Available online at www.sciencedirect.com
ScienceDirect
Journal of Discrete Algorithms 6 (2008) 420–432
JOURNAL OF DISCRETE ALGORITHMS
www.elsevier.com/locate/jda

Approximate colored range and point enclosure queries ☆

Ying Kit Lai, Chung Keung Poon *, Benyun Shi

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

Received 17 July 2007; received in revised form 23 October 2007; accepted 23 October 2007

Available online 6 November 2007

Abstract

In this paper, we formulate two classes of problems, the *colored range query problems* and the *colored point enclosure query problems* to model multi-dimensional range and point enclosure queries in the presence of categorical information. Many of these problems are difficult to solve using traditional data structural techniques. Based on a new framework of combining sketching techniques and traditional data structures, we obtain two sets of results in solving the problems approximately and efficiently. In addition, the framework can be employed to attack other related problems by finding the appropriate summary structures.

Keywords: Colored range queries; Colored point enclosure queries; Sketches; Approximate data 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 [19] and in indexing multi-dimensional strings [13].

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.

☆ 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]. Some of the results in this paper appeared in a preliminary form in the 16th Symposium on Algorithms and Computation (ISAAC'05), December 2005, pp. 360–369.

* Corresponding author.
E-mail address: ckpoon@cs.cityu.edu.hk (C.K. Poon).

1570-8667/$ – see front matter
doi:10.1016/j.jda.2007.10.001

The time attribute naturally defines a dimension endowed with a total ordering and so do the x- and y-coordinates. However, the product ID attribute defines a categorical dimension whose values are not ordered. We can artificially impose a total order but it will not be very often (and natural) for one to ask for information about an arbitrary range of product ID. Thus here one can model a record as a data point in a 3-dimensional space (time and x, y-coordinates) such that each point is associated with a color (representing the product ID) and a value (for the sales attribute).

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 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.

Point enclosure query problems are similar to range query problems but they are about maintaining a set of multidimensional hyper-rectangles and queries are posed as points. Given any arbitrary query point, we are to report information pertaining to the set of hyper-rectangles intersecting the query point. Examples are point enclosure report, count, max and min queries.

Data regions (hyper-rectangles) can also be associated with categories (colors) and *colored point enclosure query* problems are defined for such context. These problems are to compute information concerning colors of data regions containing a query point.

To motivate our study of the class of colored point enclosure query problems, consider network filters that apply different security policies on packets depending on their source and/or destination addresses. Each of the network filters is specified with ranges of source and destination addresses, as well as the security operations to be applied if a packet with addresses in the ranges is encountered. Source and destination IP addresses possess total ordering and are naturally dimensional attributes. Then packets can be modeled as points, and network filters as data regions in this 2-dimensional plane. Further, filters can be labeled with colors while each color represents a single security operation. Given a packet, if we ask for the number of security operations to be applied on it, the problem is a colored point enclosure counting problem. On the other hand, finding those security operations to be applied on a packet would be a colored point enclosure report problem.

1.1. Problem formulation

We now define the class of colored range query problems formally. Generalizing the above sample range queries, a 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 *color* 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 color 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

$$\mathcal{F} = \{\mathsf{sum}, \mathsf{count}, \mathsf{max}, \mathsf{min}, \mathsf{report}, \mathsf{heavy}\}.$$

Then for any $f \in \mathcal{F}$, a *colored range-f query* on a query region $R_1 \times R_2 \times \cdots \times R_d$ (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 $f = \mathsf{sum}$, 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 $f = \mathsf{count}$, both the weighted and unweighted queries are equivalent to the colored range counting problem we mentioned earlier.

The class of point enclosure query problems can be defined similarly with the dataset now consisting of n colored data regions instead of points. We still assume the number of colors m is very large which renders storing an ordinary data structure for each color infeasible. Then for any $f \in \mathcal{F}$, a *colored point enclosure-f query* on a query point $q = (q_1, q_2, \ldots, q_d)$ (where q_i is the coordinate of q on dimension i) is to apply f on the set of colors containing the query point. Again, colored point enclosure queries can be weighted or unweighted, depending whether we are counting the number of data regions having the same color or summing the associated values of the regions with that color.

1.2. Previous work

Except for the colored range reporting and counting problems, none of the colored range query 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 [17] were the first to investigate the static case for 1- and 2-dimensional points. Gupta et al. [15] 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 [15,17] generally require $O(n \cdot \mathit{polylog}(n))$ space and $O(\mathit{polylog}(n) + k)$ query time (where k is the number of distinct colors in the result set) and $O(\mathit{polylog}(n))$ 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 [15,17] and extended them to work for higher dimensions. Motivated by a number of document retrieval problems, Muthukrishnan [19] studied yet another variant where the points are stored in a 1-dimensional array. Nanopoulos et al. [20] 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. [15] obtained several efficient static and dynamic data structures with $O(n \cdot \mathit{polylog}(n))$ space and $O(\mathit{polylog}(n))$ query/update time. Using their chaining technique (mentioned above) and the best-known results on 2-dimensional range counting [8,22], 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 [15,17] and the *colored significant-presence* queries proposed in [6]. The

Table 1
Summary of results on colored range counting. (The approximation problem is to give estimates within $1 \pm \epsilon$ of the true value with probability $1 - \delta$ for all queries.)

Problem	Space	Query time	Update time	Reference
exact dynamic 1-dim	$n \log n$	$\log^2 n$	$\log^2 n$	[15]
	$n \log n$	$\log n$	$\log n$	[15] + [22]
	n	$\log^2 n$	$\log^2 n$	[15] + [8]
approx. dynamic 1-dim	n	$(\frac{1}{\epsilon^2} \log \frac{n}{\delta})^2$	$\frac{\log n}{\epsilon^2} \log \frac{n}{\delta}$ amortized	our result
exact **static** 2-dim	$n^2 \log^2 n$	$\log n$		[15] + [22]
approx. dynamic 2-dim	$n \log n$	$\frac{\log n}{\epsilon^4} \log^2 \frac{n}{\delta}$	$\frac{\log^2 n}{\epsilon^2} \log \frac{n}{\delta}$ amortized	our result
approx. dynamic d-dim ($d > 2$)	$n \log^{d-1} n$	$\frac{d \log^{d-1} n}{\epsilon^4} \log^2 \frac{n}{\delta}$	$\frac{d \log^d n}{\epsilon^2} \log \frac{n}{\delta}$ amortized	our result

Table 2
Summary of results on colored point enclosure counting. (The approximation problem is to give estimates within $1 \pm \epsilon$ of the true value with probability $1 - \delta$ for all queries.)

Problem	Space	Query Time	Update time	Reference
exact **static** 1-dim	n	$\log n$		[15]
approx. dynamic 1-dim	$\frac{n}{\epsilon^2} \log \frac{n}{\delta}$	$\frac{\log n}{\epsilon^2} \log \frac{n}{\delta}$	$\frac{\log n}{\epsilon^2} \log \frac{n}{\delta}$ amortized	our result
	n	$\frac{\log n}{\epsilon^4} \log^2 \frac{n}{\delta}$	$\frac{\log n}{\epsilon^2} \log \frac{n}{\delta}$ amortized	our result
exact **static** 2-dim	$n \log n$	$\log n$		[15]
approx. dynamic 2-dim	$\frac{n \log n}{\epsilon^2} \log \frac{n}{\delta}$	$\frac{\log^2 n}{\epsilon^2} \log \frac{n}{\delta}$	$\frac{\log^2 n}{\epsilon^2} \log \frac{n}{\delta}$ amortized	our result
	$n \log n$	$\frac{\log^2 n}{\epsilon^4} \log^2 \frac{n}{\delta}$	$\frac{\log^2 n}{\epsilon^2} \log \frac{n}{\delta}$ amortized	our result
approx. dynamic d-dim ($d > 2$)	$\frac{dn \log^{d-1} n}{\epsilon^2} \log \frac{n}{\delta}$	$\frac{d \log^d n}{\epsilon^2} \log \frac{n}{\delta}$	$\frac{d \log^d n}{\epsilon^2} \log \frac{n}{\delta}$ amortized	our result
	$n \log^{d-1} n$	$\frac{d \log^d n}{\epsilon^4} \log^2 \frac{n}{\delta}$	$\frac{d \log^d n}{\epsilon^2} \log \frac{n}{\delta}$ amortized	our result

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.

For the colored point enclosure query problems, only colored point enclosure reporting and counting have been studied. Janardan and Lopez [17] designed a static structure that answers the reporting queries in $O(\log n + k)$ time where k is the number of colors to be reported. The structure uses $O(n^{\beta_d})$ space where $\beta_1 = 1$ and $\beta_d = \beta_{d-1} + \frac{1}{1+(d-1)\beta_{d-1}}$ and d is the number of the dimensions. For example, $\beta_2 = 1.5$ and $\beta_3 = 1.75$. Gupta et al. [15] proposed a segment tree that makes use of the 1-dimensional colored point enclosure report structure as given in [17] to solve the 2-dimensional dynamic reporting problem. They managed to obtain a linear space structure with $O((k+1)\log n)$ query time and $O(\log n)$ amortized update time.

For the colored point enclosure counting problem, we found only static solutions. As shown in Table 2, Gupta et al. [15] produced efficient linear static structures that solves the 1 and 2-d colored point enclosure counting problem but no solutions for higher dimensions ($d > 2$) are known. We believe the exact versions of the problem are difficult and traditional point enclosure structures are not known to provide promising solutions.

1.3. Our contributions

In Section 3, we will propose a general methodology for designing randomized data structures that can produce approximate answers. That is, given any parameters $\epsilon, \delta > 0$, our construction yields, with probability $1 - \delta$, a data structure that gives ϵ-*approximate* answers for all queries where the definition of "ϵ-approximate" depends on the specific problem. For colored range-count queries, it means to return a value within $1 \pm \epsilon$ of the true value. For colored range-min, the error is bounded by an ϵ fraction of the total value of all colors. For colored range-heavy queries, the structure should return all colors with values above the threshold by a margin proportional to ϵ and not to return any color with value below the threshold. Our approach significantly differs from previous techniques for colored range queries and colored point enclosure 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. In addition, we are the first to devise solutions for the dynamic colored point enclosure problems.

We believe that there are two benefits of proposing the general method. First, advancements in the research on sketches or other summary structures having the required properties mentioned in the start of Section 3 immediately improve the solutions for the corresponding colored range and point enclosure 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. For example, our colored range counting technique works on R-tree [16] and its variants [3,21].

1.4. Organization of the paper

The remainder of this paper is organized as follows. In Section 2, we provide some backgrounds on the contemporary summary structures, namely the Count-Min sketches [10] and the l_0-sketch [9]. We note that the sketches possess additivity and their space is independent of the number of vector components, which are important properties that we make use of in later sections.

Section 3 addresses the problem of colored range queries and explains the difficulties in achieving exact solutions. We describe in detail our framework employed to attack the above defined colored range query problems and present three solutions based on the framework to attack three different problems. The material covered in this section is published in [18].

In Section 4, we investigate the problem of colored point enclosure queries. We present two solutions for the colored point enclosure counting problem that represent a tradeoff between space and query bounds. The first approach incorporates the l_0-sketch into a segment tree while the second one employs the 1-d colored range counting result mentioned in Section 3.

Finally, Section 5 summarizes the results of our research, and discuss the research contributions. We also suggest future directions in which our research can be extended.

2. Background on summary structures

2.1. Notations and sketches

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

The sketch of a vector $\vec{a}$ is the projection of $\vec{a}$ onto a set of random basis vectors. The purpose of a sketch is to estimate certain quantities about $\vec{a}$ 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, ϵ and μ, that control 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 $\vec{a} \pm \vec{b}$ can be computed by adding/subtracting the sketch of $\vec{a}$ with that of $\vec{b}$. Below we describe two sketches that we will employ, namely the *Count-Min* or *CM sketch* and the *l_0-sketch*.

2.2. Count-Min sketch

When the components of a vector $\vec{a}$ are all non-negative, its *CM sketch* [10], denoted $CM(\vec{a})$, allows us to estimate any component, a_i, of $\vec{a}$ by specifying its index i. We call this a *point estimate* on $\vec{a}$. A certain collection of CM sketches, denoted $CCM(\vec{a})$, allows us to report the set of indices i such that a_i is larger than a certain threshold fraction ϕ of $\|\vec{a}\|_1$. We call this *heavy hitters estimate*.

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

Given $\vec{a}$, we choose d hash functions $h_1, \ldots, h_d$ (where $h_i : \{1, \ldots, m\} \to \{1, \ldots, w\}$) uniformly at random from a family of pairwise independent hash functions [7]. We also initialize all the counts in the sketch to 0. Then for each $j \in \{1, \ldots, d\}$, we hash the components of $\vec{a}$ into different cells of row j according to h_j. That is, for each $i \in \{1, \ldots, m\}$, 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 $\vec{a}$ is estimated as $\hat{a}_i = \min_{j=1}^{d} \{count(j, h_j(i))\}$.

Theorem 2.1. *(See [10].) For any vector $\vec{a} = (a_1, \ldots, a_m)$ and any pair of parameters $\epsilon, \mu > 0$, the CM sketch of $\vec{a}$, $CM(\vec{a})$, has size $h = O(\frac{1}{\epsilon} \log \frac{1}{\mu})$ and supports point estimates in $g = O(\log \frac{1}{\mu})$ time. The estimate $\hat{a}_i$ of a_i satisfies:* (1) $a_i \leqslant \hat{a}_i$ *and* (2) *with probability* $1 - \mu$, $\hat{a}_i \leqslant a_i + \epsilon \|\vec{a}\|_1$. *Further, the sketch can be updated (to handle changes in a component of $\vec{a}$) in $O(g)$ time 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 $\{1, \ldots, m\}$ is an interval of the form $[k2^j, (k+1)2^j - 1]$, for $j \in \{1, \ldots, \log m\}$ and $k \in \{0, \ldots, \frac{m}{2^j} - 1\}$. The parameter j of a dyadic interval is its *resolution level* from the *finest*: $I_{0,i} = \{i\}$ to the *coarsest* $I_{\log m,0} = \{1, \ldots, m\}$. 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 $CCM(\vec{a})$ consists of $\log m$ CM sketches, each of size $dw = \log(\frac{\log m}{\mu\phi}) \times \frac{e}{\epsilon} = O(\frac{1}{\epsilon} \log(\frac{\log m}{\mu\phi}))$. The first sketch is just $CM(\vec{a})$. The second sketch is the CM sketch on a vector with $\frac{m}{2}$ components, each being the sum of a_i's over all i in a dyadic interval $I_{1,k}$ where $0 \leqslant k \leqslant \frac{m}{2} - 1$. The third up to the $\log m$th sketch are CM sketches of $\vec{a}$ 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 $(\phi + \epsilon)\|\vec{a}\|_1$. 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 $(\phi + \epsilon)\|\vec{a}\|_1$. Note that in each sketch, we only need to examine at most $\frac{2}{\phi}$ dyadic intervals.

As for updates, note that each point in the universe $\{1, \ldots, m\}$ is a member of $\log m$ sketches. Thus when an update on that point arrives, each of these sketches is updated.

Theorem 2.2. *(See [10].) For any $\vec{a} = (a_1, \ldots, a_m)$ and $\epsilon, \mu, \phi > 0$, the collection of sketches $CCM(\vec{a})$ has size $h = O(\frac{1}{\epsilon} \log m \log(\frac{\log m}{\mu\phi}))$. In $O(\frac{\epsilon h}{\phi})$ time, it can report all the components with weight at least $(\phi + \epsilon)\|\vec{a}\|_1$, and with probability $1 - \mu$, it reports no component with weight less than $\phi\|\vec{a}\|_1$. The sketch can be updated in $g = O(\log m \log(\frac{\log m}{\mu\phi}))$ time and constructed in $O(gm + h)$ time.*

Lemma 2.1. *If for all i, $a_i \pm b_i \geqslant 0$, then $CM(\vec{a} \pm \vec{b}) = CM(\vec{a}) \pm CM(\vec{a})$.*

2.3. l_0-Sketch

When the magnitude of each component of $\vec{a}$ is bounded from above by some value U, we can compute its l_0-sketch [9], denoted $L_0(\vec{a})$, which allows us to approximate, $\|\vec{a}\|_0$, the number of non-zero entries in $\vec{a}$.

Theorem 2.3. *(See [9].) For any $\vec{a} = (a_1, \ldots, a_m)$ such that $|a_i| \leqslant U$ for every i, and any $\epsilon, \mu > 0$, the l_0-sketch of $\vec{a}$, $L_0(\vec{a})$, has size $h = O(\frac{1}{\epsilon^2} \log \frac{1}{\mu})$. Using the sketch, the l_0-norm, $\|\vec{a}\|_0$, can be approximated to within a factor of $1 \pm \epsilon$ with probability $1 - \mu$ in $O(h)$ time. Further, it can be updated in $O(h)$ time and with knowledge of U, it can be constructed in $O(hm)$ time.*

Lemma 2.2. $L_0(\vec{a} \pm \vec{b}) = L_0(\vec{a}) \pm L_0(\vec{a})$.

3. Approximate colored range queries

In this section, we consider the problem of colored range queries. We start by illustrating 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 can, for example, 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 ϵ 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 ϵ as a fixed constant). Applying standard techniques [5], 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 summaries 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, support queries in $O(d^2 \log^{d+1} n)$ and perform 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.

3.1. 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 $\vec{a} = (a_1, a_2, \ldots, a_m)$ 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 $\vec{a}$, or formally the l_0-norm of $\vec{a}$. In general m can be very large and hence storing an $\vec{a}$ explicitly for each node u will be very space-consuming. Therefore, we will store the l_0-sketch of $\vec{a}$ ($L_0(\vec{a})$) so that we only need $h = O(\frac{1}{\epsilon^2} \log(\frac{1}{\mu}))$ space per sketch. (The value of ϵ and μ will be determined later.) Hence in total we need $O(hn)$ space.

When constructing $L_0(\vec{a})$, we need to have an upper bound U on the magnitude of each component a_i of $\vec{a}$. Obviously, $|a_i| \leqslant n$. To allow for changes in n, we actually set U to a larger value and periodically rebuild the structures when n gets too large or too small. Details will be given shortly.

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 β. Then for each node v along the path from the root to x, its sketch has to be updated. Let $\vec{a}$ be the color frequency vector associated with v. Then inserting (deleting) the point will cause a_i to increase (decrease) by β. Let $\vec{b} = (b_1, b_2, \ldots, b_m)$ such that $b_i = \beta$ and $b_j = 0$ for all $j \neq i$. Then

$L_0(\vec{a})$ should be updated to $L_0(\vec{a}) \pm L_0(\vec{b})$ 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 [2,14] 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 [23].)

Since the intrinsic color vectors in the tree may change during updates, we have to ensure that our upper estimate, U, on the maximum magnitude of any component of the color vector remains valid so that the l_0-sketches continue to give accurate answers. Suppose initially when we build the structure, there are $n \leqslant c$ points for some constant c. Then we initially set $U = 2c$. If n grows to $2c$ after a series of updates, we rebuild the whole structure and the sketches with U reset to $U = 2n = 4c$. In general, we will maintain that $\frac{U}{2} \leqslant n \leqslant U$ and whenever n becomes too large or too small, we double or half U respectively. Thus $U \geqslant n \geqslant \|\vec{a}\|_1 \geqslant |a_i|$ for all i at all times. Since the rebuilding takes $O(hn)$ time and occurs only after at least n updates, we have the amortized update time of $O(h \log n)$.

Storage reduction. We can reduce the space requirement of the data structure to be independent of ϵ and μ. 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 $O(\frac{n}{h})$ 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 $\vec{a}$ for only the points lying within the query region. This takes $O(h)$ time by definition of F. Then we construct $L_0(\vec{a})$ using $O(h^2)$ time. The total query time now becomes $O(h \log n + h^2)$. When inserting a new data point x, we first add a leaf node v representing x in T using $O(\log n)$ time. We then update the sketches of nodes that are lying along the path from root to v and above the frontier. This takes $O(h \log n)$ and hence the total update time.

Choosing μ. Now we consider the value of μ (the failure probability of a sketch). 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 the data structure fails with probability δ. That is, with probability $1 - \delta$, all the $O(n^2)$ sketches give good enough approximations. Thus, we set $\mu = \frac{\delta}{n^2}$ and so $h = O(\frac{1}{\epsilon^2} \log \frac{n^2}{\delta})$. In general for d dimensions, number of query regions become $O(n^{2d})$ and so $h = O(\frac{1}{\epsilon^2} \log \frac{n^{2d}}{\delta}) \leqslant O(\frac{d}{\epsilon^2} \log \frac{n}{\delta})$. Since the size of the sketches now depends on n, we may have to change the size of the sketches as n changes. The periodic rebuilding of the sketches has already been discussed in previous paragraphs.

3.2. 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 $\vec{a} = (a_1, a_2, \ldots, a_m)$ 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 using $CM(\vec{a})$ for estimating individual component, we will use it to estimate the minimum value of a component of $\vec{a}$. We do this by finding the minimum value in each row of $CM(\vec{a})$ and then taking the minimum among the $\lceil \ln \frac{1}{\mu} \rceil$ rows. In other words, we find the minimum value within the whole sketch $CM(\vec{a})$.

There is a small technical issue. It is possible that some cells in $CM(\vec{a})$ are not hashed into by any index $i \in \{1, \ldots, m\}$. Then our method will return a zero value (the initial value of a cell in the sketch). 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 $\vec{a}$ 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 in the cell. When computing the minimum value in the sketch, we only consider those non-negative cells. The following lemmas prove that the estimated minimum is accurate with high probability.

Lemma 3.1. *(See [10].) For* $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, \lceil \ln \frac{1}{\mu} \rceil\}$, $E[count(j, h_j(i)) - a_i] \leqslant \frac{\epsilon}{e}\|a\|_1$.

Lemma 3.2. *Let* $a_t = \min_{i=1}^{m}\{a_i\}$. *For all* j, *define*

$$m_j = \min_{i' \text{ s.t. } count(j,i') \geqslant 0} \{count(j, i')\}.$$

Then $E[m_j - a_t] \leqslant \frac{\epsilon}{e}\|\vec{a}\|_1$.

Proof. By definition, $m_j \leqslant count(j, h_j(t))$ for all j, and thus $E[m_j - a_t] \leqslant E[count(j, h_j(t)) - a_t]$. Then apply Lemma 3.1. □

Lemma 3.3. *Let* $a_t = \min_{i=1}^{m}\{a_i\}$. *Define* $cmm = \min_j\{m_j\}$. *Then* $cmm \geqslant a_t$ *and* $cmm \leqslant a_t + \epsilon\|\vec{a}\|_1$ *with probability* $1 - \mu$.

Proof. Note that $count(j, h_j(i)) \geqslant a_i \geqslant a_t$ for $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, \lceil \ln \frac{1}{\mu} \rceil\}$. By definition, $m_j \geqslant a_t$ for all j, thus $cmm \geqslant a_t$. By Markov inequality and Lemma 3.2,

$$Pr\big[cmm > a_t + \epsilon\|\vec{a}\|_1\big] = Pr\big[\forall_j . m_j - a_t > \epsilon\|\vec{a}\|_1\big] \leqslant Pr\big[\forall_j . m_j - a_t > eE[m_j - a_t]\big] < e^{-\lceil \ln \frac{1}{\mu} \rceil} \leqslant \mu. \qquad \square$$

We know that to store a CM sketch we need $h = O(\frac{1}{\epsilon}\log\frac{1}{\mu})$ space. Hence, 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 3.1 and the only difference is the update time. Note that we need only $O(g) = O(\log\frac{1}{\mu})$ 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 μ. If we just store the CM sketches for nodes above the *frontier* F as defined in Section 3.1, $h = O(\frac{1}{\epsilon}\log\frac{1}{\mu})$ 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 $\mu = n^{-\Omega(1)}$ and taking ϵ as constant, the asymptotic complexity of query and update times are both $O(\log^2 n)$. Moreover, the failure probability of the data structure is $n^{-\Omega(1)}$.

3.3. Heavy colors

Construction and storage. Our main result in this subsection 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 $\vec{a} = (a_1, \ldots, a_m)$ 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 $\vec{a}$ to be non-negative. Our problem then is to report those colors with $a_i \geqslant \phi\|\vec{a}\|_1$, where $\phi \in [0, 1]$. We store for each node u the collection of sketches $CCM(\vec{a})$ for the vector $\vec{a}$. Each of the sketches represents a level of dyadic intervals (as described in Section 2.2). In this setting, we require only $h = O(\frac{\log m}{\epsilon}\log(\frac{\log m}{\mu\phi}))$ space per node. Thus, the total space needed is $O(hn)$ for the whole tree T.

Using similar idea as in the previous subsection, the construction time of sketches for the whole tree takes $O(hn)$ time. Querying takes $O(h \log n + \frac{\epsilon h}{\phi}) = O(h \log n)$ time according to Theorem 2.2 and since ϕ 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 $O(h \log n)$.

4. Approximate colored point enclosure queries

In this section, we consider the problem of colored point enclosure queries. Our main results are some dynamic approximate solutions to the colored point enclosure counting problem. They serve not only as solutions to the specific colored point enclosure counting problem but also the class of colored point enclosure query problems in general. By replacing the l_0-sketch with the appropriate summary structures, the framework can be employed to solve other colored point enclosure problems such as colored point enclosure min and colored point enclosure heavy problems. For simplicity reason, we ignore the specific solutions to different colored point enclosure problems and focus on the colored point enclosure counting problem in this section.

We have devised two solutions to the colored point enclosure counting problem and they represent a tradeoff between space and query time. The two approaches are based on two different traditional point enclosure structures, namely the segment tree and interval tree.

In the segment tree approach, we employ the l_0-sketch (Theorem 2.3) that can answer the number of non-zero components of the color vector $\vec{a}$. Please be reminded that l_0-sketch has size of $h = O(\frac{1}{\epsilon^2}\log(\frac{1}{\mu}))$ and can be updated and constructed in $O(h)$ and $O(hm)$ time respectively. The sketch parameters ϵ and μ specify the relative error and failure probability of the sketch respectively. The number of colors is assumed to be very large so that $h \ll m$ and we enjoy the size independence of m.

In the interval tree approach, we will make use of the 1-dimensional colored range counting structures detailed in Section 3.1 as auxiliary structures.

4.1. Colored point enclosure counting with segment tree

Construction and storage. We consider the 1-dimensional case and the result can be readily generalized to higher dimensions using standard technique [5]. We construct a segment tree [4], T, on the distinct coordinates of the left and right endpoints of the n colored intervals (1-d data regions). Let $p(v)$ be the parent of node v and $mid(v)$ be the midpoint that separates the leftmost endpoint contained in the right subtree of v from the rightmost endpoint of the left subtree of v, then the range of v, $range(v) = [range_l(v), range_r(v)]$, is defined as follows: if v is the root, $range(v) = [-\infty, \infty]$; otherwise if v is the left child of $p(v)$, $range(v) = [range_l(p(v)), mid(p(v))]$; otherwise, $range(v) = [mid(p(v)), range_r(p(v))]$. We associate an interval $[x_1, x_2]$ to a node $v \in T$ if the interval contains the $range(v)$ but not those of the ancestors of v. Note that any interval is associated with at most $O(\log n)$ nodes and given a query point q, we can obtain the set of intersecting intervals as the union of sets of associated intervals of nodes along the path traversed with q.

Then for each node v we define a frequency vector $\vec{a} = (a_1, a_2, \ldots, a_m)$ so that each a_i represents the number of occurrences of color i in the associated set. Instead of storing the whole vector $\vec{a}$, we store for each node an l_0-sketch of $\vec{a}$. This is because we need to report the number of non-zero components of the vector $\vec{a}$ for each node v, i.e. the l_0-norm of $\vec{a}$ to solve the colored point enclosure counting problem.

Therefore, we need for each node an l_0-sketch of size $h = O(\frac{1}{\epsilon^2}\log(\frac{1}{\mu}))$. The size of the sketches for the whole tree is thus $O(hn)$. For the purpose of supporting updates, we need to store in each node a list of associated intervals, the size of which is $O(n \log n)$. The space requirement of the whole structure becomes $O(n(\log n + h))$.

We can construct the structure by first building the base tree and initializing the sketch with a zero vector for each of the nodes, this takes $O(hn)$ time. Then we update the corresponding sketches for each of the intervals, this step requires $O(\frac{n \log n}{\epsilon^2}\log(\frac{1}{\mu})) = O(hn \log n)$. Hence, the total construction time is $O(hn \log n)$.

Querying. Given a query point q, we traverse the tree with q and we add together the l_0-sketches of the nodes along the search path. Due to the additivity of l_0-sketches, the resulting sketch represents a color frequency vector for the set of all intervals intersecting with q. The traversal and additions require $O(h \log n)$ time and hence the query. We ignore the insubstantial time for querying the resulting sketch.

Updating. Suppose we are inserting/deleting an interval and this update does not incur changes on the structure of the base tree. The update can be done readily by obtaining the set of nodes that are associated with the interval in $O(\log n)$ time and then inserting into/deleting from the sketches of those nodes the color of the interval. A total of $O(h \log n)$ time is required.

However, if such an update requires a change in the structure, we have to rebuild the lists of associated intervals and hence the sketches of the affected nodes in tree rotations. We define the weight of a node v, $w(v)$, as the number of

leaves in the subtree rooted at v. It can be shown that the number of associated intervals of v is bounded by $O(w(v))$, and thus the time for rebuilding the auxiliary structures for an affected node v in rotation is $O(hw(v))$. Applying the Willard and Lueker's dynamic transformation [23], one can conclude than the worst case update time is $O(h \log n)$.

To ensure that the data structure is good for all queries with probability at least $1-\delta$, we choose μ (the failure probability of one sketch) to be $\mu = \frac{\delta}{n^2}$ for the reason mentioned in Section 3.1. The sketch size h becomes $O(\frac{1}{\epsilon^2} \log \frac{n}{\delta})$ and is dependent on the number of intervals which changes in a dynamic context. Instead of changing the sketch size every time n changes, we build the structure with sketch size doubled initially and rebuild the whole structure if the current size doubles or becomes half of the initial one as mentioned in Section 3.1. We also have to reset the U parameters of the l_0-sketches as well during rebuildings to ensure the sketches continue to produce good results. Such rebuildings take $O(hn \log n)$ time and since they occur only after at least $O(n)$ updates, the amortized update time is $O(h \log n)$.

4.2. Colored point enclosure counting with interval tree

Construction and storage. We now describe the 1-dimensional structure. This time we build the base tree as an interval tree [11,12]. The interval tree is a binary search tree, T, built on the endpoints of the n intervals. Each node $v \in T$ in the interval tree has a midpoint value $mid(v)$ that separate the leftmost endpoint contained in the right subtree of v from the rightmost endpoint of the left subtree of v. We associate an interval $[x_1, x_2]$ with the highest node whose midpoint is contained in the interval. Note that every interval is assigned to at most one node. For each node v, we store a left list $L(v)$ of all associated intervals sorted with their left endpoints and another right list $R(v)$ defined in a similar way. In addition, we build two 1-dimensional colored range counting structures ($LC(v)$ and $RC(v)$) (described in detail in Section 3.1), one for each of the lists, on those endpoints in the lists. The sketches of such structures are randomized with failure probability $\frac{\delta}{n^{\Omega(1)}}$ as mentioned in the previous chapter. Hence they answer queries in $O(\frac{1}{\epsilon^4} \log^2 \frac{n}{\delta})$ time and support updates in $O(\frac{\log n}{\epsilon^2} \log \frac{n}{\delta})$ time with linear space. Since each interval is assigned to only one node and one set $LC(v)$ and $RC(v)$, the whole structure requires only linear space.

The structure is constructed by first building the base tree and then inserting the intervals into the structure. The insertion requires first finding the associated node v with $O(\log n)$ time and then inserting into the 1-d colored range counting structures $LC(v)$ and $RC(v)$ with $O(\frac{\log n}{\epsilon^2} \log \frac{n}{\delta})$ time. The construction of the whole structure thus requires $O(\frac{n \log n}{\epsilon^2} \log \frac{n}{\delta})$ time.

Querying. With a query point q, we traverse the tree and obtain all the nodes along the traversed path. Then for each of these nodes v, we do the following: if $q < mid(v)$ we query the left 1-dimensional colored range counting structure, $LC(v)$, with the range $[-\infty..q]$ and obtain the resulting sketch, i.e. the sketch that summarizes the color frequency vector for the subset of $LC(v)$ lying in $[-\infty..q]$; otherwise, we query the 1-d range counting structure $RC(v)$ with the query range $[q..\infty]$ and obtain the result. Note that all the intervals as represented by this resulting sketch intersects with q. Querying such a structure requires $O(\frac{1}{\epsilon^4} \log^2 \frac{n}{\delta})$ time and because there are at most $O(\log n)$ nodes on the traversed path, it takes in total $O(\frac{\log n}{\epsilon^4} \log^2 \frac{n}{\delta})$ time. We then find the union of all the resulting sketches in $O(\frac{\log n}{\epsilon^2} \log \frac{n}{\delta})$ time and obtain the count estimate from the union as the result. The total query time is thus $O(\frac{\log n}{\epsilon^4} \log^2 \frac{n}{\delta})$ time.

Updating. Insertion of an interval requires first adding the endpoints of the interval to the base tree. Then we add the interval into appropriate auxiliary structures by first finding its associated node v with $O(\log n)$ time and then adding it into the $LC(v)$ and $RC(v)$ structures with $O(\frac{\log n}{\epsilon^2} \log \frac{n}{\delta})$ time. Hence, we need $O(\frac{\log n}{\epsilon^2} \log \frac{n}{\delta})$ update time in total. Using similar arguments for the case in segment tree, we can obtained $O(\frac{\log n}{\epsilon^2} \log \frac{n}{\delta})$ update time in case where rotations are required.

As the size of the auxiliary structures changes with the number of intervals n, we build the range counting structures with size doubled so that we can get a final amortized update time of $O(\frac{\log n}{\epsilon^2} \log \frac{n}{\delta})$.

5. Conclusion

5.1. Summary of the work

We have formulated a new class of colored range query problems that includes colored range reporting, counting, min and traditional range sum/counting, etc. Many of these problems are difficult to solve using traditional data

structural techniques and we propose a general approach of using sketches to solve them approximately. We believe that there are two benefits of proposing the general method. First, advancements in the research on sketches or other summary structures having the required properties mentioned in the start of Section 3 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. For example, our colored range counting technique works on R-tree [16] and its variants [3,21].

To demonstrate other possibilities in the use of sketches in solving related problems, we investigated the problem of colored point enclosure. We proposed two approaches of building structures with sketches to solve the colored point enclosure counting problem approximately. The results enjoy the same benefits of those of the colored range queries and are to be generalized to solve other colored point enclosure problems by employing the appropriate summary structures.

5.2. Future directions

There are several directions for our future research work. First, the possibilities of employing other summary structures in solving related problems can be explored. Interesting problems include colored range max, range quantiles and their corresponding point enclosure problems. Until now, very little work has been done in solving these problems directly or providing the appropriate sketches for our framework.

Next, this research can be extended to context in which the colors form a hierarchy. Currently, we have focused on solving problems where no relationship exist between colors. In some applications, colors can be organized in a hierarchical manner such that colors at a lower-level of hierarchy are grouped into higher-level ones. A possible approach in solving the problem is to build an instance of our approximate solution for each level of hierarchy. More efficient solutions are yet to be researched.

Another direction for future work is to consider applications with more than one color dimensions. One limitation of our work is that we have considered only one dimension of colors. With two or more color dimensions, different types of queries can be posed on various color dimensions. For example, we can query the colored range counting on one dimension while at the same time asking for the colored range max on the other. The problem may require solutions in which sketches are embedded within other sketches.

Acknowledgement

The authors thank the anonymous referees for their helpful comments on improving the presentation of the paper.

References

[1] P.K. Agarwal, S. Govindarajan, S. Muthukrishnan, Range searching in categorical data: colored range searching on grid, in: 10th European Symposium on Algorithms, 2002, pp. 17–28.
[2] R. Bayer, Symmetric binary B-trees: Data structures and maintenance algorithms, Acta Inform. 1 (1972) 290–306.
[3] N. Beckmann, H.P. Kriegel, R. Schnieder, B. Seeger, The R*-tree: An efficient and robust access method for points and rectangles, in: ACM SIGMOD Conference on the Management of Data, 1990, pp. 322–331.
[4] J.L. Bentley, Algorithms for Klee's rectangle problems, Department of Computer Science, Carnegie Mellon Univ., unpublished notes, 1977.
[5] J.L. Bentley, Multidimensional divide-and-conquer, Comm. ACM 23 (4) (1980) 214–228.
[6] M. Berg, H.J. Haverkort, Significant-presence range queries in categorical data, in: Workshop on Algorithms and Data Structures, in: Lecture Notes in Computer Science, vol. 2748, Springer, 2003, pp. 462–473.
[7] J.L. Carter, M.N. Wegman, Universal classes of hash functions, J. Comput. Syst. Sci. 18 (1979) 143–154.
[8] B. Chazelle, A functional approach to data structures and its use in multidimensional searching, SIAM J. Comput. 17 (3) (1988) 427–462.
[9] G. Cormode, M. Datar, P. Indyk, S. Muthukrishnan, Comparing data streams using Hamming norms (how to zero in), in: Proceedings of the 28th International Conference on Very Large Data Bases, 2002, pp. 335–345.
[10] G. Cormode, S. Muthukrishnan, An improved data stream summary: the count-min sketch and its applications, in: The 6th Latin American Theoretical Informatics (LATIN), in: Lecture Notes in Computer Science, vol. 2976, Springer, 2004, pp. 29–38.
[11] H. Edelsbrunner, A new approach to rectangle intersections, part i, Int. J. Comput. Math. 13 (1983) 209–219.
[12] H. Edelsbrunner, A new approach to rectangle intersections, part ii, Int. J. Comput. Math. 13 (1983) 209–219.
[13] P. Ferragina, N. Koudas, D. Srivastava, S. Muthukrishnan, Two-dimensional substring indexing, in: Proceedings of the 20th Annual ACM Symposium on Principles of Database Systems, 2001, pp. 282–288.
[14] L.J. Guibas, R. Sedgewick, A dichromatic framework for balanced trees, in: 19th Annual Symposium on Foundations of Computer Science, IEEE, 1978, pp. 8–21.

[15] P. Gupta, R. Janardan, M. Smid, Further results on generalized intersection searching problems: Counting, reporting, and dynamization, J. Algorithms 19 (1995) 282–317.
[16] A. Guttman, R-trees: A dynamic index structure for spatial searching, in: ACM SIGMOD Conference on the Management of Data, 1984, pp. 47–57.
[17] R. Janardan, M. Lopez, Generalized intersection searching problems, Internat. J. Comput. Geom. Appl. 3 (1) (1993) 39–69.
[18] Y.K. Lai, C.K. Poon, B.Y. Shi, Approximate colored range queries, in: 16th Symposium on Algorithms and Computation (ISAAC'05), Sanya, China, in: Lecture Notes in Computer Science, vol. 3827, Springer, 2005, pp. 360–369.
[19] S. Muthukrishnan, Efficient algorithms for document retrieval problems, in: Proceedings of the Thirteenth Annual ACM–SIAM Symposium on Discrete Algorithms, 2002, pp. 657–666.
[20] A. Nanopoulos, P. Bozanis, Categorical range queries in large databases, in: Proc. 8th Int. Symp. Spatial and Temporal Databases (SSTD 2003), in: Lecture Notes in Computer Science, vol. 2750, Springer, 2003, pp. 122–139.
[21] T. Sellis, N. Roussopoulos, C. Faloutsos, The R+-tree: A dynamic index for multi-dimensional objects, in: Proceedings of the 13th International Conference on Very Large Data Bases, Morgan-Kaufmann, 1987, pp. 507–518.
[22] D.E. Willard, New data structures for orthogonal queries, SIAM J. Comput. 14 (1) (1985) 232–253.
[23] D.E. Willard, G.S. Lueker, Adding range restriction capability to dynamic data structures, J. ACM 32 (3) (1985) 597–617.