

# Approximate Colored Range Queries<sup>\*</sup>

Ying Kit Lai, Chung Keung Poon, and Benyun Shi

Dept. of Computer Science, City U. of Hong Kong, China

**Abstract.** In this paper, we formulate a class of colored range query problems to model the multi-dimensional range queries in the presence of categorical information. By applying appropriate sketching techniques on our framework, we obtained efficient data structures that provide approximate solutions to these problems. In addition, the framework can be employed to attack other related problems by finding the appropriate summary structures.

## 1 Introduction

Range query problems are concerned with the storage and maintenance of a set of multi-dimensional data points so that given any query region, certain information about the data points lying within the region can be answered efficiently. Typical information of interest includes the set of points, the number of them, the sum, maximum and minimum of their associated values, etc. The corresponding problems are referred to as the range report, count, sum, max and min queries respectively.

In many applications, the data points are associated with categorical attributes and we are interested in information about the categories of the points lying within a query region, instead of the individual points themselves. Such problems can be abstracted as *colored range query* problems in which points in the given dataset are labeled with colors and we ask for information about the colors of points within a query region. As pointed out by Agarwal et al. [1], colored range queries are highly prevalent in database queries. Applications of such queries can also be found in document retrieval [13] and in indexing multi-dimensional strings [9].

To give a concrete example and to motivate our definition of a whole class of colored range queries, consider a database of sales records, each of which consisting of the following attributes: time, branch location (x, y-coordinates), product ID and sales. Suppose we are interested in information about sales grouped by product. Then we can model the database as a set of points in a 3-dimensional space (defined by the time and branch attributes) where each point is associated with a color (the product ID) as well as a value (the sales).

A specific query to the database can be to ask for the number of different products sold during a period of time within a region of branch locations. In

---

<sup>\*</sup> The work described in this paper was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 1198/03E].

other words, we want to compute the number of distinct colors for the points lying within the corresponding query region. This is called the *colored range counting* problem. Another possible query is to ask for the best-selling products, i.e., we group the individual points in the region by product ID, sum the sales of each group and then find those groups that contribute most for the total sales of all products. This is a query that has not been formulated and studied before.

### 1.1 Problem Formulation

Generalizing the above sample queries, we can say that a general class of queries is to aggregate the points by colors (where the aggregation can be a simple count of points or a sum of values associated with points having that color) and then extract certain information about the colors. To capture such queries, we formalize the following class of colored range query problems. We will assume that our dataset consists of  $n$  points lying over a  $d$ -dimensional space and each point is associated with a color as well as a numeric value. We also assume there are  $m$  distinct colors in the universe of colors where  $m$  is large. Otherwise, one can construct an ordinary data structure for each color and solve the colored range queries by querying the  $m$  structures one by one.

Conceptually, one can view the set of all possible colors as a *colored* dimension. After the aggregation by colors, we have a set of points (colors) along the colored dimension, each associated with a value. Now, consider the set  $\mathcal{F}$  of possible functions on the colored dimension. One may be interested in reporting the set or number of colors. These we denote by **report** and **count** respectively. One can also ask for the total, maximum or minimum value associated with the colors. These are denoted as **sum**, **max** and **min** respectively. We denote by **heavy** the query that asks for the colors with values above a certain threshold. Thus, we define  $\mathcal{F} = \{\text{sum, count, max, min, report, heavy}\}$ . Then for any  $f \in \mathcal{F}$ , a *colored range- $f$  query* on a query region  $R_1 \times R_2 \times \dots \times R_d$  (where each  $R_i$  is an interval in dimension  $i$ ) is to apply  $f$  on the set of colors within the query region. Depending on whether the value associated with a color is: (1) a count of points having that color, or (2) a sum of values associated with points having that color, we called the corresponding query *unweighted* or *weighted* respectively.

Note that for  $f = \text{sum}$ , the weighted and unweighted colored range- $f$  queries are just ordinary range sum and range count queries (without colors) respectively. Nevertheless, this gives an alternative formulation of these ordinary range query problems. Also, both the weighted and unweighted colored range-**report** queries are equivalent to the colored range reporting problem. For  $f = \text{count}$ , both the weighted and unweighted queries are equivalent to the colored range counting problem we mentioned earlier.

### 1.2 Previous Work

Except for the colored range reporting and counting problems, none of the problems we defined above have been investigated before. As one will see below, many

of these problems are difficult and it is not clear if existing techniques for range query problems can yield efficient data structures for them.

The colored range reporting problem is probably the most studied among all the known colored range query problems and there are rather efficient solutions. To our knowledge, Janardan and Lopez [12] were the first to investigate the static case for 1- and 2-dimensional points. Gupta et al [11] devised a *chaining* technique that transforms a set of 1-dimensional colored points into a set of 2-dimensional points (without colors). Using this technique, they obtained dynamic structures for colored range reporting up to 2 dimensions and a static structure for 3 dimensions. The structures of [12,11] generally require  $O(n \cdot \text{polylog}(n))$  space and  $O(\text{polylog}(n) + k)$  query time (where  $k$  is the number of distinct colors in the result set) and  $O(\text{polylog}(n))$  update time. Recently Agarwal et al. [1] studied the variant of the problem where the data points are lying on an integer grid. In terms of techniques, they adapted the intuition and ideas from [12,11] and extended them to work for higher dimensions. Motivated by a number of document retrieval problems, Muthukrishnan [13] studied yet another variant where the points are stored in a 1-dimensional array. Nanopoulos et al. [14] studied the problem in the context of large, disk resident data sets. They proposed a multi-tree index to solve the problem but did not provide any analytical bound for their method.

For the colored range counting problem, much less is known. In the 1-dimensional case, Gupta et al. [11] obtained several efficient static and dynamic data structures with  $O(n \cdot \text{polylog}(n))$  space and  $O(\text{polylog}(n))$  query/update time. Using their chaining technique (mentioned above) and the best-known results on 2-dimensional range counting [15,6], one can actually improve their bounds, see Table 1. For 2 dimensions, their approach is to first obtain a persistent 1-dimensional data structure and then apply a standard line-sweeping approach. This results in static 2-dimensional structures with high (quadratic and cubic) storage requirement. Moreover, the approach does not readily yield a dynamic 2-dimensional structure.

There are other variants of colored queries which are outside the scope of this paper, . This includes the various colored intersection problems studied in [12,11] and the *colored significant-presence* queries proposed in [4]. The colored significant-presence queries is to report only those colors with at least a certain fraction of their points fallen into the query range. It looks similar but is in fact different from our colored range-heavy queries.

**Table 1.** Summary of results on colored range counting

Problem	Space	Query Time	Update Time	Reference
exact dynamic 1-d	$n \log n$	$\log^2 n$	$\log^2 n$	Gupta et al. [11]
	$n \log n$	$\log n$	$\log n$	Gupta et al. [11] + Willard [15]
	$n$	$\log^2 n$	$\log^2 n$	Gupta et al. [11] + Chazelle [6]
approx. dynamic 1-d	$n$	$\log^2 n$	$\log^2 n$	this paper
exact <b>static</b> 2-d	$n^2 \log^2 n$	$\log n$		Gupta et al. [11] + Willard [15]
approx. dynamic 2-d	$n \log n$	$\log^3 n$	$\log^3 n$	this paper

### 1.3 Our Contribution

In this paper, we propose a general methodology for constructing randomized data structures that can produce approximate answers. Our approach significantly differs from previous techniques for colored range queries in that sketching techniques are employed. For most of the problems, our method yields efficient solutions. In particular, our colored range counting structure has much smaller space than previous structures for 2-dimensions or above, as shown in Table 1.

We now illustrate our methodology using 1-dimensional range queries. A common technique for (1-dimensional) range query is to build a balanced binary search tree  $T$  where each leaf represents a data point and each internal node represents a group of data points, i.e., those represented by the leaves in the subtree of that internal node. Denote by  $S(u)$  the set of points represented by node  $u$ . It is well-known that given any query range (an interval), one can always represent the set of points within the query range as a disjoint union of  $S(u)$ 's for at most  $O(\log n)$   $u$ 's, with no more than two such  $u$ 's in each level of the tree  $T$ . This union of sets can be obtained by traversing  $T$  with the left and right boundaries of the query interval in  $O(\log n)$  time. Thus, if we store with each node  $u$  certain summary information about  $S(u)$ , a range query may be solvable in  $O(\log n)$  time. For example, for range sum queries, the relevant information about  $S(u)$  would be the sum of values among all points in  $S(u)$ .

Unfortunately, the success of this approach very much depends on the additivity of the summary information. Consider the colored range counting problem in which we are to find the number of distinct colors in a query range. Suppose we store the number of distinct colors in  $S(u)$  for each tree node  $u$ . However, the number of distinct colors in the query range cannot be computed by simply adding the count for each  $S(u)$  for the  $O(\log n)$   $u$ 's that partitions the query region. Similarly, problems such as colored range minimum cannot be solved by associating with each node  $u$  in  $T$  the minimum aggregate value among the colors of points in  $S(u)$ .

To overcome the problem, our approach is to store an appropriate sketch with each node  $u$  to summarize the relevant information about  $S(u)$ . Specifically, the sketch should possess the following properties: (1) the size of a sketch is small, (2) the sketches are additive, and (3) an approximate answer can be obtained from the sketch. Property (1) ensures that the time and space of the data structure are within good bounds. Property (2) allows us to generate the required sketch for any query region on-the-fly (i.e., during the query time). By the structure of  $T$ , we only need to add  $O(\log n)$  sketches to form the desired sketch. Finally, property (3) is essential if the sketch is to be useful for our problem at all.

Using this approach, we obtain a data structure for the 1-dimensional colored range counting problem with  $O(hn)$  space and  $O(h \log n)$  query and update times where  $h = O(\frac{1}{\epsilon^2} \log n)$  is the size of an appropriate sketch. The data structure is randomized in the sense that with probability  $1 - n^{-\Omega(1)}$ , it produces an approximate answer accurate to within  $\epsilon$  factor of error for *all* possible queries. We can further reduce the storage to  $O(n)$  without increasing the asymptotic

complexities of queries and updates (treating  $\epsilon$  as a fixed constant). Applying standard techniques [3], we obtain multi-dimensional colored range query structures with a blow-up factor of  $\log n$  in storage and operation time per dimension. More specifically, we utilize the additivity of the appropriate sketches to obtain a sketch that summarizes the  $d$ -dimensional query region and then obtain the desired information from the sketch. In general, our data structures require  $O(dn \log^{d-1} n)$  space and support queries and updates in  $O(d \log^{d+1} n)$  time.

Using the same methodology but with different sketches, we obtain data structures with similar performance bounds for many other colored range query problems we defined here.

## 2 Preliminaries

### 2.1 Notations and Sketches

We will represent information about colors using vectors. Thus, we will be considering vectors and their norms. For a vector  $\mathbf{a} = (a_1, a_2, \dots, a_m)$ , we denote by  $\|\mathbf{a}\|_p$  its  $l_p$ -norm, i.e.,  $\|\mathbf{a}\|_p = (\sum_{i=1}^m |a_i|^p)^{1/p}$ . When  $p = 0$ ,  $\|\mathbf{a}\|_0$  is defined as the number of non-zero components in  $\mathbf{a}$ .

The sketch of a vector  $\mathbf{a}$  is the projection of  $\mathbf{a}$  onto a set of random basis vectors. The purpose of a sketch is to estimate certain quantities about  $\mathbf{a}$  to within certain error with high probability. The number,  $h$ , of required random basis vectors, and hence the size of the sketch, depends on the two parameters,  $\epsilon$  and  $\delta$ , that controls the error and failure probability respectively; and may also depend on  $m$ . Usually, we have  $h \ll m$  so that storage reduction is achieved. Another important property of sketches is that additivity holds for many types of sketches, i.e., the sketch of  $\mathbf{a} \pm \mathbf{b}$  can be computed by adding/subtracting the sketch of  $\mathbf{a}$  with that of  $\mathbf{b}$ . Below we describe two sketches that we will employ, namely the *Count-Min* or *CM sketch* and the  *$l_0$ -sketch*. They both possess properties (1) and (2) mentioned in section 1.3.

### 2.2 Count-Min Sketch

When the components of a vector  $\mathbf{a}$  are all non-negative, its *CM sketch* ([8]), denoted  $CM(\mathbf{a})$ , allows us to estimate any component,  $a_i$ , of  $\mathbf{a}$  by specifying its index  $i$ . We call this a *point estimate* on  $\mathbf{a}$ . A certain collection of CM sketches, denoted  $CCM(\mathbf{a})$ , allows us to report the set of indices  $i$  such that  $a_i$  is larger than a certain threshold fraction  $\phi$  of  $\|\mathbf{a}\|_1$ . We call this *heavy hitters estimate*.

*Point Estimates.* We first describe the construction of  $CM(\mathbf{a})$  and the algorithm for point estimate. The sketch is essentially a two-dimensional array of counts with width  $w = \lceil \frac{m}{\epsilon} \rceil$  and depth  $d = \lceil \ln \frac{1}{\delta} \rceil$ , given the sketch parameters of  $\epsilon$  and  $\delta$ . The size of the sketch is thus  $h = wd = O(\frac{1}{\epsilon} \log \frac{1}{\delta})$ . We represent the counts in the sketch by  $count(1, 1), \dots, count(d, w)$ .

Given  $\mathbf{a}$ , we choose  $d$  hash functions  $h_1, \dots, h_d$  ( $h_i : \{1, \dots, m\} \rightarrow \{1, \dots, w\}$ ) uniformly at random from a family of pairwise independent hash functions ([5]). We initialize all the counts in the sketch to 0. Then for each  $j \in \{1, \dots, d\}$ , we

hash the components of  $\mathbf{a}$  into different cells of row  $j$  according to  $h_j$ . That is, for each  $i \in \{1, \dots, m\}$ , the value  $a_i$  is added to the cell  $count(j, h_j(i))$ . Besides the sketch, we also store  $d$  hash functions which takes negligible storage. Any component  $a_i$  of  $\mathbf{a}$  is estimated as  $\hat{a}_i = \min_{j=1}^d \{count(j, h_j(i))\}$ .

**Theorem 1.** [8] For any vector  $\mathbf{a} = (a_1, \dots, a_m)$  and any pair of parameters  $\epsilon, \delta > 0$ , the CM sketch of  $\mathbf{a}$ ,  $CM(\mathbf{a})$ , has size  $h = O(\frac{1}{\epsilon} \log \frac{1}{\delta})$  and supports point estimates in  $g = O(\log \frac{1}{\delta})$  time. The estimate  $\hat{a}_i$  of  $a_i$  satisfies: (1)  $a_i \leq \hat{a}_i$  and (2) with probability  $1 - \delta$ ,  $\hat{a}_i \leq a_i + \epsilon \|\mathbf{a}\|_1$ . Further, the sketch can be updated (for changes in a component of  $\mathbf{a}$ ) in  $O(g)$  and constructed in  $O(gm + h)$  time.

*Heavy Hitters Estimates.* We first explain the notion of *dyadic intervals*. For convenience, we assume  $m$  is a power of 2. A *dyadic interval*  $I_{j,k}$  on the universe  $\{1, \dots, m\}$  is an interval of the form  $[k2^j, (k + 1)2^j - 1]$ , for  $j \in \{1, \dots, \log m\}$  and  $k \in \{0, \dots, m/2^j - 1\}$ . The parameter  $j$  of a dyadic interval is its *resolution level* from the *finest*:  $I_{0,i} = \{i\}$  where  $i \in \{1, \dots, m\}$  to the *coarsest*  $I_{\log m,0} = \{1, \dots, m\}$ . There are  $\log m$  resolution levels and  $2m - 1$  dyadic intervals altogether, organized in a binary tree-like structure.

To support heavy hitter estimates, the collection  $CCM(\mathbf{a})$  consists of  $\log m$  CM sketches, each of size  $dw = \log(\frac{\log m}{\delta\phi}) \times \frac{\epsilon}{\delta} = O(\frac{1}{\epsilon} \log(\frac{\log m}{\delta\phi}))$ . The first sketch is just  $CM(\mathbf{a})$ . The second sketch is the CM sketch on a vector with  $m/2$  components, each being the sum of  $a_i$ 's over all  $i$  in a dyadic interval  $I_{1,k}$  where  $0 \leq k \leq m/2 - 1$ . The third up to the  $\log m$ -th sketch are CM sketches of  $\mathbf{a}$  at a progressively coarser and coarser resolution.

To find the heavy hitters, we start from the sketch of the coarsest resolution level and check for the dyadic interval(s) with (estimated) weights exceeding the threshold  $(\phi + \epsilon)\|\mathbf{a}\|_1$ . We then explore their children dyadic intervals at the next finer level of resolution. Repeating this until we arrive at dyadic intervals of length 1, we can locate all the components  $a_i$  such that the estimated  $a_i$  exceeds  $(\phi + \epsilon)\|\mathbf{a}\|_1$ . Note that in each sketch, at most  $2/\phi$  dyadic intervals are examined. As for updates, since each point in the universe  $\{1, \dots, m\}$  is a member of  $\log m$  sketches, each of the sketches are updated when an update on a point arrives.

**Theorem 2.** [8] For any  $\mathbf{a} = (a_1, \dots, a_m)$  and  $\epsilon, \delta, \phi > 0$ , the collection of sketches  $CCM(\mathbf{a})$  has size  $h = O(\frac{1}{\epsilon} \log m \log(\frac{\log m}{\delta\phi}))$ . In  $O(\epsilon h/\phi)$  time, it can report all the components with weight at least  $(\phi + \epsilon)\|\mathbf{a}\|_1$ , and with probability  $1 - \delta$ , it reports no component with weight less than  $\phi\|\mathbf{a}\|_1$ . The sketch can be updated in  $g = O(\log m \log(\frac{\log m}{\delta\phi}))$  time and constructed in  $O(gm + h)$  time.

### 2.3 $l_0$ -Sketch

When the magnitude of each component of  $\mathbf{a}$  is bounded from above by some value  $U$ , we can compute its  $l_0$ -sketch ([7]), denoted  $L_0(\mathbf{a})$ , which allows us to approximate,  $\|\mathbf{a}\|_0$ , the number of non-zero entries in  $\mathbf{a}$ .

**Theorem 3.** [7] For any  $\mathbf{a} = (a_1, \dots, a_m)$  such that  $|a_i| \leq U$  for every  $i$ , and any  $\epsilon, \delta > 0$ , the  $l_0$ -sketch of  $\mathbf{a}$ ,  $L_0(\mathbf{a})$ , has size  $h = O(\frac{1}{\epsilon^2} \log(\frac{1}{\delta}))$ . Using the sketch, the  $l_0$ -norm,  $\|\mathbf{a}\|_0$ , can be approximated to within a factor of  $1 \pm \epsilon$  with

probability  $1 - \delta$  in  $O(h)$  time. Further, it can be updated in  $O(h)$  time and constructed in  $O(hm)$  time.

### 3 Colored Range Counting

*Construction and Storage.* To solve for the 1-dimensional case, we construct a balanced binary search tree  $T$  to store the points. For each node  $u$  in  $T$ , we apply the  $l_0$ -sketch to summarize the at most  $m$  distinct colors present in  $S(u)$ . More precisely, we define  $\mathbf{a} = (a_1, a_2, \dots, a_m)$  to be the color frequency vector so that  $a_i$  represents the number of occurrences of color  $i$  in a region. In the region, we would like to know the number of non-zero components of the vector  $\mathbf{a}$ , or formally the  $l_0$ -norm of  $\mathbf{a}$ . In general  $m$  can be very large and hence storing an  $\mathbf{a}$  explicitly for each node  $u$  will be very space-consuming. Therefore, we will store the  $l_0$ -sketch of  $\mathbf{a}$  so that we only need  $h = O(\frac{1}{\epsilon^2} \log(\frac{1}{\delta}))$  space per sketch. (The value of  $\epsilon$  and  $\delta$  will be determined later.) Hence in total we need  $O(hn)$  space.

The  $l_0$ -sketch of a leaf  $u$  (representing one data point) can be constructed in  $O(h)$  time. By additivity, the  $l_0$ -sketch of an internal node  $v$  can be computed by adding that of its two children. This takes again  $O(h)$  time. Hence, the construction of the whole data structure requires  $O(hn)$  time.

*Querying.* To handle a query  $[x, y]$ , we search  $T$  for  $x$  and  $y$  in order to identify the set  $V$  of internal nodes the union of whose leaves is the set of points lying within  $[x, y]$ . Note that  $|V| = O(\log n)$ . By additivity, the  $l_0$ -sketch of any query region can be computed by adding the corresponding  $O(\log n)$  many  $l_0$ -sketches. Thus, a query takes  $O(h \log n)$  time. We ignore the query time for the resulting sketch here as the time for the additions clearly dominates.

*Updating.* When inserting or deleting a point  $x$  without changing the structure of  $T$ , the update can be done straightforwardly. Suppose the point has color  $i$  and value  $\mu$ . Then for each node  $v$  along the path from the root to  $x$ , its sketch has to be updated. Let  $\mathbf{a}$  be the color frequency vector associated with  $v$ . Then inserting (deleting) the point will cause  $a_i$  to increase (decrease) by  $\mu$ . Let  $\mathbf{b} = (b_1, b_2, \dots, b_m)$  such that  $b_i = \mu$  and  $b_j = 0$  for all  $j \neq i$ . Then  $L_0(\mathbf{a})$  should be updated to  $L_0(\mathbf{a}) \pm L_0(\mathbf{b})$  by additivity and this takes  $O(h)$  time. The total update time along the path is  $O(h \log n)$ .

When the update causes the structure of  $T$  to change, we need to modify the sketches of all the affected nodes. For each such node, its sketch can be re-computed by adding the sketches of all its children (in the new tree). For example, if we use a red-black tree [2,10] for  $T$ , an update requires at most 3 rotations and propagating the change from the points of rotation to the root by adding sketches takes  $O(h \log n)$  time in the worst case. (For higher dimensions, the update time can be made worst case using standard techniques [16].)

*Storage Reduction.* We can reduce the space requirement of the data structure to be independent of  $\epsilon$  and  $\delta$ . The idea is to store the sketch only for nodes  $v$  in  $T$  with sufficient number of leaves. More precisely, let  $F$  be the set of internal nodes  $v \in T$  such that  $v$  has fewer than  $O(h)$  leaves while parent of  $v$  has at least  $O(h)$  leaves. We call the nodes in  $F$  the *frontier nodes*. Clearly, there are  $O(n/h)$

of them. We will only store the  $l_0$ -sketch for nodes above the frontier  $F$ . Then the storage requirement is reduced to  $O(n)$ . When querying for a range  $[x, y]$ , we search for  $x$  and  $y$  in  $T$  as before. When the search reaches a node  $v$  in  $F$ , we construct the  $l_0$ -sketch for the portion of the subtree rooted at  $v$  that lies within the query region on the fly. This is done by traversing the subtree of  $T$  rooted at  $v$  to construct the color frequency vector  $\mathbf{a}$  for only the points lying within the query region. This takes  $O(h)$  time by definition of  $F$ . Then we construct  $L_0(\mathbf{a})$  using  $O(h^2)$  time. The total query time now becomes  $O(\log n + h^2)$ .

*Choosing  $\delta$ .* Now we consider the value of  $\delta$  which controls the failure probability. We will generate all the  $O(n)$  sketches using the same set of basis vectors. Moreover, for each query region, we will also generate the corresponding sketch on the fly. In total, there are at most  $O(n^2)$  query regions. We want to have a random basis such that with high probability (i.e.,  $1 - \delta'$  where  $\delta' = n^{-\Omega(1)}$ ), all the  $O(n^2)$  sketches give good enough approximations. Clearly,  $\delta' = O(n^2\delta)$ . Thus, we set  $\delta = n^{-\Omega(1)}$  and so  $h = O(\log n)$  for constant  $\epsilon$ . In general for  $d$  dimensions, number of query regions become  $O(n^{2d})$  and thus  $h = O(d \log n)$ .

Since the size of sketches now depends on  $n$ , we may have to change the size of the sketches as  $n$  changes. However, it can be shown that by building the structure with sketch size  $h$  doubled and rebuilding the whole structure when  $n$  becomes too large, one can achieve an amortized update bound of  $O(\log^2 n)$ .

## 4 Colored Range-Min Queries

*Construction and Storage.* For the 1-dimensional case, we consider the base tree  $T$ . For each node  $u$  in  $T$ , we will store the CM sketch of the color vector  $\mathbf{a} = (a_1, a_2, \dots, a_m)$  associated with  $u$ . Each  $a_i$  represents the sum or count of the individual point values with color  $i$  under the subtree  $T(u)$ .

However, instead of estimating individual component, we use  $CM(\mathbf{a})$  to estimate the minimum component of  $\mathbf{a}$ . We do this by finding the minimum value in each row of  $CM(\mathbf{a})$  and then taking the minimum among the  $\lceil \ln \frac{1}{\delta} \rceil$  rows. In other words, we find the minimum value within the whole sketch  $CM(\mathbf{a})$ .

There is a small technical issue. It is possible that some cells in  $CM(\mathbf{a})$  are not hashed into by any index  $i \in \{1, \dots, m\}$ . Then our method will return a zero value (the initial sketch cell value). To identify and exclude those cells, we slightly modify the sketch construction. First, we initialize the whole table to some negative values. Then, when hashing the components of  $\mathbf{a}$  into the table, the first component hashed into a cell will overwrite the initial negative value while subsequent components will add to the existing value. When computing the minimum cell value, we only consider those non-negative cells. The following lemmas prove that the estimated minimum is accurate with high probability.

**Lemma 1.** [8] For any  $i \in \{1, \dots, m\}$ ,  $j \in \{1, \dots, \lceil \ln \frac{1}{\delta} \rceil\}$ ,  $E[\text{count}(j, h_j(i)) - a_i] \leq \frac{\epsilon}{e} \|\mathbf{a}\|_1$ .

**Lemma 2.** Let  $a_t = \min_{i=1}^m \{a_i\}$ . For all  $j$ , define  $m_j = \min\{\text{count}(j, i') \mid \text{count}(j, i') \geq 0\}$ . Then  $E[m_j - a_t] \leq \frac{\epsilon}{e} \|\mathbf{a}\|_1$ .

**Lemma 3.** *Let  $a_t = \min_{i=1}^m \{a_i\}$ . Define  $cmm = \min_j \{m_j\}$ . Then  $cmm \geq a_t$  and  $cmm \leq a_t + \epsilon \|\mathbf{a}\|_1$  with probability  $1 - \delta$ .*

We need  $h = O(\frac{1}{\epsilon} \log \frac{1}{\delta})$  space to store a CM sketch and thus for all node  $u$  of tree  $T$ , we need  $O(hn)$  space where  $n$  is the total number of points. The CM sketch of a leaf can be constructed in  $O(h)$  time. By additivity, the sketch of an internal node can be constructed by adding those of its two children using  $O(h)$  time. The construction of the whole data structure thus requires  $O(hn)$  time.

*Querying.* Like the query in colored range counting, we first identify the set  $V$  of  $O(\log n)$  internal nodes corresponding to the query range. Then, by the additivity, we compute the sketch of the query range by adding the corresponding  $O(\log n)$  sketches. So, it takes  $O(h \log n)$  time for a query.

*Updating.* In the case where structure of  $T$  remains unchanged, the update procedure is just similar to Section 3 and the only difference is the update time. Note that we need only  $O(g) = O(\log \frac{1}{\delta})$  time to update a node in  $T$  with a single value. So the total time we need for updating will be  $O(g \log n)$ . In case where rotations occur due to changes in the structure of  $T$  (implemented as a red-black tree), the time required to update the sketch of an affected node by adding those of its children is  $O(h)$ . Since the number of affected nodes is at most  $O(\log n)$ , an update requires  $O(h \log n)$  time in the worst case.

*Storage Reduction and choosing  $\delta$ .* If we just store the CM sketches for nodes above the frontier  $F$  as defined in Section 3,  $h = O(\frac{1}{\epsilon} \log \frac{1}{\delta})$  in this case, we can reduce the storage requirement to  $O(n)$  while the query time increases to  $O(h \log n + hg)$  and update time remains to be  $O(h \log n)$ . Choosing  $\delta = n^{-\Omega(1)}$ , the asymptotic complexity of query and update times are both  $O(\log^2 n)$ .

## 5 Heavy Colors

*Construction and Storage.* Our main result in this section is the solution to the approximate colored range-heavy problem, i.e.  $f = \text{heavy}$ . We will focus on the unweighted case. (The weighted case is very similar.) Again, we associate with a node  $u$  in  $T$  the vector  $\mathbf{a} = (a_1, \dots, a_m)$  such that  $a_i$  is the number of occurrences of color  $i$  in the region ( $a_i$  is the sum of values with color  $i$  when weighted). We further assume all components of  $\mathbf{a}$  to be non-negative. Our problem then is to report those colors with  $a_i \geq \phi \|\mathbf{a}\|_1$ , where  $\phi \in [0, 1]$ . We store for each node  $u$  the collection of sketches  $CCM(\mathbf{a})$  for the vector  $\mathbf{a}$ . Each of the sketches represents a level of dyadic intervals (as described in Section 2.2). In this setting, we require only  $h = O(\frac{1}{\epsilon} \log(m) \log(\frac{\log m}{\delta \phi}))$  space per node. Thus, the total space needed is  $O(hn)$  for the whole tree  $T$ .

Using similar idea as in the previous section, the construction time of sketches for the whole tree takes  $O(hn)$  time. Querying takes  $O(h \log n + \epsilon h / \phi) = O(h \log n)$  time according to Theorem 2 and since  $\phi$  is constant. Updating takes  $O(h \log n)$  time. Furthermore, the storage can be reduced to  $O(n)$  while the query time is increased to  $O(h \log n + h^2)$  and update time remains to be  $O(h \log n)$ .

## 6 Conclusion

In this paper, we have formulated a new class of colored range problems and proposed a general approach of using sketches to solve many of them approximately. We believe that there are two benefits in adopting this approach. First, advancements in the research on sketches and summary structures with the required properties immediately improve the solutions for the corresponding colored range query problems. Second, by finding the appropriate summary structures, the method can be employed to solve other unsolved problems. Also, our approach can be applied to other indexing structures, e.g. R-tree and its variants.

## References

1. P.K. Agarwal, S. Govindarajan, and S. Muthukrishnan. Range searching in categorical data: colored range searching on grid. In *ESA'02*, pages 17–28, 2002.
2. R. Bayer. Symmetric binary B-trees: data structures and maintenance algorithms. *Acta Informatica*, 1:290–306, 1972.
3. J.L. Bentley. Multidimensional divide-and-conquer. *Comm. ACM*, 23(4):214–228, April 1980.
4. M. Berg and H.J. Haverkort. Significant-presence range queries in categorical data. In *WADS'03, LNCS 2748*, pages 462–473, 2003.
5. J. L. Carter and M. N. Wegman. Universal classes of hash functions. *J. Computer and System Sciences*, 18:143–154, 1979.
6. B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Computing*, 17(3):427–462, June 1988.
7. G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan. Comparing data streams using Hamming norms (how to zero in). In *Proc. of 28th Int'l Conf. on Very Large Data Bases*, pages 335–345, 2002.
8. G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. In *6th Latin American Theoretical Informatics (LATIN), LNCS 2976*, pages 29–38, 2004.
9. P. Ferragina, N. Koudas, D. Srivastava, and S. Muthukrishnan. Two-dimensional substring indexing. In *Proc. 20th ACM Symp. on Principles of Database Systems*, pages 282–288, 2001.
10. L.J. Guibas and R. Sedgwick. A dichromatic framework for balanced trees. In *FOCS'78*, pages 8–21, 1978.
11. P. Gupta, R. Janardan, and M. Smid. Further results on generalized intersection searching problems: counting, reporting, and dynamization. *J. Algorithms*, 19:282–317, 1995.
12. R. Janardan and M. Lopez. Generalized intersection searching problems. *Int'l J. Computational Geometry and Applications*, 3(1):39–69, 1993.
13. S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *SODA'02*, pages 657–666, 2002.
14. A. Nanopoulos and P. Bozaris. Categorical range queries in large databases. In *SSTD'03, LNCS 2750*, pages 122–139, 2003.
15. D.E. Willard. New data structures for orthogonal queries. *SIAM J. Computing*, 14(1):232–253, February 1985.
16. D.E. Willard and G.S. Lueker. Adding range restriction capability to dynamic data structures. *J. ACM*, 32(3):597–617, July 1985.