Range-Aggregate Queries for Geometric Extent Problems

Peter Brass[1] **Christian Knauer**[2] **Chan-Su Shin**[3] **Michiel Smid**[4]

Ivo Vigan[5]

[1] Department of Computer Science
The City College of New York, New York, NY, 10031, USA.
Email: peter@cs.ccny.cuny.edu

[2] Institute of Computer Science
Universität Bayreuth, 95440 Bayreuth, Germany.
Email: christian.knauer@uni-bayreuth.de

[3] Department of Digital Information Engineering
Hankuk University of Foreign Studies, Yongin, 449-791, Korea.
Email: cssin@hufs.ac.kr

[4] School of Computer Science
Carleton University, Ottawa, Ontario, K1S 5B6, Canada.
Email: michiel@scs.carleton.ca

[5] Department of Computer Science
City University of New York, The Graduate Center, New York, NY 10016, USA
Email: ivigan@gc.cuny.edu

Abstract

Let S be a set of n points in the plane. We present data structures that solve range-aggregate query problems on three geometric extent measure problems. Using these data structures, we can report, for any axis-parallel query rectangle Q, the area/perimeter of the convex hull, the width, and the radius of the smallest enclosing disk of the points in $S \cap Q$.

Keywords: Computational geometry, range-aggregate query, orthogonal range query, convex hull, width, smallest enclosing disk.

1 Introduction

In the range searching problem, we have to preprocess a given set S of geometric objects (such as points) into a data structure such that, for any query range Q, counting or reporting $S \cap Q$ can be done efficiently. Various forms in diverse applications have been proposed and extensively studied (see for example Agarwal & Erickson (1999)).

A variant of this problem is the *range-aggregate query* problem, which can deal with more complex queries. In its general form, we have a fixed *aggregate function* f, and the set S gets preprocessed into a data structure such that, for any given query range Q, the value $f(S \cap Q)$ can be computed efficiently. Examples of such functions f include algebraic ones such as "sum", "minimum", and "maximum" over the weights pre-assigned to the objects of S, and geometric ones, such as "closest pair", "diameter", and "width" of $S \cap Q$.

These range-aggregate query problems have been recently studied in the fields of computational geometry (Abam et al. 2009, Brodal & Tsakalidis 2011, Das et al. 2012, Davoodi et al. 2012, Gupta 2006, Gupta et al. 2009, 2007, Nekrich & Smid 2010, Rahul et al. 2010, 2011, Sharathkumar & Gupta 2007), database theory (Tao & Papadias 2004), and VLSI layout design (Sharathkumar & Gupta 2006).

In general, geometric aggregate functions are not decomposable, i.e., the answer $f(S \cap Q)$ cannot be derived efficiently from the answers of the subsets which form a partition of $S \cap Q$. Because of this, different techniques have to be developed to answer these kind of queries.

Let S be a set of n points in the plane. For the case when f is the function "closest pair" and Q is an axis-parallel rectangle, (Sharathkumar & Gupta 2007) and (Gupta et al. 2009) present data structures that solve the problem using $O(n \cdot \mathit{polylog}(n))$ space and $O(\mathit{polylog}(n))$ query time. These structures, however, have $\Omega(n^2)$ preprocessing time. (Abam et al. 2009) show that variants of these structures can be constructed in $O(n \cdot \mathit{polylog}(n))$ time, while preserving the bounds on the space and query time. (Abam et al. 2009) also show that closest pair queries in a halfplane can be answered in close to $O(\sqrt{n})$ time using $O(n \cdot \mathit{polylog}(n))$ space.

For the case when f is the function "maximal points" and Q is an axis-parallel rectangle, (Brodal & Tsakalidis 2011) and (Das et al. 2012) present data structures having size $O(n \cdot \mathit{polylog}(n))$ and query time $O(\mathit{polylog}(n) + k)$, where k is the size of the output.

Assume that f is the function "diameter" and Q

The research of the first and fifth author is supported by NSF grant 1017539. The research of the fourth author is supported by NSERC. The research of the third author is supported by NRF grant 2011-0002827.

query type		query time	preprocessing time	preprocessing space	reference
convex hull	area/perimeter	$O(\log^5 n)$	$O(n \log^3 n)$	$O(n \log^2 n)$	Theorem 1
	report	$O(\log^5 n + h)$			
width		$O(k \log^4 n + \log^5 n)$	$O((n^2/k) \log^7 n)$	$O((n^2/k) \log^5 n)$	Theorem 2
smallest enclosing disk		$O(\log^9 n)$	$O(n \log^2 n)$	$O(n \log^2 n)$	Theorem 3

Table 1: Our contributions; k is a parameter between 1 and n. h is the output size of the convex hull.

is an axis-parallel rectangle. Let k be a parameter with $1 \leq k \leq n$. Gupta et al. (2009) present a data structure of size $O((n + (n/k)^2) \log^2 n)$ having query time $O(k \log^5 n)$. If we aim for $O(polylog(n))$ query time, this structure uses close to quadratic space. On the other hand, using $O(n \cdot polylog(n))$ space, the query time will be $\Omega(\sqrt{n})$. In fact, (Davoodi et al. 2012) present evidence that this trade-off cannot be improved.

Let f be a function that can be approximated using coresets; examples are "diameter", "width", and "smallest enclosing disk". For the case when Q is an axis-parallel rectangle, an approximation to $f(S \cap Q)$ can be computed in $O(polylog(n))$ time using $O(n \cdot polylog(n))$ space; see (Gupta et al. 2009, Nekrich & Smid 2010).

1.1 Our contributions

We present data structures for solving *exactly* range-aggregate query problems on sets S of n points in the plane and axis-parallel query regions Q, for three geometric extent measures: the area/perimeter of the convex hull of $S \cap Q$, the width of $S \cap Q$, and the radius of the smallest enclosing disk of $S \cap Q$. Our results are summarized in Table 1. These are the first non-trivial results for solving these queries exactly; previously, only non-trivial results were known for approximation versions of these query problems.

2 Preliminaries

Let S be a set of n points in the plane. We assume that the points in S are in general position. Let $Q := [a_x, b_x] \times [a_y, b_y]$ be a query range, where $a_x \leq b_x$ and $a_y \leq b_y$.

A standard method to identify $S \cap Q$ is by storing the points of S in a *range tree*; see (de Berg et al. 2008). Using this data structure, identifying $S \cap Q$ is done in two phases: (1) Find all points of S lying in the vertical strip of Q defined by the x-interval $[a_x, b_x]$. (2) Select the points in the y-interval $[a_y, b_y]$ among the points in $[a_x, b_x]$.

Let T be a primary range tree, i.e., a balanced binary search tree in which the leaf nodes store the points in S in non-decreasing order of their x-coordinates from left to right. For every node u, we denote by $S(u)$ the *canonical set* of points in S that are stored at the leaf nodes of the subtree rooted at u. It is well-known that a subset of points in the x-interval $[a_x, b_x]$ can be represented as the disjoint union of $O(\log n)$ canonical subsets. If $S(u)$ contributes to $O(\log n)$ of these canonical subsets, then u is called a *canonical node* for the interval. For a given x-interval, we can identify these canonical nodes in $O(\log n)$ time by traversing T from the root.

We associate each node u in T with a secondary range tree $T_y(u)$, built on the y-coordinates of the points in $S(u)$. Then we can identify $O(\log n)$ canonical subsets (or nodes) in $T_y(u)$ whose points lie in $[a_y, b_y]$.

As a result, for a given query range Q, we can compute, in $O(\log^2 n)$ time, a sequence of $O(\log^2 n)$ canonical nodes $v_1, \ldots, v_m$ in the secondary range trees, such that

$$S \cap Q = \bigcup_{i=1}^{m} S(v_i).$$

In addition, we will associate each node in each $T_y(\cdot)$ with additional preprocessed information depending on the individual problem.

The *width* of a point set V is defined to be the minimum distance between any two parallel lines such that V is contained in the strip bounded by these lines. A *smallest enclosing disk* is a minimum-radius disk which encloses all points of V.

Let $\mathrm{ch}(S(u))$, $\mathrm{width}(S(u))$, and $\mathrm{sed}(S(u))$ (in short, $\mathrm{ch}(u)$, $\mathrm{width}(u)$, and $\mathrm{sed}(u)$) denote the convex hull, the width, and the smallest enclosing disk for the points of $S(u)$, for any node u in the range tree. We use $|S|$ and $|T|$ to denote the cardinality of the set S and the number of nodes in the tree T.

3 Convex hull queries

We consider the problem of computing the area and perimeter of $\mathrm{ch}(S \cap Q)$ for a given axis-parallel query rectangle Q.

Additional information. We maintain a two-dimensional range tree mentioned in the previous section. At each node v of each secondary range tree $T_y(\cdot)$, we store three additional pieces of information:

1. $\mathrm{ch}(v)$, the convex hull of $S(v)$,
2. $\mathrm{area}(\mathrm{ch}(v))$, the area of $\mathrm{ch}(v)$, and
3. $T_A(v)$, a balanced binary search tree whose leaf nodes store the points of $\mathrm{ch}(v)$ in counterclockwise order. Let z be an internal node of $T_A(v)$ with at least three points $p_i, p_{i+1}, \ldots, p_j$, $i < j$, at the leaf nodes of its subtree in $T_A(v)$. At each z, we store the area of $\mathrm{ch}(v)$ to the right of $p_i p_j$, i.e., $\mathrm{area}(\mathrm{ch}(\{p_i, \ldots, p_j\}))$. Once we know this area, we can easily get the area of $\mathrm{ch}(v)$ to the left of $p_i p_j$ as $\mathrm{area}(\mathrm{ch}(v)) - \mathrm{area}(\mathrm{ch}(\{p_i, \ldots, p_j\}))$.

Let us check how much space this additional information requires. For any node v in a fixed secondary range tree $T_y(u)$ for some $u \in T$, it takes $O(|S(v)| \log |S(v)|)$ time and $O(|S(v)|)$ space to store both $\mathrm{ch}(v)$ and $\mathrm{area}(\mathrm{ch}(v))$. Computing the area information stored in $T_A(v)$ is done in a bottom-up fashion, thus it takes a time of $O(|S(v)| \log |S(v)|)$ and a

space of $O(|S(v)|)$. For a fixed $T_y(u)$, we need a time of

$$\sum_{v \in T_y(u)} O(|S(v)| \log |S(v)|) = O(|T_y(u)| \log^2 |T_y(u)|),$$

and a space of

$$\sum_{v \in T_y(u)} O(|S(v)|) = O(|T_y(u)| \log |T_y(u)|).$$

Thus, for the range tree T, we need a time of

$$\sum_{u \in T} O(|T_y(u)| \log^2 |T_y(u)|) = O(|T| \log^3 |T|),$$

which is $O(n \log^3 n)$. By a similar analysis, we need $O(n \log^2 n)$ space. As a result, the data structure can be built in $O(n \log^3 n)$ time and uses $O(n \log^2 n)$ space.

Query. To compute $\mathrm{ch}(S \cap Q)$ and its area for the query range Q, we first identify $O(\log^2 n)$ canonical nodes $v_1, \ldots, v_m$ in all secondary range trees such that $S \cap Q = \cup_{1 \le i \le m} S(v_i)$. Using $\mathrm{ch}(v_i)$ stored at each canonical node v_i, we next compute $\mathrm{ch}(S \cap Q) = \mathrm{ch}(\mathrm{ch}(v_1) \cup \ldots \cup \mathrm{ch}(v_m))$ together with its area in $O(\log^5 n)$ time as follows.

To compute $\mathrm{ch}(S \cap Q)$, we compute two outer tangents between convex hulls $\mathrm{ch}(v_i)$ and $\mathrm{ch}(v_j)$, i.e., tangent lines containing two hulls in their same sides, for all pairs (v_i, v_j) with $i \neq j$. Since any two convex hulls are disjoint, we can apply the prune-and-search algorithm by Kirkpatrick & Snoeyink (1995) to compute them in $O(\log(|S(v_i)| + |S(v_j)|)) = O(\log n)$ time. This takes $O(\log^5 n)$ total time. For each $\mathrm{ch}(v_i)$, we now collect the tangents incident to each point $p_k \in \mathrm{ch}(v_i)$ which has at least one tangent. Let e and e' be two edges incident to p_k on $\mathrm{ch}(v_i)$ where e appears before e' in counterclockwise order. Among the tangents from p_k, we choose the one with the smallest angle with respect to the line containing e in counterclockwise direction. Denote the chosen tangent by t_k. We store such t_k for all points p_k having at least one tangent in a list L_i in counterclockwise order. Clearly L_i contains $O(\log^2 n)$ tangents, which can be sorted angularly in $O(\log^2 n \log \log n)$ time. As a result, the ordered lists L_i for all $1 \le i \le m$ can be computed in $O(\log^5 n)$ time.

We now compute $\mathrm{ch}(S \cap Q)$ by traversing the computed tangents. The method is similar to the gift-wrapping convex hull algorithm (O'Rourke 1998). We start with the southernmost point p_k in the southernmost $\mathrm{ch}(v_i)$. Starting from p_k, we want to find the first point in counterclockwise direction along the edges of $\mathrm{ch}(v_i)$ which has some tangent in L_i. This is equivalent to finding two consecutive points in L_i such that p_k lies between the two points along the boundary of $\mathrm{ch}(v_i)$. This can be done by a binary search over the indices of the points in L_i, which takes $O(\log |L_i|) = O(\log \log n)$ time. Traverse the found tangent, say t_l to reach a point p_l in another convex hull $\mathrm{ch}(v_j)$. Again we use the list L_j to find the next point on $\mathrm{ch}(v_j)$ which has a tangent, and traverse it. We continue in this way until we return to $\mathrm{ch}(v_i)$, and thereby complete the construction of $\mathrm{ch}(\mathrm{ch}(v_1) \cup \cdots \cup \mathrm{ch}(v_m))$ in $O(\log^2 \log \log n)$ time.

As a result, we can compute $\mathrm{ch}(S \cap Q)$ in $O(\log^5 n)$ time, in the sense that if we have to report its edges explicitly, then we can do it in $O(\log^5 n + |\mathrm{ch}(S \cap Q)|)$ time.

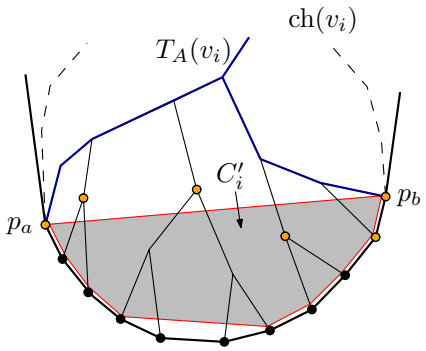

Figure 1: Six canonical nodes for $\{p_a, \ldots, p_b\}$. Only one among them stores the positive area because the others have less than three points in their subtrees.

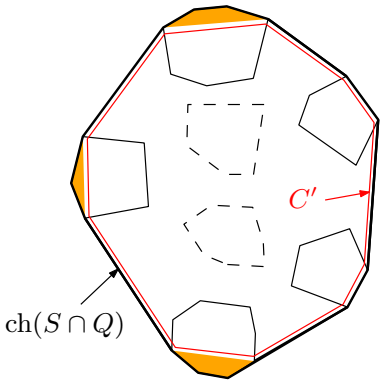

Figure 2: area($\mathrm{ch}(S \cap Q)$) is the sum of area(C') and the area of the shaded regions.

Finally we calculate area($\mathrm{ch}(S \cap Q)$). Consider any $\mathrm{ch}(v_i)$ whose boundary appears on the boundary of $\mathrm{ch}(S \cap Q)$. Then, as shown in Figure 1, we assume that the boundary of $\mathrm{ch}(v_i)$ appears on the boundary $\mathrm{ch}(S \cap Q)$ from a point p_a to a point p_b. Let C_i be the convex polygon with points $p_a, p_{a+1}, \ldots, p_{b-1}, p_b$. We now compute area($C_i$) as follows. We traverse $T_A(v_i)$ to search the leaf nodes storing p_a and p_b, and obtain two paths to p_a and p_b. We collect the positive areas stored at the canonical nodes "below" both paths in $T_A(v_i)$. Since the positive area is defined for three or more consecutive points in $\mathrm{ch}(v_i)$, deleting such points from C_i results in a smaller convex polygon $C'_i \subset C_i$. Then C'_i consists of $O(\log n)$ points because there are $O(\log n)$ canonical nodes in $T_A(v_i)$ and among them only $O(1)$ canonical nodes have one or two points in their subtrees. Now area(C_i) is the sum of the areas stored at the nodes with positive area plus area(C'_i). Since area(C'_i) can be directly computed in $O(|C'_i|) = O(\log n)$ time, we can compute area(C_i) in $O(\log n)$ time. By the same method, we compute area(C_j) for all $\mathrm{ch}(v_j)$ which belong to the boundary of $\mathrm{ch}(S \cap Q)$ in $O(\log^3 n)$ time.

Let $C := \bigcup_i C_i$ and $C' := \mathrm{ch}(S \cap Q) \setminus C$; see Figure 2. Note that area(C) $= \sum_i$ area(C_i), so it can be computed in $O(\log^2 n)$ time. In the worst case, all the canonical nodes $\mathrm{ch}(v_i)$ can contribute to the boundary of $\mathrm{ch}(S \cap Q)$, so C' can have $O(\log^2 n)$ points on its boundary and area(C') can be directly computed in the same time. Since area($\mathrm{ch}(S \cap Q)$) =

area(C) + area(C'), area(ch$(S \cap Q)$) can be computed in $O(\log^3 n)$ time.

The most time consuming step in the computation of area(ch$(S \cap Q)$) is to compute the tangents for all possible pairs of $O(\log^2 n)$ canonical nodes in $S \cap Q$. Thus we can answer area(ch$(S \cap Q)$) in $O(\log^5 n)$ time. The perimeter of ch$(S \cap Q)$ can be obtained in a similar way.

Theorem 1 *Let S be a set of n points in the plane. In $O(n \log^3 n)$ time, we can construct a data structure of size $O(n \log^2 n)$, such that for any axis-parallel query rectangle Q, we can report* ch$(S \cap Q)$ *in $O(\log^5 n + K)$ time and compute the area or the perimeter of* ch$(S \cap Q)$ *in $O(\log^5 n)$ time. Here $K = |\text{ch}(S \cap Q)|$.*

4 Width queries

For the diameter query problem, the diameter diam$(S \cap Q)$ is determined by exactly two points of $S \cap Q$. Each of them belongs to one of the $O(\log^2 n)$ canonical subsets $S(v_1), \ldots, S(v_m)$. If i and j are the indices such that $S(v_i)$ and $S(v_j)$ contain these two points (with $i = j$ being possible), then, diam$(S \cap Q)$ = diam$(S(v_i) \cup S(v_j))$. As a result, diam$(S \cap Q)$ is the maximum value of diam$(S(v_i) \cup S(v_j))$ for all pairs $1 \le i, j \le m$. If we compute in advance a table of diam$(S(v_i) \cup S(v_j))$ for all pairs (v_i, v_j), then we can simply look up the entries in the table which correspond to the canonical subset pairs for $S \cap Q$.

This approach is not applicable to the width problem: The value of width$(S \cap Q)$ is determined by three points of S and we can easily construct an example for which width$(S \cap Q)$ is not in the set of width$(S(v_i) \cup S(v_j) \cup S(v_k))$ over all triples of canonical subsets for $S \cap Q$. Furthermore, the width problem is essentially a non-convex optimization problem, unlike the smallest enclosing circle problem which can use properties associated with convex programming (Eppstein 1992). This is what makes the width problem difficult.

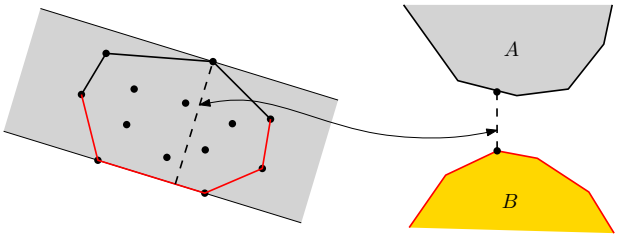

Figure 3: A strip containing S is transformed into a vertical segment connecting ∂A and ∂B.

Additional information. To store an additional information, we use data structures that (Chan 2003) uses to maintain the width of a set of points in a dynamic way.

The width of S, width(S), is determined by three points on ch(S). We consider a dual transformation such that a point (a, b) in the primal plane maps to the line $y = ax + b$ in the dual plane. Then, in the dual plane, the set of all lines above the convex hull of S becomes an unbounded convex polygon A in the positive y-direction, and the set of all lines below the convex hull becomes an unbounded convex polygon B in the negative y-direction; see Figure 3. The strip containing S is mapped to a vertical segment in the dual plane which connects either a vertex of ∂A and a point on ∂B or a point on ∂A and a vertex of ∂B. So width(S) is attained by the minimum vertical distance between ∂A and ∂B. If we denote by $d(A, B)$ the minimum vertical distance between ∂A and ∂B in the dual plane, then width$(S) = d(A, B)$.

Chan (2003) built two data structures $Y(A)$ and $Z(A, B)$ for two convex hulls A and B defined above in the dual plane, which support the following queries:

1. $Y(A)$ can compute $d(A, e)$, for any query line segment e below A, in $O(\log^2 |A|)$ time. $Y(B)$ is defined in a symmetric way.
2. $Z(A, B)$ can compute $d(A, \gamma)$, for any chain $\gamma \subset \partial B$ with two arbitrary endpoints on ∂B, in $O(\log |B|)$ time. $Z(B, A)$ is defined similarly. In fact, $Z(A, B)$ and $Z(B, A)$ are based on $Y(A)$ and $Y(B)$.

The data structure $Y(A)$ can be built in $O(|A| \log^2 |A|)$ time and space, and $Z(A, B)$ in $O(|A| \log^2 |A| + |B| \log^2 |B|)$ time and space (Chan 2003).

For a fixed parameter $1 \le k \le n$, a node v in any secondary structure $T_y(\cdot)$ (in short, T_y) is said to be *big* if $|S(v)| \ge k$, otherwise *small*. In a fixed T_y, there are $O(|T_y|/k)$ big nodes and the number of levels of T_y containing big nodes is $O(\log(|T_y|/k))$.

At each "big" node v in any T_y, we maintain the following three additional pieces of information:

1. $A(v)$ and $B(v)$ as the dual structure for ch$(S(v))$,
2. $Y(A(v))$ and $Y(B(v))$ by Chan (Chan 2003),
3. $Z(A(v), B(w))$ and $Z(B(v), A(w))$ for every big node w in any T_y by Chan (Chan 2003).

Let us check the amount of space needed for these structures. If T_y has a big node, then T_y has at least k nodes, so there are only $O(|T|/k) = O(n/k)$ internal nodes in the primary tree T having such T_y. Since T_y can have at most $|T_y|/k$ big nodes, the number of big nodes in all T_y stored at one level in T is $O(\sum_{T_y} |T_y|/k) = O(n/k)$. In total, there are $O((n/k) \log(n/k))$ big nodes.

For the first additional information, we need $O(|T_y|)$ space for all big nodes at the same level in a fixed T_y. Since T_y has $O(\log(n/k))$ levels, we need $O(|T_y| \log(n/k))$ space for T_y. For T_y stored at the same level of T, the required space is $O(\sum |T_y| \log(n/k)) = O(n \log(n/k))$. Only $O(\log(n/k))$ levels in T store T_y which contains big nodes, so the space amount for the first one is $O(n \log^2(n/k))$.

For the second one, we can similarly check that it needs $O(n \log^2 n \log^2(n/k))$ space.

For the third one, we first check the space needed for $Z(A(v), \cdot)$ and $Z(B(v), \cdot)$ of a fixed v. The space associated with w in some T_y is $O(|T_y| \log^2 n \log(n/k))$. Summing up over all T_y, it becomes $O(n \log^2 n \log^2(n/k))$. Since we have $O((n/k) \log(n/k))$ big nodes v in T, the total space is $O((n^2/k) \log^2 n \log^3(n/k))$.

As a result, we need $O((n^2/k) \log^2 n \log^3(n/k))$ space for the three additional pieces of information. By a similar analysis, we can show that it takes $O((n^2/k) \log^4 n \log^3(n/k))$ time to prepare them.

Query. Let $Q := [a_x, b_x] \times [a_y, b_y]$, where $a_x < b_x$ and $a_y < b_y$. We first identify the $O(\log^2 n)$ canonical nodes for $S \cap Q$. These canonical nodes (or canonical subsets) partition Q, as in Figure 4(a), into disjoint rectangular regions, each of which is associated with a specific canonical node.

We merge the regions for the "small" canonical nodes (equivalently, collect the points stored in the small canonical nodes) such that the merged regions are still rectangular and are disjoint from each other as in Figure 4(b). By the definition of small nodes, it is easy to show that the number of resulting merged regions is $O(\log n)$, and the number of points of $S \cap Q$ lying in each merged region is $O(k)$. We again call a *small node* the union of the small canonical nodes in the region. We denote the small nodes by $u_1, \ldots, u_l$ and the big nodes by $v_1, \ldots, v_m$, where $l = O(\log n)$ and $m = O(\log^2 n)$. Let $\mathcal{C}$ be the union of all small and big canonical nodes for $S \cap Q$.

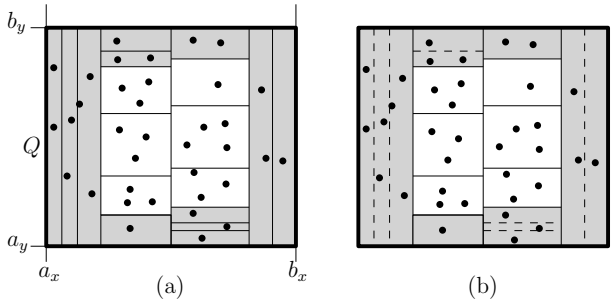

Figure 4: (a) The shaded rectangular regions represent small nodes lying in Q and the white ones represent big nodes. Before merging the small nodes. (b) After merging the small nodes.

The width of $S \cap Q$ is the width of the points stored at the canonical nodes of $\mathcal{C}$. For any node pair $u, v \in \mathcal{C}$, width$(S(u) \cup S(v))$ is the width of the convex hull of $S(u) \cup S(v)$. In the dual plane, this is the minimum vertical distance of $A(u) \cap A(v)$ and $B(u) \cap B(v)$, i.e., width$(S(u) \cup S(v)) = d(A(u) \cap A(v), B(u) \cap B(v))$. Let $\mathcal{A} := \cap_{v \in \mathcal{C}} A(v)$ and $\mathcal{B} := \cap_{v \in \mathcal{C}} B(v)$. Then width$(S \cap Q) = d(\mathcal{A}, \mathcal{B})$.

Since $S(u)$ and $S(v)$, for any two distinct $u, v \in \mathcal{C}$, are separated either horizontally or vertically, $\partial A(u)$ and $\partial A(v)$ intersect at most once, and $\partial B(u)$ and $\partial B(v)$ also intersect at most once. Using this property, we can compute the intersection points by applying the dual version of the method that we already used in Section 3 to compute the convex hull of the convex hulls of canonical subsets in the primal plane. This is done in $O(\log^5 n)$ time. Recall here that small nodes do not have the data structures for convex hulls, but we can construct them from scratch in $O(k \log k)$ time for each small node, so it takes $O(k \log k \log n)$ total time for $O(\log n)$ small nodes. Thus, in $O(k \log^2 n + \log^5 n)$ time, we can know which portion of which $\partial A(v)$ (resp., $\partial B(v)$) contributes to $\partial \mathcal{A}$ (resp., $\partial \mathcal{B}$).

As in Figure 5, draw the vertical lines at these intersection points on each of ∂A and ∂B, and compute the intersections of the vertical lines with the opposite boundary. Since there are $O(\log^2 n)$ vertical lines, we can find such intersections in $O(\log^3 n)$ time by binary searches. As a result, these lines divide the plane into $O(\log^2 n)$ vertical slabs τ, and in a slab τ only one $A(v)$ (resp., only one $B(u)$) contributes to

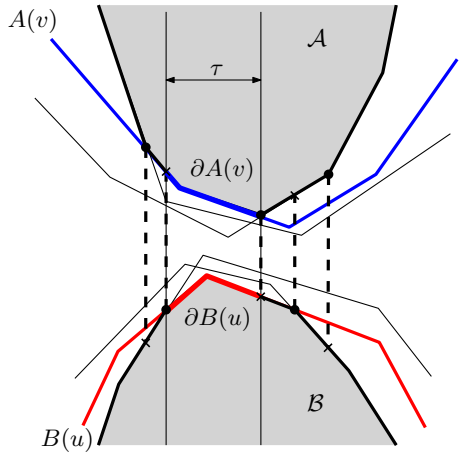

Figure 5: $\mathcal{A}$, $\mathcal{B}$, and vertical slabs.

$\mathcal{A} \cap \tau$ (resp., $\mathcal{B} \cap \tau$). It is clear that width$(S \cap Q)$ is the minimum of $d(A(v), B(u) \cap \tau)$ over all slabs τ.

Fix τ and assume that $\partial A(v)$ and $\partial B(u)$ have the chains whose edges in τ coincide with $\partial \mathcal{A} \cap \tau$ and $\partial \mathcal{B} \cap \tau$, respectively. We have three cases to find the distance $d(A(v), B(u) \cap \tau)$. If both v and u are big nodes, then we ask the distance to $Z(A(v), B(u))$ and $Z(B(u), A(v))$ in $O(\log n)$ time. If one of them is small, say u, then we compute $d(A(v), e)$ for each edge $e \in \partial B(u) \cap \tau$ by asking to $Y(A(v))$, and again compute $d(B(v), e')$ for $e' \in \partial A(u) \cap \tau$ by asking to $Y(B_v)$, which are both done in $O(k \log^2 n)$ time. If they are both small, then we compute the distance directly by two linear scans; one between $\partial A(v) \cap \tau$ and $\partial B(u) \cap \tau$, and the other between $\partial B(v) \cap \tau$ and $\partial A(u) \cap \tau$ in total $O(k)$ time. Then we simply take the minimum of them as $d(A(v), B(u) \cap \tau)$. As a result, we can compute $d(A(v), B(u) \cap \tau)$ for a fixed τ in $O(k \log^2 n)$ time, thus we can compute $d(\mathcal{A}, \mathcal{B})$ in $O(k \log^4 n)$ time in total.

Theorem 2 *Let S be a set of n points in the plane, and let $1 \le k \le n$ be a parameter. In $O((n^2/k) \log^4 n \log^3(n/k))$ time, we can construct a data structure of size $O((n^2/k) \log^2 n \log^3(n/k))$, such that for any axis-parallel query rectangle Q, the width of $S \cap Q$ can be computed in $O(k \log^4 n + \log^5 n)$ time.*

For example, setting $k = n^\varepsilon$, we can answer a query in $O(n^\varepsilon \log^4 n)$ time using $O(n^{2-\varepsilon} \log^5 n)$ space.

5 Smallest enclosing disk queries

A smallest enclosing disk sed(S) for a point set S is determined by two or three points on its boundary. To find it, we first lift the points in S onto a paraboloid in three dimensions, and transform the lifted points, using the duality transform, to the dual space. Then the radius of sed(S) becomes the minimum vertical distance between a convex polyhedron and a paraboloid. We can compute the distance by adapting the three-dimensional linear programming algorithm developed for a semi-dynamic environment by (Eppstein 1992), which guarantees polylogarithmic query time with a data structure of subquadratic size.

5.1 The dual problem

The lifting map and duality transform are well explained in the literature; refer, e.g., to Section 5.7 in the book by O'Rourke (O'Rourke 1998). For completeness, we explain these transformations.

The lifting map. We define the *lifting map*, which maps a point in $\mathbb{R}^2$ to a point on the paraboloid $P : z = x^2 + y^2$ in $\mathbb{R}^3$:

$$p = (x, y) \mapsto p_\uparrow = (x, y, x^2 + y^2).$$

Let C be a circle in $\mathbb{R}^2$ with center (a, b) and radius r. Let H_C be the non-vertical plane defined by

$$H_C : z = 2ax + 2by + r^2 - a^2 - b^2.$$

Let $p = (x, y)$ be a point in $\mathbb{R}^2$. Then p is on, inside, or outside C if and only if $p_\uparrow$ is on, below, or above H_C, respectively.

For a set S of n points, define

$$S_\uparrow = \{p_\uparrow \mid p \in S\}.$$

Let C be a circle with center (a, b) and radius r, and assume that C encloses all points in S. Then all points of $S_\uparrow$ are on or below the plane H_C. Consider a plane H'_C which is parallel to H_C and tangent to the paraboloid P. Then

$$H'_C : z = 2ax + 2by - a^2 - b^2.$$

The vertical distance between H_C and H'_C is equal to r^2. Thus the following observation holds.

Observation 1 *The radius of the smallest enclosing disk of S is the vertical distance between two parallel planes H and H' such that*

1. *all points of $S_\uparrow$ are on or below H,*
2. *H contains either a face or an edge of the upper convex hull of $S_\uparrow$, and*
3. *H' is tangent to the paraboloid P.*

The duality transform. We now define the *duality transform*, which maps any non-vertical plane in $\mathbb{R}^3$ to a point in $\mathbb{R}^3$ as follows:

$$H : z = ax + by + c \mapsto H^* = (a/2, b/2, c).$$

Let S be a set of points in $\mathbb{R}^3$ and define

$$S^* = \{H^* \mid \begin{array}{l} H \text{ is a non-vertical plane} \\ \qquad \text{on or above } \mathrm{ch}(S)\}. \end{array}$$

Then the following observation holds:

Observation 2 *Let S be a set of n points in $\mathbb{R}^3$. The set S^* is convex and unbounded in the positive z-direction.*

Using the paraboloid $P : z = x^2 + y^2$, we define the set

$$P' = \{H^* \mid \begin{array}{l} H \text{ is a non-vertical plane} \\ \qquad \text{on or below } P\}. \end{array}$$

Let P^* denote the boundary of P'. Then we also have a similar observation as we did for S^*.

Observation 3 *The set P^* is convex and unbounded in the negative z-direction. Furthermore P^* is the paraboloid $z = -x^2 - y^2$.*

Dual problem. We are now ready to define the dual problem of the original problem of computing the radius of $\mathrm{sed}(S)$ for a point set S.

We get $S_\uparrow$ by the lifting map, and $S^*_\uparrow$ by the dual transform. Then $S_\uparrow$ is the set of points in the primal space, and $S^*_\uparrow$ is the set of points in the dual space. Also we apply the duality transform to map the paraboloid P with equation $z = x^2 + y^2$ to the paraboloid P^* with equation $z = -x^2 - y^2$.

Let us define a set B^* of points in the dual space as follows:

$$B^* = \{H^* \mid \begin{array}{l} H \text{ is a non-vertical plane containing} \\ \qquad \text{some face of the upper hull of } S_\uparrow\}. \end{array}$$

Let $\mathcal{B}^*$ be the set of all the points, in the dual space, "on" or "vertically above" the lower convex hull of B^*. Then $\mathcal{B}^*$ is a convex polyhedron unbounded in the positive z-direction, which is fully contained in $S^*_\uparrow$. We also easily prove that $\mathcal{B}^*$ and P^* are disjoint.

Consider two parallel planes H and H' such that all points of $S_\uparrow$ are on or below H, and H' is tangent to the paraboloid P. If the distance between H and H' gives the radius of $\mathrm{sed}(S)$, then in the dual space, the point H^* is on the boundary of $S^*_\uparrow$ and the point $(H')^*$ is on the paraboloid P^*. Furthermore, since $\mathrm{sed}(S)$ contains either two or three points on its boundary, H must contain either an edge or a face of the upper convex hull of $S_\uparrow$, which implies that H^* is either an edge or a vertex of the lower convex hull of B^*, i.e., an edge or a vertex of $\mathcal{B}^*$. Thus we have the following fact.

Fact 1 *Let S be a set of n points in the plane. The radius of the smallest enclosing disk $\mathrm{sed}(S)$ of S is equal to the minimum vertical distance between $\mathcal{B}^*$ and P^*.*

Let us go back to our query problem. Let $v_1, \ldots, v_m$ be the canonical nodes in T_y for $S \cap Q$. Then $S \cap Q = \bigcup_{i=1}^m S(v_i)$. We want to compute $\mathrm{sed}(S \cap Q)$. For each v_i, we define $S_\uparrow(v_i)$, $S^*_\uparrow(v_i)$, and $\mathcal{B}^*(v_i)$ for the canonical subset $S(v_i)$ as above. To guarantee that a disk contains all points in $S \cap Q$, its associated plane H in the lifting space must be on or above all $S_\uparrow(v_i)$, i.e., on or above $\mathrm{ch}\,(\bigcup_i S_\uparrow(v_i))$. But this means that H^* is a point on the boundary of $\bigcap_{i=1}^m S^*_\uparrow(v_i)$ in the dual space. More precisely, H^* is a point on $\bigcap_{i=1}^m \mathcal{B}^*(v_i)$.

As a result, computing the smallest enclosing disk of $S \cap Q$ is equivalent to computing the minimum vertical distance between the paraboloid P^* and the intersection of the convex polyhedra $\mathcal{B}^*(v_1), \ldots, \mathcal{B}^*(v_m)$. Actually, Eppstein (1992) already explained this dual transformation to maintain the smallest enclosing disk of points in two dimensions in a semi-dynamic setting; see Corollary 1 in (Eppstein 1992).

5.2 Data structure and algorithm

Let A be a convex polyhedron of n vertices. We represent A by a *hierarchical representation* of A, as given by Dobkin & Kirkpatrick (1990). See also Section 7.10 in the book by O'Rourke (1998) for a detailed explanation.

A sequence $\mathrm{hier}(A) = A_1, \ldots, A_k$ of convex polyhedra is said to be a hierarchial representation of A if

1. $A_1 = A$ and A_k is a tetrahedron,

2. $A_{i+1} \subset A_i$ for $1 \leq i < k$,
3. $V(A_{i+1}) \subset V(A_i)$ for $1 \leq i < k$, where $V(A_j)$ denotes the set of vertices of A_j, and
4. the vertices of $V(A_i) \setminus V(A_{i+1})$ form an independent set in A_i for $1 \leq i < k$.

Dobkin & Kirkpatrick (1990) presented an algorithm to construct hier(A) in $O(n)$ time such that (1) $k = O(\log n)$, (2) $\sum_{i=1}^{k} |V(A_i)| = O(n)$, and (3) the maximum degree of the vertices of $V(A_i) \setminus V(A_{i+1})$ in A_i is a constant, say at most 8. They also showed the following crucial lemmas.

Lemma 1 (Dobkin & Kirkpatrick 1990) *Given a hierarchical representation* hier$(A) = A_1, \ldots, A_k$*, and any query plane H such that $A_{i+1} \subset H^+$ for some i, then either $A_i \subset H^+$ or there exists a unique vertex $v \in V(A_i)$ such that $v \in H^-$, where H^+ and H^- denote the half-space above and below H, respectively. Furthermore, such v can be found in constant time.*

Lemma 2 (Dobkin & Kirkpatrick 1990) *Given a hierarchical representation* hier(A) *and any query plane H that does not intersect A, the minimum vertical distance between A and H can be computed in $O(\log |A|)$ time.*

Let $d(A, B)$ be the minimum vertical distance between disjoint convex sets A and B. The algorithm given in Lemma 2 computes $d(A, B)$ when B is a plane. We can simply adapt the algorithm in Lemma 2 to compute $d(A, B)$ for the case when B is a paraboloid:

Lemma 3 *Let A be a convex polyhedron which is unbounded in the positive z-direction, let P^* be the paraboloid $z = -x^2 - y^2$, and assume that A is above P^*. If* hier$(A) = A_1 (= A), \ldots, A_k$ *is given, we can compute $d(A, P^*)$ in $O(\log |A|)$ time.*

Proof. We first compute $d(A_k, P^*)$ in constant time, which is possible because A_k is a tetrahedron. We now update a pair (a_i, p_i) of points $a_i \in \partial A_i$ and $p_i \in P^*$, realizing $d(A_i, P^*)$, as i is decremented from k to 1. Since $d(A_i, P^*)$ is equal to $|a_i p_i|$, when we translate P^* in the positive z-direction by $d(A_i, P^*)$, it first hits A_i at a_i. Let H_{P^*} be the plane tangent to P^* at the point p_i, and let H_A be the plane through a_i parallel to H_{P^*}. Then it follows that A_i is above H_A and P^* is below H_{P^*}, i.e., their interiors are separated by both of H_{P^*} and H_A. We now compute $d(A_{i-1}, P^*)$ by identifying (a_{i-1}, p_{i-1}). Since $A_{i-1} = (A_{i-1} \cap H_A^+) \cup (A_{i-1} \cap H_A^-)$,

$$d(A_{i-1}, P^*) = \min\{d(A_{i-1} \cap H_A^+, P^*), d(A_{i-1} \cap H_A^-, P^*)\}.$$

Clearly, $d(A_{i-1} \cap H_A^+, P^*)$ is attained by (a_i, p_i). Thus it suffices to compute $d(A_{i-1} \cap H_A^-, P^*)$. By Lemma 1, there can be only one vertex $v \in A_{i-1} \cap H_A^-$ and it can be identified in constant time. Thus if such v exists, then $A_{i-1} \cap H_A^-$ is of constant complexity because v has constant degree in A_{i-1} by definition of hier(A). This allows us to compute $d(A_{i-1} \cap H_A^-, P^*)$ in $O(1)$ time. Therefore, we can update (a_{i-1}, p_{i-1}) from (a_i, p_i) in $O(1)$ time. Since $k = O(\log |A|)$, the total time is $O(\log |A|)$. ◻

Additional information. At each node v in any secondary range tree T_y, we maintain only one additional structure as follows:

1. hier$(\mathcal{B}^*(v))$, a hierarchical representation for the convex polyhedron $\mathcal{B}^*(v)$.

Since hier(A) can be constructed in $O(|A|)$ time (Dobkin & Kirkpatrick 1990), this information can be computed in $O(n \log^2 n)$ time and space.

Query. Let $v_1, \ldots, v_m$ be the canonical nodes for $S \cap Q$. Recall that $m = O(\log^2 n)$. As mentioned earlier, we need to compute

$$d(\bigcap_{i=1}^{m} \mathcal{B}^*(v_i), P^*).$$

Let us consider the elementary case that $m = 1$, i.e., computing $d(\mathcal{B}^*(v_1), P^*)$. This can be done in $O(\log n)$ time by Lemma 3. Once we can solve this elementary case in $O(\log n)$ time, we can employ the algorithm by Eppstein (1992) as follows:

Lemma 4 (Eppstein 1992) *Given m convex polyhedra represented by their hierarchical representations, we can optimize any given objective function over their common intersection in $O(\gamma \cdot m^3 \log^2 n)$ time, provided that the elementary problem of optimizing the function over one convex polyhedron can be done in $O(\gamma)$ time.*

In our case, since $\gamma = O(\log n)$ and $m = O(\log^2 n)$, we can compute $d(\bigcap_{i=1}^{m} \mathcal{B}^*(v_i), P^*)$ in $O(\log^9 n)$ time. Thus, the radius of $\mathrm{sed}(S \cap Q)$ can be computed within the same time bound.

Theorem 3 *Let S be a set of n points in the plane. In $O(n \log^2 n)$ time, we can construct a data structure of size $O(n \log^2 n)$, such that for any axis-parallel query rectangle Q, the radius of the smallest enclosing disk of $S \cap Q$ can be computed in $O(\log^9 n)$ time.*

6 Concluding remarks

An immediate open question is to construct more efficient data structures for the three extent measures. It might be quite possible to reduce a few logarithmic factors from the current time bounds.

Another interesting question is whether width queries can be answered in $O(\mathit{polylog}(n))$ time using a data structure of size $O(n \cdot \mathit{polylog}(n))$.

It would be interesting to consider other extent measures such as the largest empty disk within the convex hull of $S \cap Q$, and the minimum annulus containing $S \cap Q$. Finally, it would be interesting to consider different query ranges, such as circles or halfplanes, or to extend to higher dimensions.

References

Abam, M. A., Carmi, P., Farshi, M., & Smid, M., (2009), On the power of the semi-separated pair decomposition, in *Proc. of WADS*, LNCS Springer, 2009, pp. 1–12.

Agarwal, P.K. & Erickson, J. (1999), Geometric range searching and its relatives. *in* B. Chazelle, J. E. Goodman, and R. Pollcak, editors, Advances in Discrete and Computational Geometry, Contemporary Mathematics, 23, pp. 1–56, AMS.

de Berg, M., Cheong, O.,van Kreveld, M., & Overmars, M. (2008), *Computational Geometry: Algorithms and Applications*. Springer-Verlag.

Brodal, G. S. & Tsakalidis, K. (2011), Dynamic planar range maxima queries, in *ICALP*, LNCS Springer, pp. 256–267.

Chan, T.M. (2003), A fully dynamic algorithm for planar width, *Discrete and Computational Geometry*, 30(1), pp. 17–24.

Das, A.S., Gupta, P., & Srinathan, K. (2012), Counting maximal points in a query orthogonal rectangle, in *Proc. of CCCG*, pp. 37–42.

Davoodi, P., Smid, M., & van Walderveen, F., (2012), Two-dimensional range diameter queries, in *LATIN*, LNCS Springer, 2012, pp. 219–230.

Dobkin, D. P. & Kirkpatrick, D. G. (1990), Determining the separation of preprocessed polyhedra - a unified approach, in *ICALP*, LNCS Springer, 443, pp. 400–413.

Eppstein, D. (1992), Dynamic three-dimensional linear programming, *INFORMS J. on Computing*, 4(4), pp. 360–368.

Gupta, P. (2006), Algorithms for Range-Aggregate Query Problems Involving Geometric Aggregation Operations, *Nordic Journal of Computing*, 13(4), pp. 294–308.

Gupta, P., Janardan, R., Kumar, Y. & Smid, M. (2009), Data Structures for Range-Aggregate Extent Queries. in *Proc. of CCCG*, pp. 7–10.

Gupta, P., Janardan, R. & Smid, M. (2007), Efficient non-intersection queries on aggregated geometric data. *Int. J. of Computational Geometry and Applications*, 19(6), pp. 479–506.

Kirkpatrick, D. G. & Snoeyink, J., (1995), Computing common tangents without separating lines, in *Proc. of WADS*, LNCS Springer, 1995, pp. 183–193.

Nekrich, Y. & Smid, M. (2010), Approximating range-aggregate queries using coresets. in *Proc. of CCCG*, pp. 253–256.

Overmars, M. & van Leeuwen, J. (1981), Maintenance of configurations in the plane, *J. Comput. Syst. Sci.,*, 23, pp. 116–204.

O'Rourke, J. (1998), *Computational Geometry in C*, Cambridge University Press, Second Edition.

Rahul, S., Bellam, H., Gupta, P. & Rajan, K. (2010), Range aggregate structures for colored geometric objects. in *Proc. of CCCG*, pp. 249–252.

Rahul, S., Das, A.S., Rajan, K.S. & Srinathan, K. (2011), Range-Aggregate Queries Involving Geometric Aggregation Operations. in *Proc. of WALCOM: Algorithms and Computation*, 6552, pp. 122–133.

Sharathkumar, R. & Gupta, P.(2006), Range-aggregate proximity detection for dsign rule checking in VLSI layouts. in *Proc. of CCCG*, pp. 151–154.

Sharathkumar, R. & Gupta, P. (2007), Range-aggregate proximity queries. Technical Report IIIT/TR/2007/80, I.I.I.T. Hyderabad.

Tao, Y. & Papadias, D. (2004), Range aggregate processing in spatial databases. *IEEE Trans. on Knowledge and Data Engineering*, 16, pp. 1555–1570.