Approximating Range-Aggregate Queries using Coresets

Yakov Nekrich* Michiel Smid†

Abstract

Let μ be a function that assigns a real number $\mu(P) \geq 0$ to any point set P in $\mathbb{R}^d$; for example, $\mu(P)$ can be the diameter or radius of the smallest enclosing ball of P. Let S be a set of n points in $\mathbb{R}^d$. We consider the problem of storing S in a data structure, such that for any query rectangle Q, we can efficiently compute an approximation to the value $\mu(S \cap Q)$. Our solutions are obtained by combining range-searching techniques with coresets.

1 Introduction

Let S be a set of n points in $\mathbb{R}^d$. In the *orthogonal range searching* problem, we have to construct a data structure such that the following type of queries can be answered: Given a query rectangle $Q = \prod_{i=1}^{d}[a_i, b_i]$, report or count the points of $S \cap Q$, i.e., all points of S that are in the query range Q. This problem has a rich history in computational geometry; see, e.g., the survey by Agarwal and Erickson [2]. In this paper, we consider *range-aggregate* queries, in which we want to compute some function of the point set $S \cap Q$; examples are the diameter, width, and radius of the smallest enclosing ball.

To be more precise, let μ be a function that assigns to any finite set P of points in $\mathbb{R}^d$ a non-negative real number $\mu(P)$. We want to construct a data structure for the set S, such that, for any query range Q, we can efficiently compute the value $\mu(S \cap Q)$.

Gupta *et al.* [6] considered this problem for the planar case (i.e., $d = 2$) and the cases when $\mu(P)$ is the closest-pair distance in P, the diameter of P, and the width of P. They presented a data structure of size $O(n \log^5 n)$, such that the closest-pair in a query rectangle Q can be computed in $O(\log^2 n)$ time. (The preprocessing time, however, is $\Omega(n^2)$. Abam *et al.* [1] gave an algorithm that constructs this data structure in $O(n \log^5 n)$ time.) For the cases when μ is the diameter or width, no data structure is known having size $O(n \cdot polylog(n))$ and $O(polylog(n))$ query time. Therefore, Gupta *et al.* considered approximation algorithms. They presented a data structure of size $O(n \log n)$ that allows a $(1-\varepsilon)$-approximation to the diameter of $S \cap Q$ to be computed in $O(\log^2 n)$ time. They also presented a data structure of size $O(n \log^2 n)$ that allows to compute a $(1+\varepsilon)$-approximation to the width of $S \cap Q$ in $O(\log^3 n)$ time.

In this paper, we consider approximate range-aggregate queries in a more general framework. The main result is that for any function μ that can be approximated using a decomposable coreset (to be defined below), we can construct a data structure of size $O(n \cdot polylog(n))$ that allows to approximate $\mu(S \cap Q)$ in $O(polylog(n))$ time.

The notion of a coreset was introduced by Agarwal *et al.* [3]:

Definition 1 *Let S be a finite set of points in $\mathbb{R}^d$ and let $\varepsilon > 0$ be a real number. A subset S' of S is called an* ε-coreset *of S (with respect to μ) if $\mu(S') \geq (1-\varepsilon) \cdot \mu(S)$*

Let $\mathcal{C}$ be a function that assigns to any finite set S of points in $\mathbb{R}^d$ and any real number $\varepsilon > 0$, an ε-coreset $\mathcal{C}(S, \varepsilon)$ of S.

Let $f(n, \varepsilon)$ be the smallest integer such that for any set S of n points in $\mathbb{R}^d$ and any real number $\varepsilon > 0$, the coreset $\mathcal{C}(S, \varepsilon)$ has size at most $f(n, \varepsilon)$. Since the maximum size of all coresets that we are aware of only depends on ε (and not on n), we will write $f(\varepsilon)$ instead of $f(n, \varepsilon)$.

Definition 2 *The function $\mathcal{C}$ is called a* decomposable coreset function*, if the following holds for any finite set S of points in $\mathbb{R}^d$, any $\varepsilon > 0$, and any partition of S into two sets U and V: Given only the coresets $\mathcal{C}(U, \varepsilon)$ and $\mathcal{C}(V, \varepsilon)$ of U and V, respectively, we can compute the ε-coreset $\mathcal{C}(S, \varepsilon)$ of S in $O(f(\varepsilon))$ time.*

Thus, the algorithm that computes $\mathcal{C}(S, \varepsilon)$ only has "access" to $\mathcal{C}(U, \varepsilon)$ and $\mathcal{C}(V, \varepsilon)$; it does not have access to the entire point sets U and V.

Throughout the rest of this paper, we assume that $\mathcal{C}$ is a decomposable coreset function.

Let S be a set of n points in $\mathbb{R}^d$ and let $\varepsilon > 0$ be a real number. We consider the problem of storing the points of S in a data structure such that, for any query rectangle Q, we can efficiently compute the coreset $\mathcal{C}(S \cap Q, \varepsilon)$.

Using a d-dimensional range tree, in which each node stores an ε-coreset of all points in its subtree, we can compute, given a query rectangle Q, $O(\log^d n)$ canonical nodes whose subtrees partition $S \cap Q$. By Definition 2,

*Department of Computer Science, University of Bonn.

†School of Computer Science, Carleton University, Ottawa, Ontario, Canada K1S 5B6. Research supported by NSERC.

we can then use the coresets stored at these nodes to obtain the coreset $\mathcal{C}(S \cap Q, \varepsilon)$. This data structure has size $O(f(\varepsilon) n \log^{d-1} n)$ and a query time of $O(f(\varepsilon) \log^d n)$. In the rest of this paper, we will present improved solutions.

We start in Section 2 by considering the planar case, i.e., $d = 2$. In Section 3, we extend the solution to any constant dimension $d \geq 2$. In Section 4, we consider the dynamic problem, in which points can be inserted to and deleted from the point set S. Finally, in Section 5, we present some applications for specific functions μ.

2 Coreset Range Queries in $\mathbb{R}^2$

Let S be a set of n points in the plane. We want to store the points of S in a data structure such that the following type of queries can be answered: Given a query rectangle $Q = [a, b] \times [c, d]$, compute the ε-coreset $\mathcal{C}(S \cap Q, \varepsilon)$.

We start by giving a solution for the case when Q is a vertical strip. Then we extend this solution to the case when Q is a three-sided rectangle. Finally, we consider the general case.

2.1 Vertical Strips

Given two real numbers a and b with $a < b$, we want to compute the coreset $\mathcal{C}(S \cap Q, \varepsilon)$, where Q is the set of all points in the plane whose x-coordinates are in the interval $[a, b]$.

We construct a one-dimensional range tree T on the x-coordinates of the points in S. Let S^x denote the set that contains the x-coordinates of all points in S. The leaves of T store the elements of S^x. The *range* of a node v is the interval $rng(v) = [\ell_v, r_v]$, where ℓ_v and r_v denote the smallest and largest values stored in the leaf descendants of v. We denote by S_v the set of all points whose x-coordinates belong to $rng(v)$. At each node v of T, we store the coreset $\mathcal{C}(S_v, \varepsilon)$.

Consider a vertical query slab Q with x-interval $[a, b]$. We can find, in $O(\log n)$ time, $O(\log n)$ canonical nodes $v_1, \ldots, v_m$, such that $S \cap Q = \bigcup_{i=1}^m S_{v_i}$. Thus, using Definition 2, we can use the coresets $\mathcal{C}(S_{v_i}, \varepsilon)$ to compute $\mathcal{C}(S \cap Q, \varepsilon)$ in $O(f(\varepsilon) \log n)$ time.

The space usage of the data structure is $O(f(\varepsilon)n)$. We can reduce this to $O(n)$ by storing x-coordinates of $f(\varepsilon)$ points in every leaf of the tree T. We store coresets $\mathcal{C}(S_v, \varepsilon)$ only at the internal nodes v. Since the number of internal nodes is $O(n/f(\varepsilon))$, all coresets can be stored in $O(n)$ space.

Consider again the vertical query slab Q. Suppose that the successor of a in S^x and the predecessor of b in S^x are stored in the leaves $\ell(a)$ and $\ell(b)$, respectively. Let $S'_{\ell(a)}$ be the set of all points in $\ell(a)$ that are contained in Q, and let $S'_{\ell(b)}$ be the set of all points in $\ell(b)$ that are contained in Q. Then we can compute $O(\log n)$ canonical internal nodes $v_1, \ldots, v_m$, such that

$$S \cap Q = S'_{\ell(a)} \cup S'_{\ell(b)} \cup \left(\bigcup_{i=1}^m S_{v_i} \right).$$

Observe that $S'_{\ell(a)}$ and $S'_{\ell(b)}$ are coresets of themselves and both have size at most $f(\varepsilon)$. Therefore, we can again use Definition 2 to compute the coreset $\mathcal{C}(S \cap Q, \varepsilon)$ in $O(f(\varepsilon) \log n)$ time.

2.2 Three-Sided and General Rectangles

We now consider the case when the query region Q is the set of all points in $\mathbb{R}^2$ whose x-coordinates are in the interval $[a, b]$ and whose y-coordinates are at most c, i.e., $Q = [a, b] \times (-\infty, c]$.

Essentially, our data structure is based on a combination of the sweepline technique and a persistent variant of the data structure of Section 2.1. We can navigate in the tree and obtain the appropriate version of the coreset stored in a node of the range tree using the fractional cascading technique [4]. Details of the construction are given below.

We sort the points of S in increasing order of their y-coordinates. Let the sorted sequence be $p_1, p_2, \ldots, p_n$. A hypothetical horizontal sweepline h is moved in the positive y-direction. Initially, the y-coordinate of h is set to $-\infty$. At any moment, all points of S that are below h are stored in the the tree T of Section 2.1. Thus, T is initially empty and new points are inserted into T as h is moved upwards.

Each node v of T contains the following information: (1) an array v.children that is used to navigate from the node v to its children, (2) sets $Y(v)$ and $Y_1(v)$, where $Y(v)$ contains the y-coordinates of h for all times when a new point is inserted into S_v, and $Y_1(v)$ contains the y-coordinates of h for all times when the set of children of v is updated, and (3) arrays $v.\max$ and $v.\min$ that contain the maximal and minimal values stored in the leaf descendants of v. Every entry in v.children corresponds to an element of $Y_1(v)$ and every entry of $v.\min$ ($v.\max$) corresponds to an element of $Y(v)$.

When the sweepline h is moved above a point p_i, we insert p_i into the corresponding leaf ℓ of T. We update the coresets for all ancestor nodes u of ℓ. That is, we add p_i to the set S_u and construct the coreset $\mathcal{C}(S_u \cup \{p_i\}, \varepsilon)$. We associate each coreset C stored in a node u with the y-coordinate of the point p_i. We also add $p_i.y$ to the set $Y(u)$ and insert a new entry into the arrays $u.\max$ and $u.\min$. Observe that we insert a new entry into $u.\max$ and $u.\min$ even if p_i does not have the largest (smallest) x-coordinate among all point in S_u.

When the number of points in a leaf ℓ equals 4, we replace ℓ with two new leaves ℓ_1 and ℓ_2. We add a new entry to the array v.children for the parent v of ℓ. The new entry v.children[i] contains pointers to ℓ_1 and ℓ_2 instead of a pointer to ℓ; v.children[i] also contains pointers to all other children of v. We associate v.children[i] with the current y-coordinate of the sweepline h. We also add the current y-coordinate of the sweepline h to the set $Y_1(v)$.

Let v be an internal node and let i be its height. When the total number of points that belong to the range of v exceeds 2^{i+1}, we replace v with two new nodes v_1 and v_2. The array w.children for the parent w of v is updated in the same way as for the parent of a leaf node.

Consider a query range $Q = [a, b] \times (-\infty, c]$. Let $\ell(a)$ and $\ell(b)$ denote the leaves that contain the successor of a and the predecessor of b at the time when the sweepline passed c. First, we identify all relevant nodes on the path from the root to $\ell(a)$ and $\ell(b)$. We start at the root; in every visited node v, we identify the predecessor $c(v)$ of c in $Y_1(v)$ using fractional cascading [4]. Then, we use the corresponding entry in the array v.children to find the leftmost child of v that contains an element that is larger than a (resp. the rightmost child that contains an element smaller than b). We can identify the relevant child of v in $O(1)$ time because each node has $O(1)$ children at any time.

When the leaves $\ell(a)$ and $\ell(b)$ and all nodes on the paths from the root to $\ell(a)$ and from the root to $\ell(b)$ are found, we can identify the lowest common ancestor q of $\ell(a)$ and $\ell(b)$. Let π be the set of all nodes that lie on the path from $\ell(a)$ to q or on the path from $\ell(b)$ to q when the sweepline h passes c.

We can find nodes v_i such that $S \cap Q = \bigcup_i S_{v_i}$ and each v_i is the child of some node in π. We can find the predecessors $c(v_i)$ of c in $Y(v_i)$ for all nodes v_i in $O(\log n)$ time using fractional cascading [4]. Consider the coreset $\mathcal{C}(S_{v_i}, \varepsilon)$ associated with the y-coordinate $c(v_i)$. Then we obtain the coreset $\mathcal{C}(S \cap Q, \varepsilon)$ from the coresets $\mathcal{C}(S_{v_i}, \varepsilon)$ for all $v_i \in \pi$. Thus, we can construct the coreset $\mathcal{C}(S \cap Q, \varepsilon)$ in $O(f(\varepsilon) \log n)$ time. The total space usage of the data structure is $O(f(\varepsilon) n \log n)$.

If the x-coordinates of the points are integers, we can reduce the query time to $O(f(\varepsilon) \log n / \log\log n)$ by slightly increasing the space usage. All points are stored in a one-dimensional range tree with node degree $\log^{\delta/2} n$, for any constant $\delta > 0$. The data structure is constructed in the same way as above, but for every node v we maintain coresets for the sets $S_{u_i} \cup S_{u_{i+1}} \cup \ldots \cup S_{u_j}$ for all $1 \le i \le j \le \log^{\delta/2} n$. Additionally, we store a data structure N_u for every node u that enables us to navigate from u to an appropriate child of u in constant time. The data structure N_u contains the values of $u_i.\min$ and $u_i.\max$ for each child u_i of u and supports predecessor queries; a new version of D_u is created every time when a new point is inserted into the range of u. We implement D_u with q-heaps [5], so that predecessor queries are supported in $O(1)$ time.

Every inserted point leads to the construction of $O(\log^{1+\delta} n)$ new coresets. Hence, the space usage of the improved data structure is $O(f(\varepsilon) n \log^{1+\delta} n)$.

We can extend the result for three-sided rectangles to the case of general rectangles using the technique that was previously used for range reporting queries [4, 8]; this technique will be described in the full version.

We thus obtain two data structures that allow to compute, for an arbitrary query rectangle Q, the coreset $\mathcal{C}(S \cap Q, \varepsilon)$. The first structure has size $O(f(\varepsilon) n \log^2 n)$ and query time $O(f(\varepsilon) \log n)$. The second structure has size $O(f(\varepsilon) n \log^{2+\delta} n)$ and query time $O(f(\varepsilon) \log n / \log\log n)$, if all point coordinates are integers.

3 Coreset Range Queries in $\mathbb{R}^d$

Consider a set S of n points in $\mathbb{R}^d$, where $d \ge 3$. We will denote point coordinates by $x, y, z_1, \ldots, z_{d-2}$. A two-dimensional query $Q_2 = [a, b] \times [c, d] \times \mathbb{R}^{d-2}$ can be answered in the same way as in Section 2.2. We can answer three-dimensional queries $Q_3 = [a, b] \times [c, d] \times [e_1, f_1] \times \mathbb{R}^{d-3}$ by constructing a constant-degree range tree T_3 on the coordinate z_1. In every node v of T_3, we store a data structure D_v that answers two-dimensional queries of the form $Q_2 = [a, b] \times [c, d] \times \mathbb{R}^{d-2}$ for all points whose z_1-coordinates belong to the range of v. Given the interval $[e_1, f_1]$, we can compute $O(\log n)$ canonical nodes $v_1, \ldots, v_m$ in T_3 such that $\{p \in S : e_1 \le p_1 \le f_1\}$ is equal to $\bigcup_{i=1}^m S_{v_i}$. Hence, we can compute the coreset for $S \cap Q_3$, by first computing, for all $1 \le i \le m$, the coresets for $S_{v_i} \cap Q_2$ using the data structure D_{v_i}, and then combining them using Definition 2. This can be done in $O(f(\varepsilon) \log^2 n)$ time. The total space used by all data structures of T_3 is $O(f(\varepsilon) n \log^3 n)$.

Alternatively, we can use the range tree T_3 with node degree $\log^{\delta'} n$ for $\delta' = \delta/3$. We can assume w.l.o.g. that all point coordinates are integers by applying a standard reduction to rank space. For any $1 \le i \le j \le \log^{\delta'} n$ and each node u, we store the data structure D_u^{fg} that contains all points whose z-coordinates belong to $rng(u_f) \cup \ldots \cup rng(u_g)$ and answers two-dimensional queries in $O(f(\varepsilon)(\log n / \log\log n))$ time. Given the interval $[e_1, f_1]$, we can compute $O(\log n / \log\log n)$ canonical nodes $v_1, \ldots, v_m$ in T_3 such that $\{p \in S : e_1 \le p_1 \le f_1\}$ is equal to $\bigcup_{i=1}^m \bigcup_{j=f_i}^{g_i} S_{v_{ij}}$, where v_{ij} denotes the j-th child of node v_i. We can find the coreset for each set $S_{v_{ij}} \cap Q_2$ using the data structure $D_{v_i}^{f_i g_i}$, and then combine them using Definition 2. This can be done in $O(f(\varepsilon)(\log n / \log\log n)^2)$ time. As shown in Section 2.2, each D_u^{ij} needs $O(m f(\varepsilon) \log^{2+\delta'} m)$ space, where m is the number of points in D_u^{ij}. Since every point is stored in $O(\log^{1+2\delta'} n)$ data structures, the total space usage increases to $O(n f(\varepsilon) \log^{3+\delta} n)$.

By repeating the construction described above $d - 2$

times, we obtain the following result:

Theorem 1 *Let S be a set of n points in $\mathbb{R}^d$, $d \geq 3$, and let $\varepsilon > 0$ be a real number.*

1. *There exists a data structure of size $O(f(\varepsilon)n \log^d n)$ such that, for any query rectangle Q, the coreset $\mathcal{C}(S \cap Q, \varepsilon)$ can be computed in $O(f(\varepsilon) \log^{d-1} n)$ time.*
2. *For any $\delta > 0$, there exists a data structure of size $O(f(\varepsilon)n \log^{d+\delta} n)$ such that, for any query rectangle Q, the coreset $\mathcal{C}(S \cap Q, \varepsilon)$ can be computed in $O(f(\varepsilon)(\log n/ \log\log n)^{d-1})$ time.*

4 Dynamic Data Structures

We can support one-dimensional queries $Q_1 = [a, b] \times \mathbb{R} \times \ldots \times \mathbb{R}$ by constructing a dynamic range tree T on the first coordinates of the points. Each leaf contains $\Theta(f(\varepsilon) \log n)$ points and every internal node has $O(1)$ children. For every node v of T, we maintain the coreset for all points in the range of v. When a new point p is inserted (deleted), we traverse the path from the leaf that contains p to the root of T and re-build the coreset in each node. We can re-build the coreset for a leaf in $O(f(\varepsilon) \log n)$ time; by Definition 2, we can construct the coreset for an internal node from the coresets of its children in $O(f(\varepsilon))$ time. The tree can be re-balanced using standard techniques. A coreset for an arbitrary interval $[a, b]$ can be constructed as shown in Section 2.1. We can extend this result to d-dimensional queries using the same techniques as described above. Thus we obtain the following theorem:

Theorem 2 *Let S be a set of n points in $\mathbb{R}^d$, $d \geq 2$, and let $\varepsilon > 0$ be a real number.*

1. *There exists a data structure of size $O(n \log^{d-1} n)$ such that, for any query rectangle Q, the coreset $\mathcal{C}(S \cap Q, \varepsilon)$ can be computed in $O(f(\varepsilon) \log^d n)$ time. Updates are supported in $O(f(\varepsilon) \log^d n)$ time.*
2. *For any $\delta > 0$, there exists a data structure of size $O(n \log^{d-1+\delta} n)$ such that, for any query rectangle Q, the coreset $\mathcal{C}(S \cap Q, \varepsilon)$ can be computed in $O(f(\varepsilon)(\log n/ \log\log n)^d)$ time. Updates are supported in $O(f(\varepsilon) \log^{d+\delta} n)$ time.*

5 Applications

Let S be a set of n points in $\mathbb{R}^d$ and let $\varepsilon > 0$ be a real number. To approximate $\mu(S \cap Q)$ for any given query rectangle Q, we first use the results from the previous sections to compute the coreset $S' = \mathcal{C}(S \cap Q, \varepsilon)$. Then we use a brute-force or more sophisticated algorithm to compute $\mu(S')$. By Definition 1, this gives a $(1 - \varepsilon)$-approximation to $\mu(S \cap Q)$. Observe that S' has size at most $f(\varepsilon)$. As a result, the time to compute $\mu(S')$ does not depend on n.

Thus, in order to apply our results, we need a decomposable coreset function $\mathcal{C}$ for the measure μ. Consider a collection D of $O(1/\varepsilon^{d-1})$ directions in $\mathbb{R}^d$ such that any two of them make an angle of $O(\varepsilon)$. Let $\mathcal{C}(S, \varepsilon)$ be the subset of S that contains, for each direction in D, the extreme point of S in this direction. Then $\mathcal{C}(S, \varepsilon)$ is a decomposable coreset function of size $f(\varepsilon) = O(1/\varepsilon^{d-1})$ for measures μ such as the diameter and radius of the smallest enclosing ball. (See Janardan [7] for the case when μ is the diameter.)

We can also define a coreset $\mathcal{C}$ to be *decomposable* if the following condition is satisfied: For any sets S_1 and S_2 with ε-coresets $\mathcal{C}(S_1, \varepsilon)$ and $\mathcal{C}(S_2, \varepsilon)$, $\mathcal{C}(S_1, \varepsilon) \cup \mathcal{C}(S_2, \varepsilon)$ is an ε-coreset for $S_1 \cup S_2$. We can obtain results that are very similar to Theorems 1 and 2. The only major difference is that the coreset for the points inside a query rectangle Q is of poly-logarithmic size.

References

[1] M.A. Abam, P. Carmi, M. Farshi, M. Smid, *On the power of the semi-separated pair decomposition,* WADS, LNCS Volume 5664, 2009, pp. 1–12.

[2] P.K. Agarwal and J. Erickson, *Geometric range searching and its relatives*, In: Advances in Discrete and Computational Geometry, American Mathematical Society, 1999, pp. 1–56.

[3] P.K. Agarwal, S. Har-Peled, K.R. Varadarajan, *Approximating extent measures of points*, Journal of the ACM **51**, 2004, pp. 606–635.

[4] B. Chazelle, L.J. Guibas, *Fractional cascading: I. a data structuring technique*, Algorithmica **1(2)**, 1986, pp. 133–162.

[5] M.L. Fredman, D.E. Willard, *Trans-dichotomous algorithms for minimum spanning trees and shortest paths*, J. Comput. Syst. Sci. **48(3)**, 1994, pp. 533–551.

[6] P. Gupta, R. Janardan, Y. Kumar, M. Smid, *Data structures for range-aggregate extent queries*, CCCG, 2008, pp. 7–10.

[7] R. Janardan, *On maintaining the width and diameter of a planar point-set online*, International Journal of Computational Geometry & Applications **3**, 1993, pp. 331–344.

[8] S. Subramanian, S. Ramaswamy, *The p-range tree: a new data structure for range searching in secondary memory*, SODA 1995, pp. 378–387.