Range Aggregate Structures for Colored Geometric Objects.

Saladi Rahul* Haritha Bellam† Prosenjit Gupta‡ K. S. Rajan§

Abstract

A set of n colored objects (points/hyperboxes) lie in $\mathbb{R}^d$. Each object has a weight associated with it. Given a query orthogonal range q, for each distinct color c of the objects in $S \cap q$, the tuple $\langle\ c,\ \mathcal{F}(c)\ \rangle$ is reported where $\mathcal{F}(c)$ is the object of color c with the maximum weight.

1 Introduction

Range Searching and its variants have been widely studied in the field of Computational Geometry. In many applications a more general form of these problems arise. The objects in S come aggregated in disjoint groups and of interest are questions regarding the intersection of q with the groups rather than with the objects. For convenience we shall associate a distinct color for each group and assume that all the objects in the group have that color. These class of problems are referred to as Generalized Intersection Searching. [6] is a survey paper on the latest results on this topic.

In many applications like on-line analytical processing (OLAP), geographic information systems (GIS) and information retrieval (IR), aggregation plays an important role in summarizing information [12] and hence large number of algorithms and storage schemes have been proposed to support such queries. In *range-aggregate query* problems [12] many composite queries involving range searching are considered, wherein one needs to compute the aggregate function of the objects in $S \cap q$ rather than report all of them as in a range reporting query. In [9], range-aggregate problems were solved on colored objects.

This paper presents some results on range-aggregate queries on colored objects. The general problem statement is the following : Preprocess a set S of weighted colored geometric objects in $\mathbb{R}^d$, $d = 1, 2$, such that given a query orthogonal range q, we can report efficiently for each distinct color c of the objects in $S \cap q$, the tuple $\langle c, \mathcal{F}(c)\rangle$ where $\mathcal{F}(c)$ is the object of color c with the maximum weight. Section 2 and Section 3 deal with these problems. The uncolored version of these problems were discussed in [1].

We define a couple of terms. Consider two points $p(p_1,\ p_2,\ \ldots,\ p_d)$ and $q(q_1,\ q_2,\ \ldots,\ q_d)$. If $p_i > q_i$, $\forall\ 1 \leq i \leq d$ then p is defined to be *dominating* q and q is defined to be *dominated* by p. Given a point set S, a point $q \in S$ is called a *maximal point* iff q is not dominated by any other point in S.

2 Generalized Orthogonal Range-Max query

Problem: S is a set of n colored objects in $\mathbb{R}^d$ where each point p is assigned a weight $w(p)$. S_c is the set of points of S having color c. We wish to preprocess S into a data structure so that given a query orthogonal hyperbox $q = \Pi_{i=1}^{d}[a_i, b_i]$, we can report for each distinct color c of the points in q, the tuple $\langle\ c,\ p_c\ \rangle$ where p_c = max $\{w(p_c) \mid p_c \in S_c$ and $p_c \in$ q$\}$, i.e., the point in $S_c \cap q$ with the *topmost/maximum* weight.

This problem can be solved by modifying the solution to the range-aggregate problem on colored points where the function $\mathcal{F}(c)$ was the sum of the weights of the points of color c in q [9]. The *sum* function is replaced by *max* function. This leads to a solution of space $O(n^{1+\epsilon})$ and $O(\log n + k)$ query time for $d \geq 2$. However, in this section we shall come up with a solution which reduces the space needed to $O(n \log^2 n)$ while retaining the same query time for d=2.

2.1 Quadrant query, $q = [a_1, \infty) \times [a_2, \infty)$

In this subsection we shall consider the problem for quadrant queries, $q = [a_1, \infty) \times [a_2, \infty)$. Consider points $p(p_x,\ p_y)$ and $r(r_x,\ r_y)$ both having the same color c. Let $r_x > p_x$, $r_y > p_y$ and $w(r) > w(p)$. For an arbitrary query q, if p lies in q then r will also lie in q and since $w(r)$ is greater than $w(p)$, point p cannot have the maximum/topmost weight among $S_c \cap q$. Hence, such points are can be removed from consideration.

In order to remove the points of S which cannot be candidates for maximum/topmost weight, we do the following: Fix a color c. Map each point $p(p_x,\ p_y) \in S_c$ to a three-dimensional point $p'(p_x,\ p_y,\ w(p))$. Call this new set of transformed points S'_c. Maximal points, M'_c, of S'_c are found out in $\mathbb{R}^3$. This can be done in time $O(|S'_c| \log^2 |S'_c| + |M'_c|)$. M'_c represents the set of points from S'_c (or S_c) which are possible

*Lab for Spatial Informatics, IIIT-Hyderabad, India, saladi.rahul@gmail.com

†Lab for Spatial Informatics, IIIT-Hyderabad, India, pinki.b.haritha@gmail.com

‡Yahoo! Research and Development, Bangalore 560093, India, prosenjit_gupta@acm.org

§Lab for Spatial Informatics, IIIT-Hyderabad, India, rajan@iiit.ac.in

candidates for topmost/maximum weight for color c. This process is repeated for each color c. Denote $M' = \bigcup_{\forall c} M'_c$. The total time taken for finding M' will be $O(\Sigma_{\forall c}(|S'_c| \log^2 |S'_c| + |M'_c|)) \equiv O(\log^2 n \times \Sigma|S'_c| + \Sigma|M'_c|) \equiv O(n \log^2 n + |M'|) \equiv O(n \log^2 n)$ since $|M'| \leq n$.

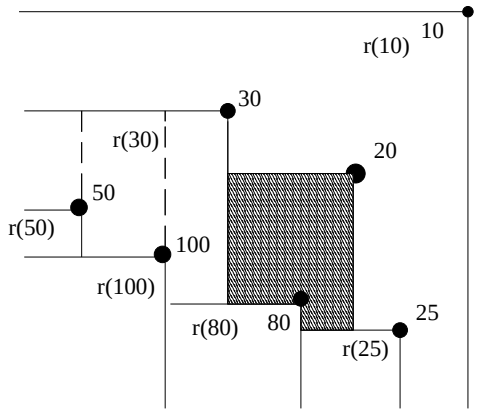

Figure 1: For a particular color c, set M_c is being shown. These points are the only candidates from color c which can have the topmost/maximum weight for a Generalized Orthogonal Range-Max query.

Once again fix a color c. New we shift our attention from S_c to M'_c. Each point $p'(p_x, p_y, w(p)) \in M'_c$ is mapped back to its original two-dimensional point $p(p_x, p_y)$ with weight $w(p)$. Call this set M_c. Notice that all the points in M_c need not be maximal w.r.t. to the two-dimensional plane, though they were all maximal points in three-dimensional space as part of set M'_c (as shown in Figure 1). This process is repeated for each color c.

For each color c, set M_c is divided into *layers of maximal points* in $\mathbb{R}^2$. Layer 1 is denoted by M_c^1 which is nothing but the set of maximal points of M_c in $\mathbb{R}^2$. Layer l, M_c^l ($l > 1$), is defined to be the set of maximal points of $M_c \setminus \bigcup_{j=1}^{l-1} M_c^j$ in $\mathbb{R}^2$. These layers are defined until we reach an empty layer. In Figure 1, we show an example of a set M_c having seven points. The weight associated with each point is also shown. For the purpose of discussion, each point is uniquely referred to by its weight. In this example M_c gets divided into three layers of maximal points. Layer 1, $M_c^1 = \{10\}$; Layer 2, $M_c^2 = \{30, 20, 25\}$ and Layer 3, $M_c^3 = \{50, 100, 80\}$.

These "layers of maximal points" for each set M_c are obtained as follows: Based on the points in M_c we build a data structure T described in [10]. T maintains the set of maximal points in the plane of set M_c. Initial building of the structure takes $O(|M_c| \log |M_c|)$ time. Insertion or Deletion of a point is handled in $O(\log |M_c|)$ amortized time. The reporting of maximal points takes $O(r)$ time where r is the number of maximal points. Using T we can directly find out the set of maximal points of M_c which constitutes M_c^1. Now all the points in M_c^1 are deleted from T. Now the maximal points reported by T will be M_c^2. Next all the points in M_c^2 are deleted from T. This process is repeated iteratively till T becomes empty. The time taken for finding layers of M_c will be $O(|M_c| \log |M_c|)$. Total time taken for finding all sets M_c will be $O(n \log n)$.

For a query quadrant $q = [a_1, \infty) \times [a_2, \infty)$, call ($a_1$, a_2) to be an apex point of q. Now for each point $p \in M_c$, we shall define a region $r(p)$. If (a_1, a_2) stabs a particular region $r(p)$ then the point p corresponding to that region will have the maximum weight among $S_c \cap q$. Let M_c have l layers of maximal points. We start with M_c^l (layer l) and go till M_c^1 (layer 1). The points in M_c^l are sorted in decreasing order based on their weights. The first point in the list, $p(p_x, p_y)$, is assigned the region $r(p) = (-\infty, p_x] \times (-\infty, p_y]$. The i^{th} point in the list M_c^l is assigned the region $r(p) = (-\infty, p_x] \times (-\infty, p_y] \setminus \bigcup r(p')$, where the union is over the first i - 1 points in the list M_c^l. See M_c^3 (layer 3) in Figure 1 on how $r(100)$, $r(80)$ and $r(50)$ have been assigned. Let $R(l') = \bigcup r(p)$, where the union is over all the points in $M_c^{l'}$. For layers above M_c^l we do the following: Suppose we are at a layer l'. Then we sort the points in $M_c^{l'}$ in decreasing order of their weights. Then the i^{th} point $p(p_x, p_y)$ in $M_c^{l'}$ is assigned the region $r(p) = (-\infty, p_x] \times (-\infty, p_y] \setminus \bigcup_{j=l'+1}^{l} R(j) \setminus \bigcup r(p')$, where the union over $r(p')$ is the first i - 1 points in the list $M_c^{l'}$. The region associated with each point in M_c is shown in Figure 1. The shaded region shows the region associated with point 20. So given a query q, we need to check in which region (a_1, a_2) lies and the point of M_c corresponding to that region has to be reported.

If the region $r(p)$ associated with point $p \in M_c$ is not in the form of an axis-parallel rectangle, then $r(p)$ is broken into axis-parallel rectangles (see r(30) in Figure 1). So, for each color c we have a collection of disjoint rectangles χ_c. $\chi_c \equiv O(|M_c|)$, since from each point in M_c at most three rays are shooting out (see Figure 1) and it is a planar surface. Based on the rectangles χ_c obtained for each color c we build an instance of the structure D in [3]. Given a query point, D reports all the rectangles stabbed by the query point. The number of rectangles stored in D will be $O(n)$.

Finally, when we are given a query quadrant $q = [a_1, \infty) \times [a_2, \infty)$, we shall query D with (a_1, a_2) and for each rectangle that gets stabbed, the point that corresponds to that rectangle along with its weight is reported. Note that at most only one rectangle of each color is reported.

Theorem 1 *A set of n colored points in $\mathbb{R}^2$ can be stored in a linear-size structure such that given a query quadrant $q = [a_1, \infty) \times [a_2, \infty)$, a generalized orthogonal range-max query can be answered in $O(\log n + k)$ time.*

2.2 Bounded rectangular query

In this subsection we solve the problem for bounded orthogonal query rectangle q=$[a_1, b_1] \times [a_2, b_2]$. The solution is based on the quadrant range-max structure of Theorem 1. We first show how to solve the problem for query rectangles $q' = [a_1, b_1] \times [a_2, \infty)$. In this discussion, NE-query would mean $[a_1, \infty) \times [a_2, \infty)$ and NW-query would mean $(-\infty, b_1) \times [a_2, \infty)$. We store the points of S in sorted order by x-coordinate at the leaves of a balanced binary tree T'. At each internal node v, we store an instance of the structure of Theorem 1 for NE-queries (resp., NW-queries) built on the points in v's left (resp., right) subtree. Let $X(v)$ denote the average of the x-coordinate in the rightmost leaf in v's left subtree and the x-coordinate in the leftmost leaf of v's right subtree.

To answer a query q', we do a binary search down T', using $[a_1, b_1]$ until a highest node v is reached such that $[a_1, b_1]$ intersects $X(v)$. If v is a leaf, then if $v's$ point is in q' we report its color. If v is a non-leaf, then we query the structures at v using the NE-quadrant and the NW-quadrant derived from q' (i.e., the quadrants w.r.t. points at (a_1, a_2) and (b_1, a_2), respectively), and then combine the answers. The space occupied by T' becomes $O(n \log n)$ and the query time remains $O(\log n + k)$.

To solve the problem for general query rectangles $q = [a_1, b_1] \times [a_2, b_2]$, we use the above approach again, except that we store the points in the tree sorted by y-coordinates. At each internal node v, we store an instance of the data structure above to answer queries of the form $[a_1, b_1] \times [a_2, \infty)$ (resp., $[a_1, b_1] \times (-\infty, b_2]$) on the points in v's left (resp., right) subtree. The query strategy is similar to the previous one, except that we use the interval $[a_2, b_2]$ to search in the tree. The space increases by a log factor though the query time remains the same.

Theorem 2 *A set of n colored points in $\mathbb{R}^2$ can be stored in a structure of size $O(n \log^2 n)$ such that given a query rectangle $q = [a_1, b_1] \times [a_2, b_2]$, a generalized orthogonal range-max query can be answered in $O(\log n + k)$ time.*

2.3 Solution for $d = 1$

In this subsection we shall solve the problem for $d = 1$. Each point $p(p_x) \in S$ is mapped to a 2-dimensional point $p'(p_x, -p_x)$. Call the new set of points S'. Given a query interval q=$[a_1, b_1]$, it is mapped to a quadrant q'= $[a_1, \infty) \times [-b_1, \infty)$. So, the problem has been mapped to the quadrant query problem in two-dimensional plane, where S' is the set of points and q' is the query quadrant. Hence, the structure in Theorem 1 can be used to solve this problem.

Theorem 3 *A set of n colored points in $\mathbb{R}^1$ can be stored in a linear-size structure such that given a query interval q=$[a_1, b_1]$, a generalized orthogonal range-max query can be answered in $O(\log n + k)$ time.*

3 Generalized orthogonal stabbing-max query

Problem: S is a set of n colored hyperboxes in $\mathbb{R}^d$ where each hyperbox γ is assigned a weight $w(\gamma)$. S_c is the set of hyperboxes of S having color c. We wish to preprocess S into a data structure so that given a query point q in $\mathbb{R}^d$, we can report for each distinct color c of the hyperboxes stabbed by q, the tuple $\langle c, \gamma_c \rangle$ where γ_c = max $\{w(\gamma_c) \mid \gamma_c \in S_c$ and $q \in \gamma_c\}$, i.e., the hyperbox in $S_c \cap q$ with the *topmost/maximum* weight.

We begin by providing a solution to this problem for $d = 1$. Then a solution is provided for $d = 2$.

3.1 Solution for $d = 1$

We start of with n colored segments on the real-line. Consider a color c and the segments S_c of that color. Let $p_1, p_2, \ldots, p_m$ be the list of segment endpoints of S_c sorted from left to right. These endpoints induce partitions on the real-line and these partitions are called "elementary intervals". The elementary intervals, say I_c, from left to right are: $(-\infty, p_1)$, $[p_1, p_1]$, (p_1, p_2), $[p_2, p_2]$, $\ldots$, (p_{m-1}, p_m), $[p_m, p_m]$, (p_m, ∞). With each interval $i \in I_c$, we shall store $w(\gamma)$, where γ is the segment with the maximum weight among all the segments of S_c which intersects interval i. If any interval is not intersected by any of the segments of S_c then it is discarded from I_c. Based on the elementary intervals in I_c, for all colors c, we build an interval tree IT. The number of intervals stored in IT will be bounded by $O(n)$. Given a query point q, we search IT and the intervals stabbed by q are reported. The weight associated with each interval is the required answer.

Theorem 4 *A set of n segments can be stored in a linear-size structure such that given a query point q, a generalized stabbing-max query can be answered in $O(\log n + k)$ time.*

3.2 Solution for $d = 2$

Our approach for solving the problem in two-dimensional plane is to design a dynamic data structure for the one-dimensional version of the problem which can handle query as well as updates efficiently. This is followed by making this data structure partially persistent using the technique of Driscoll et. al. [4].

First, we build the dynamic data structure for the 1D version. Based on the segments in S an augmented segment tree T is built as follows: The segments of S divide the real-line into elementary intervals. A balanced

binary tree T is built with these elementary intervals as the leaves of the tree. Then each segment in S is inserted into T. Consider a node v of T. Let S_c^v be the set of segments of color c assigned to node v. A red-black tree T_c^v is built based on the weights of the segments in set S_c^v (in decreasing order of weights). Also a pointer is maintained from the root of T_c^v to the leftmost leaf in it. In this way red-black trees are built for each unique color of the segments assigned to node v. Given a query point q on the real-line, we search in T. At each node v visited from root to leaf, the weight stored in the leftmost leaf of each red-black tree T_c^v is reported. For each color, the maximum among all the weights reported of that color is found out. The space occupied by T is $O(n \log n)$ and the query time is $O(\log n + k \log n)$.

Now let us consider updates. Insertion of a segment of color c would involve going to $O(\log n)$ nodes in T and inserting itself into the secondary red-black tree T_c^v. A new tree T_c^v is created if it does not exist previously. So, insertion time will be $O(\log^2 n)$ amortized due to possibility of a rotation taking place. Similar analysis holds for deletion of segments.

Now the 1D solution has to be made partially persistent. Following the technique of [11], the x-span of all the rectangles are considered, then broken into elementary intervals and the primary structure of the segment tree is built. We make it persistent by sweeping a horizontal line l from top to bottom, inserting the y-span of a rectangle when it is "entered" by l and delete the same y-span when the sweeping line "leaves" that rectangle. Note that now there won't be any rotations taking place during insertions and deletions of y-spans since the primary structure has already been built on $O(n)$ segments. Hence, the number of changes taking place during an update will be bounded by $O(\log n)$ (constant changes at each of the $O(\log n)$ nodes it is/was assigned to). A 2D-Generalized orthogonal stabbing-max query thus can be answered by first identifying the appropriate version of D and then using it to answer the 1D problem. This leads to the following theorem.

Theorem 5 *A set of n rectangles can be stored in a structure of size $O(n \log n)$ such that given a query point $q \in I\!R^2$, a generalized stabbing-max query can be answered in $O(\log n + k \log n)$ time.*

References

[1] Pankaj K. Agarwal, Lars Arge, Jun Yang, Ke Yi. I/O-Efficient Structures for Orthogonal Range-Max and Stabbing-Max Queries. *11th European Symposium on Algorithms*, 7–18, 2003.

[2] Jean-Daniel Boissonnat , Micha Sharir , Boaz Tagansky , Mariette Yvinec. Voronoi diagrams in higher dimensions under certain polyhedral distance functions. *11th annual symposium on Computational geometry*, 79–88, 1995.

[3] B. M. Chazelle. Filtering search: A new approach to query answering. *SIAM Journal of Computing*, 15, 703–724, 1986.

[4] J.R. Driscoll, N. Sarnak, D.D. Sleator, and R.E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38:86–124, 1989.

[5] P. Gupta, R. Janardan and M. Smid. Efficient non-intersection queries on aggregated geometric data. *11th International Computing and Combinatorics Conference*, 544–553, 2005.

[6] P. Gupta, R. Janardan and M. Smid. Computational geometry: Generalized intersection searching. *Chapter 64, Handbook of Data Structures and Applications, D. Mehta and S. Sahni (editors), Chapman & Hall/CRC, Boca Raton, FL*, 64–1-64–17, 2005.

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

[8] E.M. McCreight. Priority search trees. *SIAM Journal of Computing*, 14(2), 257–276, 1985.

[9] Saladi Rahul, Prosenjit Gupta and Krishnan Rajan. Data Structures for Range Aggregation by Categories *21st Canadian Conference on Computational Geometry (CCCG2009)* , pages 133-136, 2009.

[10] Sanjiv Kapoor. Dynamic Maintenance of Maxima of 2-d Point Sets, *SIAM Journal of Computing*, 29(6): 1858–1877, 2000.

[11] Qingmin Shi, Joseph JáJá. Optimal and near-optimal algorithms for generalized intersection reporting on pointer machines. *Information Processing Letters*, 95(3): 382–388, 2005.

[12] Y. Tao and D. Papadias. Range aggregate processing in spatial databases. *IEEE Transactions on Knowledge and Data Engineering*, 16(12), 2004, 1555–1570.