total update time is $O(\log_f n \cdot \log^{1+o(1)} n)$.

Setting $f = 2$ and $K_0 = \log^2 n$ yields query time $O(\log n + (k \log^{1+o(1)} n)/\sqrt{w} + k)$ and update time $O(\log^{2+o(1)} n)$. ◀

2.4 From 3-Sided to 4-Sided

By building a standard binary range tree where each node stores the 3-sided structure from Theorem 3, we can reduce a 4-sided query to two 3-sided queries. The query time remains the same, and the update time increases by a logarithmic factor. This immediately yields the following result:

▶ **Corollary 4.** *Given n points in 2D, there is a data structure for colored 4-sided range reporting with*

- *query time $O(\log n + k \log^{1/2+o(1)} n)$, and*
- *amortized update time $O(\log^{3+o(1)} n)$.*

3 Improving the 4-Sided Update Time

In this section, we apply more advanced techniques to improve the update time for 4-sided queries to $O(\log^{2+o(1)} n)$, which matches our 3-sided update time, avoiding the logarithmic-factor penalty in going from 3-sided to 4-sided.

Chan and Tsakalidis [9] proposed a framework for transforming micro-structures into macro-structures for uncolored range searching. We observe that their transformation works for the capped colored problem as well, with minor modifications. Our improved colored

[2] Chan and Nekrich [7] originally suggested some nontrivial approximate range counting approach to do the pre-checking, which we have bypassed.

result then follows by combining this transformation with our 3-sided micro-structure from Section 2.1, and with the known transformation of capped to uncapped structures used in Section 2.3.

In more details, Chan and Tsakalidis started with a technique of Mortensen [28, Theorem 1], which they dubbed the *van Emde Boas transformation*, for converting a micro-structure to a data structure for solving the *narrow grid* case, where input points have a small number of distinct x-coordinates. The conversion increases time bounds by only $\log\log$ factors, and is achieved by a recursion similar to van Emde Boas trees [35]. We observe that a similar transformation holds for the capped colored range reporting problem, if the cap K_0 is not too big. The statement below is adapted from Chan and Tsakalidis [9, Lemma 4]. In this section, 3-sided query ranges are unbounded from the left or the right (instead of from below).

▶ **Lemma 5.** *Fix K_0. Let X be a set of $O(s)$ values. Given a dynamic data structure for colored j-sided range reporting ($j \in \{3,4\}$) on s' points in $X \times \mathbb{R}$ with query time $Q_j(s,s',k)$ and update time $U_j(s,s')$ (amortized), there is a data structure for colored j-sided K_0-capped range reporting on n points in $X \times \mathbb{R}$ with (amortized)*

- *query time $Q_j(s,n,k) = O\left(Q_j(s,(sK_0)^{O(1)},k)\log\log n\right)$, and*
- *update time $U_j(s,n) = O\left(U_j(s,(sK_0)^{O(1)})\log^2\log n\right)$.*

If the given data structure supports updates to X in $U_X(s)$ time and this update procedure depends solely on X (and not the point set), the new data structure can support updates to X in $U_X(s)$ time.

The proof is a straightforward modification of the previous proof [28, 9]. (Basically, whenever in the uncolored solution we refer to the topmost/bottommost point, we now consider the K_0 topmost/bottommost points of distinct colors – this explains why the number of points in the micro-structures goes up from $s^{O(1)}$ to $(sK_0)^{O(1)}$.)

Next, Chan and Tsakalidis [9, Lemma 7] described a way to convert a narrow-grid structure into a data structure for the general case. This transformation, which they dubbed the (first) *range tree transformation*, uses standard range trees with fan-out s. We observe that the same transformation works for the capped colored problem:

▶ **Lemma 6.** *Fix K_0. Given a family of data structures $\mathcal{D}_j^{(i)}$ ($i \in \{1,\ldots,\log_s n\}$) for dynamic colored j-sided K_0-capped range reporting ($j \in \{3,4\}$) on n points in $X \times \mathbb{R}$ ($|X| = O(s)$) with query time $Q_j^{(i)}(s,n,k)$ update time $U_j^{(i)}(s,n)$ (amortized), where updates to X take $U_X(s)$ time, there is a data structure for colored 4-sided K_0-capped range reporting on n points in the plane with the following query and update time (amortized):*

$$\begin{aligned} Q_4'(n,k) &= O\left(\max_i Q_4^{(i)}(s,n,k) + \max_i Q_3^{(i)}(s,n,k)\log_s n + \log_s n \log^2\log n\right) \\ U_4'(n) &= O\left(\sum_{i=1}^{\log_s n}(U_4^{(i)}(s,n) + U_3^{(i)}(s,n)) + \sum_{i=1}^{\log_s n}\frac{U_X^{(i)}(s)}{s^{i-1}} + \log_s n\log^2\log n\right). \end{aligned}$$

The proof of the above requires essentially no change in the previous proof [9].

We are now ready to put together our new improved data structure. We set $K_0 = s$. Our colored 3-sided method in Theorem 3 gives

$$\begin{aligned} Q_3(s,s^{O(1)},k) &= O(\log s + (k\log^{1+o(1)} s)/\sqrt{w} + k) \\ U_3(s,s^{O(1)}) &= O(\log^{2+o(1)} s). \end{aligned}$$

The known colored 4-sided method mentioned in the introduction gives

$$\begin{aligned} Q_4(s, s^{O(1)}, k) &= O(\tfrac{\log^2 s}{\log\log s} + k \log s) \\ U_4(s, s^{O(1)}) &= O(\log^{3/2+\varepsilon} s). \end{aligned}$$

(Actually, a weaker bound of $U_4(s, s^{O(1)}) = O(\log^{2+o(1)} s)$ suffices.) In both of these methods, $U_X(s) = 0$. Applying Lemmas 5–6 (with $Q_4^{(i)} = Q_4$ and $Q_3^{(i)} = Q_3$ for all i) yields

$$\begin{aligned} Q'_4(n, k) &= O\left(\tfrac{\log^2 s}{\log\log s} + k \log s + (\log s + (k \log^{1+o(1)} s)/\sqrt{w} + k) \log_s n\right) \cdot \log^2 \log n \\ U'_4(n) &= O(\log^{2+o(1)} s \log_s n) \cdot \log^2 \log n. \end{aligned}$$

Setting $s = 2^{\sqrt{\log n}}$ gives

$$\begin{aligned} Q'_4(n, k) &= O(\log^{1+o(1)} n + k \log^{1/2+o(1)} n) \\ U'_4(n) &= O(\log^{3/2+o(1)} n). \end{aligned}$$

All this is for the K_0-capped problem with $K_0 = s = 2^{\sqrt{\log n}}$. Finally, we solve the general uncapped problem by the color range tree technique in Section 2.3. As in the Remark after Lemma 2, it is possible to pre-check whether the query will succeed, in $O(\log^{1+o(1)} n)$ time (without paying the $O(k \log^{1/2+o(1)} n)$ cost); the modification is straightforward (see the Appendix regarding the van Emde Boas transformation). As in the proof of Theorem 3, by using a range tree on colors with fan-out f, the overall query time is $O(((fk/K_0) \log_f n + 1) \cdot \log^{1+o(1)} n + k \log^{1/2+o(1)} n)$, and the overall update time is $O(\log_f n \log^{3/2+o(1)} n)$.

By setting $f = \sqrt{K_0}$ for example, the query time remains $O(\log^{1+o(1)} n + k \log^{1/2+o(1)} n)$ and the update time becomes $O(\log^{2+o(1)} n)$.

▶ **Theorem 7.** *Given n points in 2D, there is a data structure for colored 4-sided range reporting with*

- *query time $O(\log^{1+o(1)} n + k \log^{1/2+o(1)} n)$, and*
- *amortized update time $O(\log^{2+o(1)} n)$.*

4 Alternative Method

We now describe an alternative method that has smaller update time $O(\log^{3/2+o(1)} n)$, though the query time is increased to $O(\log^{1+o(1)} n + k \log^{3/4+o(1)} n)$.

The solution uses a different micro-structure, given in the lemma below, which works directly for the general 4-sided case. Its main advantage is that it has *constant* amortized update time when the input size s is sufficiently small (around $2^{\log^{1/4} n}$). It is a colored variant of a micro-structure for uncolored 4-sided queries by Chan and Tsakalidis [9, Lemma 5(ii) and Lemma 6].

▶ **Lemma 8.** *Fix a parameter $\bar{w} \le w$ with $\bar{w} = \Omega(\log s)$. Given a set of at most s points in a universe $X \times Y$ with $|X|, |Y| = s^{O(1)}$, and there is a data structure for 4-sided colored range reporting with*

- *amortized query time $O(\log^2 s + (k \log^{3+o(1)} s)/\bar{w} + k)$, and*
- *amortized update time $O(1 + (\log^4 s)/\bar{w})$.*

In addition, it supports:

- *updates to Y in $O(\log^2 \log n)$ time, and*
- *updates to X in $2^{O(\bar{w})}$ time, where the update procedure depends solely on X (and not the point set).*

Proof. Recall that colors can be mapped to $(\log s)$-bit integers, as noted in the first paragraph of the proof of Lemma 1.

We follow the approach by Chan and Tsakalidis [9, Lemma 5(ii)] and mimick an *external-memory* data structure with block size $B := \delta\bar{w}/\log s$ for a sufficiently small constant δ. A block of B points can be encoded in $O(\delta\bar{w})$ bits and can thus be packed in a single word.

The well-known *buffer tree* of Arge [3] is a 1D external-memory data structure with subconstant $(O(\frac{1}{B}\log_B s))$ amortized update cost. For their 4-sided uncolored micro-structure, Chan and Tsakalidis suggested an analogous *buffered* version of the binary range tree in 2D, which has amortized update cost $O(\frac{1}{B}\log^2 s) = O((\log^3 s)/\bar{w})$. Our solution in the colored setting is similar.

Roughly speaking, in the primary 2D range tree, each node corresponds to a *canonical horizontal slab* and stores a secondary tree, which is a 1D range tree of the points in the corresponding canonical horizontal slab. In a secondary tree, each node corresponds a *canonical rectangle*. In the buffered version, each node of the primary tree and secondary trees holds a buffer of up to B update requests that are yet to be processed.

We will not re-explain all the details of the buffered 2D range tree, but just describe the main changes needed for the colored problem: For each node ν of a secondary tree, we additionally maintain a sorted list L_ν of the distinct colors appearing in the corresponding canonical rectangle, along with the multiplicity of these colors. We also hold a buffer Z_ν of (a possibly large number of) of update requests yet to be processed at ν. Updates in ν are handled lazily, by just appending the requests to the buffer Z_ν. We postpone the work of actually updating L_ν to querying. When a query wants to report the colors in a node ν, we sort Z_ν by color and perform a linear scan to update L_ν. Let ℓ_ν and z_ν be the number of elements in L_ν and Z_ν. The lists L_ν and Z_ν can be packed in $O((\ell_\nu \log s)/w)$ and $O(z_\nu \log s)/\bar{w})$ words respectively. By a bit-packed version of mergesort, sorting Z_ν takes time $O((z_\nu \log s)/\bar{w}) \cdot \log s$. Since an update leads to update requests in the buffers Z_ν of $O(\log^2 s)$ nodes ν, we can assign an amortized cost of $O((\log^4 s)/\bar{w})$ per update to cover this sorting cost. Scanning L_ν takes time $O((\ell_\nu \log s)/w)$. Now, ℓ_ν is bounded by the number of colors reported in ν during the previous query that involves ν. Since a query visits $O(\log^2 s)$ nodes, we can assign an amortized cost of $O((k \log^3 s)/w)$ per query with output size k to cover this scanning cost.

In a query, we report the colors in the sorted list L_ν for $O(\log^2 s)$ nodes ν. The combined list of reported colors has $O(k \log^2 s)$ colors and requires $O((k \log^3 s)/\bar{w})$ words. We remove duplicate colors by merging these $O(\log^2 s)$ sorted lists. This multi-way merging can be done in $O(\log\log s)$ rounds, each taking time linear in the number of words. Thus, the cost is $O((k \log^{3+o(1)} s)/\bar{w})$. Afterwards, we spend $O(k)$ time to translate the output colors.

Updates to X and Y can be handled using the same ideas by Chan and Tsakalidis [9, Lemma 6]. ◀

We can now put together the new data structure by combining the new micro-structure with Chan and Tsakalidis' framework [9], via the van Emde Boas transformation and range tree transformation, as we have described in Section 3.

As before, we set $K_0 = s$. For each $j \in \{3, 4\}$ and each $i \in \{1, \ldots, \log_s n\}$, we apply Lemma 8 with the choice $\bar{w} = \delta i \log s$ to get

$$\begin{aligned} Q_j^{(i)}(s, s^{O(1)}, k) &= O(\log^2 s + (k \log^{2+o(1)} s)/i + k) \\ U_j^{(i)}(s, s^{O(1)}) &= O(1 + (\log^{3+o(1)} s)/i) \\ U_X^{(i)}(s) &= s^{O(\delta i)}. \end{aligned}$$

Applying Lemmas 5–6 and noting that $\sum_{i=1}^{\log_s n} 1/i = O(\log \log_s n)$ yields

$$\begin{aligned} Q'_4(n,k) &= O\left(\log^2 s \log_s n + k \log^{2+o(1)} s + k \log_s n\right) \cdot \log^3 \log n \\ U'_4(n) &= O(\log_s n + \log^{3+o(1)} s) \cdot \log^3 \log n. \end{aligned}$$

Setting $s = 2^{\log^{1/4} n}$ gives

$$\begin{aligned} Q'_4(n,k) &= O(\log^{5/4+o(1)} n + k \log^{3/4+o(1)} n) \\ U'_4(n) &= O(\log^{3/4+o(1)} n). \end{aligned}$$

All this is for the K_0-capped problem with $K_0 = s = 2^{\log^{1/4} n}$. Finally, we solve the general uncapped problem by using the color range tree technique in Section 2.3. As before, it is possible to pre-check whether the query will succeed, in $O(\log^{5/4+o(1)} n)$ time (without paying the $O(k \log^{3/4+o(1)} n)$ cost). The overall query time is then $O(((fk/K_0) \log_f n + 1) \cdot \log^{5/4+o(1)} n + k \log^{3/4+o(1)} n)$, and the overall update time is $O(\log_f n \log^{3/4+o(1)} n)$.

By setting $f = \sqrt{K_0}$ for example, the query time remains $O(\log^{5/4+o(1)} n + k \log^{3/4+o(1)} n)$ and the update time becomes $O(\log^{3/2+o(1)} n)$.

The term $O(\log^{5/4+o(1)} n)$ dominates the query time bound only when $k \ll \sqrt{\log n}$, but in this case, we can switch to the method in Section 3, which solves the problem for an even bigger cap $2^{\sqrt{\log n}}$ with a better query time $O(\log^{1+o(1)} n + k \log^{1/2+o(1)} n)$ while keeping update time $O(\log^{3/2+o(1)} n)$.

▶ **Theorem 9.** *Given n points in 2D, there is a data structure for colored 4-sided range reporting with*

- *amortized query time $O(\log^{1+o(1)} n + k \log^{3/4+o(1)} n)$, and*
- *amortized update time $O(\log^{3/2+o(1)} n)$.*

5 Higher Dimensions

We now point out a generalization of the method in Section 2 to higher dimensions.

First, it is straightforward to generalize the micro-structure from Lemma 1 (here, a $(2d-1)$-sided range is unbounded along the d-th axis):

▶ **Lemma 10.** *Given a set of at most s points in a constant dimension d, there is a data structure for colored $(2d-1)$-sided range reporting with*

- *query time $Q_{2d-1}(s,k) = O(\log^{d-1} s + (k \log k \log^{d+o(1)} s)/w + k)$, and*
- *amortized update time $U_{2d-1}(s) = O(\log^{d-1+o(1)} s)$.*

We generalize the capped macro-structure from Lemma 2:

▶ **Lemma 11.** *Fix a value $K_0 \le 2^{w^{1/d}}$. Given n points in a constant dimension d, there is a data structure for colored $(2d-1)$-sided K_0-capped range reporting with*

- *query time $O(\log^{d-1} n + k \log K_0 \log^{d-2+1/d+o(1)} n)$;*
- *amortized update time $O(\log^{d-1+o(1)} n)$.*

Proof. For each $j \in \{0, \ldots, d-1\}$, define $\mathcal{P}_j$ to be the (capped) subproblem in d dimensions which the first j coordinates of all points come from a set X of $O(s)$ values.

Following the proof of Lemma 2 of using a range tree but along the j-th axis, we can reduce problem $\mathcal{P}_j$ to problem $\mathcal{P}_{j+1}$ at the expense of increasing the query and update time by one $O(\log_s n)$ factor.

We solve problem $\mathcal{P}_{d-1}$ directly: Points lie on $O(s^{d-1})$ vertical lines. For each such vertical lines, among the lowest point of each color, we only take the K_0 lowest points, and store all these $O(s^{d-1}K_0)$ points in the micro-structure from Lemma 10.

Thus, the original problem $\mathcal{P}_0$ can be solved with query and update time

$$\begin{aligned} Q'_{2d-1}(n,k) &= O(Q_{2d-1}(s^{d-1}K_0,k)\cdot(\log_s n)^{d-1}) \\ &= O\left(\log^{d-1}(sK_0) + k\frac{\log k \log^{d+o(1)}(sK_0)}{w} + k\right)\cdot(\log_s n)^{d-1} \\ U'_{2d-1}(n,K) &= O(U_{2d-1}(s^{d-1}K_0,k)\cdot(\log_s n)^{d-1}) = O(\log^{d-1+o(1)}(sK_0))\cdot(\log_s n)^{d-1}. \end{aligned}$$

Setting $s = 2^{w^{1/d}}$ and recalling that $K_0 \le 2^{w^{1/d}}$, we obtain $Q'_{2d-1}(n,k) = O(\log^{d-1} n + (k\log k\log^{d-1+o(1)} n)/w^{1-1/d} + k)$ and $U'_{2d-1}(n) = O(\log^{d-1+o(1)} n)$. ◀

We can solve the general uncapped problem by using the color range tree technique in Section 2.3 (invoking the capped structure with K_0 polylogarithmic). The update time is increased by a logarithmic factor:

▶ **Theorem 12.** *Given n points in a constant dimension d, there is a data structure for colored $(2d-1)$-sided range reporting with*

- *query time $O(\log^{d-1} n + k\log^{d-2+1/d+o(1)} n)$, and*
- *amortized update time $O(\log^{d+o(1)} n)$.*

Finally, as in Section 2.4, general $(2d)$-sided queries can be reduced to $(2d-1)$-sided queries, with another logarithmic-factor increase in the update time:

▶ **Corollary 13.** *Given n points in a constant dimension d, there is a data structure for colored $(2d-1)$-sided range reporting with*

- *query time $O(\log^{d-1} n + k\log^{d-2+1/d+o(1)} n)$, and*
- *amortized update time $O(\log^{d+1+o(1)} n)$.*

6 Final Remarks

All the extra $\log^{o(1)} n$ factors are $(\log\log n)^{O(1)}$. In the 2D query time bound of Corollary 4, the $O(\log n)$ first term is likely improvable slightly to $O(\frac{\log n}{\log\log n})$ (to match known lower bounds), by increasing the fan-out from 2 to $\log^{\delta} n$ in the proof of Lemma 1.

The main open question is whether $O(\text{polylog}\, n + k)$ query time is achievable with $O(\text{polylog}\, n)$ update time in 2D.

In higher dimensions, we have not tried to optimize the number of logarithmic factors in the update time, but the ideas in Section 3 should help. A more intriguing question is whether $O(\text{polylog}\, n + k\log^{d-2-\delta} n)$ query time is achievable with $O(\text{polylog}\, n)$ update time. Even for the static problem, we do not know how to get $O(\text{polylog}\, n + k)$ query time with $O(n\,\text{polylog}\, n)$ space; current techniques only give $O(\text{polylog}\, n + k\log^{d-3} n)$ query time [7].

References

1 Pankaj K. Agarwal, Sathish Govindarajan, and S. Muthukrishnan. Range searching in categorical data: Colored range searching on grid. In *Proc. 10th Annual European Symposium on Algorithms (ESA)*, pages 17–28, 2002. doi:10.1007/3-540-45749-6_6.

2 Stephen Alstrup, Thore Husfeldt, and Theis Rauhe. Marked ancestor problems. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 534–544, 1998. doi:10.1109/SFCS.1998.743504.

3 Lars Arge. The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, 2003. doi:10.1007/s00453-003-1021-x.

4 Lars Arge and Jeffrey Scott Vitter. Optimal external memory interval management. *SIAM J. Comput.*, 32(6):1488–1508, 2003.

5 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.

6 Timothy M. Chan, Qizheng He, and Yakov Nekrich. Further results on colored range searching. In *Proc. 36th International Symposium on Computational Geometry (SoCG)*, pages 28:1–28:15, 2020. doi:10.4230/LIPIcs.SoCG.2020.28.

7 Timothy M. Chan and Yakov Nekrich. Better data structures for colored orthogonal range reporting. In *Proc. 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 627–636, 2020. doi:10.1137/1.9781611975994.38.

8 Timothy M. Chan and Mihai Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In *Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 161–173, 2010. doi:10.1137/1.9781611973075.15.

9 Timothy M. Chan and Konstantinos Tsakalidis. Dynamic orthogonal range searching on the RAM, revisited. In *Proc. 33rd International Symposium on Computational Geometry (SoCG)*, pages 28:1–28:13, 2017. doi:10.4230/LIPIcs.SoCG.2017.28.

10 Timothy M. Chan and Konstantinos Tsakalidis. Dynamic planar orthogonal point location in sublogarithmic time. In *Proc. 34th International Symposium on Computational Geometry (SoCG)*, pages 25:1–25:15, 2018. doi:10.4230/LIPIcs.SoCG.2018.25.

11 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd edition, 2008.

12 Hicham El-Zein, J. Ian Munro, and Yakov Nekrich. Succinct color searching in one dimension. In *Proc. 28th International Symposium on Algorithms and Computation (ISAAC)*, pages 30:1–30:11, 2017. doi:10.4230/LIPIcs.ISAAC.2017.30.

13 Travis Gagie, Juha Kärkkäinen, Gonzalo Navarro, and Simon J. Puglisi. Colored range queries and document retrieval. *Theor. Comput. Sci.*, 483:36–50, 2013. doi:10.1016/j.tcs.2012.08.004.

14 Arnab Ganguly, J. Ian Munro, Yakov Nekrich, Rahul Shah, and Sharma V. Thankachan. Categorical range reporting with frequencies. In *Proc. 22nd International Conference on Database Theory (ICDT)*, pages 9:1–9:19, 2019. doi:10.4230/LIPIcs.ICDT.2019.9.

15 Roberto Grossi and Søren Vind. Colored range searching in linear space. In *Proc. 14th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 229–240, 2014. doi:10.1007/978-3-319-08404-6_20.

16 Prosenjit Gupta, Ravi Janardan, Saladi Rahul, and Michiel H. M. Smid. Computational geometry: Generalized (or colored) intersection searching. In *Handbook of Data Structures and Applications*, chapter 67, pages 1042–1057. CRC Press, 2nd edition, 2018. URL: https://www-users.cs.umn.edu/~sala0198/Papers/ds2-handbook.pdf.

17 Prosenjit Gupta, Ravi Janardan, and Michiel H. M. Smid. Further results on generalized intersection searching problems: Counting, reporting, and dynamization. *J. Algorithms*, 19(2):282–317, 1995. doi:10.1006/jagm.1995.1038.

18 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.

19 Prosenjit Gupta, Ravi Janardan, and Michiel H. M. Smid. A technique for adding range restrictions to generalized searching problems. *Inf. Process. Lett.*, 64(5):263–269, 1997. doi:10.1016/S0020-0190(97)00183-X.

20 Prosenjit Gupta, Ravi Janardan, and Michiel H. M. Smid. Algorithms for some intersection searching problems involving circular objects. *International Journal of Mathematical Algorithms*, 1:35–52, 1999.

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

22 Haim Kaplan, Natan Rubin, Micha Sharir, and Elad Verbin. Efficient colored orthogonal range counting. *SIAM J. Comput.*, 38(3):982–1011, 2008. doi:10.1137/070684483.

23 Haim Kaplan, Micha Sharir, and Elad Verbin. Colored intersection searching via sparse rectangular matrix multiplication. In *Proc. 22nd ACM Symposium on Computational Geometry (SoCG)*, pages 52–60, 2006. doi:10.1145/1137856.1137866.

24 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.

25 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.

26 Kurt Mehlhorn and Stefan Näher. Dynamic fractional cascading. *Algorithmica*, 5(2):215–241, 1990. doi:10.1007/BF01840386.

27 Christian Worm Mortensen. Generalized static orthogonal range searching in less space. Technical report, IT University Technical Report Series 2003-33, 2003.

28 Christian Worm Mortensen. Fully dynamic orthogonal range reporting on RAM. *SIAM J. Comput.*, 35(6):1494–1525, 2006. doi:10.1137/S0097539703436722.

29 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.

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

31 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.

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

33 F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer–Verlag, 1985.

34 Qingmin Shi and Joseph JáJá. Optimal and near-optimal algorithms for generalized intersection reporting on pointer machines. *Inf. Process. Lett.*, 95(3):382–388, 2005. doi:10.1016/j.ipl.2005.04.008.

35 Peter van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Inf. Process. Lett.*, 6(3):80–82, 1977. doi:10.1016/0020-0190(77)90031-X.

36 Bryan T. Wilkinson. Amortized bounds for dynamic orthogonal range reporting. In *Proc. 22th Annual European Symposium on Algorithms (ESA)*, pages 842–856, 2014. doi:10.1007/978-3-662-44777-2_69.