Range Query Problems in Planar Computational Geometry: Radius of Smallest Enclosing Disk & Generalized Convex Hull

Thesis submitted in partial fulfilment
of the requirements for the degree of

Master of Science
in Computer Science and Engineering
by Research

by

Sankalp Khare
200702039
sankalp.khare@research.iiit.ac.in

International Institute of Information Technology, Hyderabad
Deemed to be University
Hyderabad - 500 032, INDIA
July 2024

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Range Query Problems in Planar Computational Geometry: Radius of Smallest Enclosing Disk & Generalized Convex Hull" by Sankalp Khare, has been carried out under my supervision and is not submitted elsewhere for a degree.

_____ _____

Date Advisor: Dr. Kannan Srinathan

To my parents, who never gave up on me.

Acknowledgments

I'm very grateful to my advisor Dr. Kannan Srinathan, who has been a beacon of inspiration and guidance over the years. His kindness, patience, grace and empathy will forever stay with me and push me to be a better person. I am lucky to have been associated with him.

Forever indebted to my mentors and co-authors Jatin Agarwal and Nadeem Moidu for welcoming me into the Computational Geometry research group at CSTAR and for helping me achieve things I never imagined possible. Without their help none of this would have happened. I am also very grateful to Dr. Kishore Kothapalli for his guidance and his unwavering commitment towards the success of all researchers in the lab and at IIIT-H.

I'm fortunate to have had people around me who see more in me than I do myself. They lifted me up when I had no hope and nudged me in the right direction. I am indebted to them. Special thanks to those who pushed me to complete the dual degree, those who helped in some or the other way, and even those who exercised restraint and did not bring it up ever so frequently — Varun, Swarnabha, Charvi, Achira, Shivani, Parth, Medha, Annapurna, Himanshu, Kulbir, Yogesh, Ashray, Rohit, Rahul, Abhishek, Nikhil, Mansie and many more... you know who you are.

So much love to my parents who kept faith in me through thick and thin. They did not agree with many of my decisions yet they let me do this at my own pace, and their unwavering support played a huge part. My sister for her pep talks, and cousins and extended family for being sympathetic towards my situation.

I'm also grateful to my workplace (Indeed, where I've been employed for almost 10 years now) for being sympathetic towards this goal, and allowing me time to work on the completion of the dual degree whenever I asked for it.

There are many more people who played a part in this journey and I am thankful to all of them.

Special mention to Dr. P. Kumaraguru for his efforts towards completion of the course for long-pending dual degree students like me.

IIIT Hyderabad has given me so much. The people, the campus and the times spent here will forever be a part of me, and I hope in some small way I will also remain a part of this great institution.

Abstract

Computational geometry is a branch of computer science which deals with computation of geometric functions. It has applications in diverse fields such as computer graphics, robotics, VLSI/chip-design, etc. It has also found real-world applications in software such as maps applications that are used daily by millions of people around the world.

Range query problems are a set of problems where a data structure is designed that can pre-process the input set of points so that given a query range, the results of some geometric function over that query range can be computed efficiently. A lot of these problems are non-decomposable, in that the result cannot be computed using the result of the same function on subsets of the query range. These problems are significant today because we have numerous cases of a fixed larger dataset on which different queries are made that need to be answered in the most efficient possible way. Again map software is a great example — updates to the overall map are relatively infrequent, but queries on its data, such as finding distance between two points, are supported at scale.

In this thesis, we present results for two different Range Aggregate Query problems in Two Dimensions: Finding the radius of the smallest enclosing disk (Range Aggregate Queries) & Reporting the distinct colours of points on the Convex Hull (Generalized Intersection Searching).

Smallest Enclosing Disk Range Query: Let S be a set of n points in the plane. We present a method where, using $O(n \log^2 n)$ time and space, S can be pre-processed into a data structure such that given an axis-parallel query rectangle q, we can report the radius of the smallest enclosing disk of the points lying in $S \cap q$ in $O(\log^6 n)$ time per query.

Generalized Convex Hull: We present an output sensitive algorithm for the generalized reporting version of the planar range convex hull problem. Our solutions is in the pointer machine model, for orthogonal range queries on a static point set. We provide a solution for the planar range convex hull problem that answers queries in $O(\log^2 n + c \log n)$ time, using $O(n \log^2 n)$ space, where n denotes the number of stored points and c the number of colors reported.

Publications

List of Publications:

- Sankalp Khare, Jatin Agarwal, Nadeem Moidu and Kannan Srinathan. Improved bounds for Smallest Enclosing Disk Range Queries. *26^{th} Canadian Conference on Computational Geometry 2014*.
- Nadeem Moidu, Jatin Agarwal, Sankalp Khare, Kishore Kothapalli and Kannan Srinathan. On Generalized Planar Skyline and Convex Hull Range Queries. *WALCOM 2014*.

Contents

List of Figures

Chapter *1*

Introduction

1.1 Computational Geometry

Geometry is the study of shape, size, relative position of figures and the properties of space. Early studies of geometry was mainly concerned with lengths, areas and volumes of objects. The field has now expanded to topics like differential geometry, topology, algebraic geometry, etc.

Computational geometry is the branch of computer science devoted to the study of algorithms which involves geometry. Various geometric problems arise in different fields of computer science. Computer graphics represents images using points and lines and do operations on it like image rendering. Robotics require the best path for the robot to move given the current positions of near by objects. Integrated circuit design largely relies on geometric problems to minimize the area required for the design. Geographic Information Systems (GIS) involves a lot of location based searching and route planning. Apart from the direct application such as the ones mentioned above, we also have problems in other fields which can be converted to problems in geometry. For example, a database table with two numerical parameters can be treated as a set of points.

1.2 Range Queries

Range query data structures are those which support operations on a subset of the point set. e.g., You have a one dimensional array, A, preprocess it such that queries of the following form can be quickly reported, "Given a query $[low, high]$, report the sum of all values, A_i, where $low \leq i \leq high$". Range queries often come up in the various geometry related fields mentioned earlier, such as databases, robotics, etc.

1.3 Online and Offline Algorithms

An online algorithm is one where the input is processed one by one without waiting to receive the entire data set. An offline algorithm is one where the data processing starts only after receiving the complete data. In the case of sorting, insertion sort is considered an online algorithm since it maintains the available values in sorted order before starting to the process the next value. On the other hand, selection sort is an offline algorithm since it cannot start before receiving all the values. Having the complete input beforehand is considered an extra resource, since it can simplify certain problems, e.g., compressing a video is easier and can be done more efficiently if the complete video is available as opposed to compressing a video stream on the fly.

In range query problems, an algorithm is usually called offline if it takes all the queries before giving the output even for the first one. This method is not preferred because it does not suit most practical scenarios. However, it is still used if an online algorithm for the same is found to be difficult.

All the algorithms discussed in this thesis are online algorithms.

1.4 Output Sensitivity

The output of a range query is often a subset of the point set. If k is the size of the output, then reporting the output alone takes $O(k)$ time. It is possible that $k = \Theta(n)$, where n is the size of the complete point set. In this case, the reporting takes $O(n)$, which could be the time taken for a trivial algorithm. However, if $k = o(n)$, then we could have some algorithm which takes only time which is polynomial in k and $logn$. It is still relevant to study such algorithms. So it is important to have a parameter in the time complexity which is the size of the output. This concept of having the algorithm depend on the output size is called output sensitivity. Range query algorithms usually have complexity of the form $O(f(n) + kg(n))$, where $f(n)$ and $g(n)$ are normally $o(n)$, typically $O(polylog(n))$.

If the range query is a function of a subset of the point set with only one numerical value as the output, then it should not have a dependence on k which is the size of the subset on which the function is computed. For instance, if the sum of values in a range in an array is to be computed, then any algorithm which goes through all those values in that range will not be considered output sensitive. The size of the output in this case is $O(1)$, so the time complexity of the algorithm should only depend on n.

1.5 Preprocessing for Geometric Problems

Any query based problem requires some preprocessing. However, problems in geometry have an additional advantage that some common preprocessing structures can solve a very large set of problems. These include Voronoi diagrams, Delaunay triangulation, convex hulls, etc.

Preprocessing a set of points and efficiently answering geometric queries on a subset of the points is a common approach used in geometric range searching problems.

1.6 Convex Hull

The convex hull of a set of points is the convex figure with the minimum area which includes all the points in the given set. Intuitively it can be understood as the shape a rubber-band will take if wrapped around the set of points.

Computing the convex hull of a finite set of points on a plane is one of the oldest problems studied in computational geometry.

1.7 Generalized Range Queries

Here we consider a set of points where each point has a color associated with it. In a generalized range query problem we have to report the colorwise information about the problem. For example, the range counting problem counts the number of points in the range. The generalized range counting problem (also called colored or categorical range reporting) counts the number of distinct colors in the range. The problem has similarities to the GROUP BY operator in databases. Consider the following problem: "Find the number of items which have products with price less than x". This can be solved using a generalized range counting data structure.

Chapter *2*

Prior Work

2.1 Range Aggregate Queries

Range aggregate query problems have been the focus of much work in the past decades [29] [19] [27] [28] [8] [6] [25] [24]. A number of results came from the Computational Geometry Research Group at CSTAR, IIIT-H.

2.2 Generalized Intersection Searching

Generalized intersection searching was introduced by Janardan and Lopez [20]. Since then there has been a considerable amount of work on generalized searching and reporting problems [14] [15] [16] [5] [17] [18] [3] [12] [30] [27]. A comprehensive survey of developments in the area can be found in [13]. Some results here were also from IIIT-H (CSTAR, LSI).

Chapter *3*

Improved bounds for Planar Smallest Enclosing Disk Range Queries

3.1 Introduction

The *range-aggregate query* problem is a variant of range searching wherein the goal is to preprocess a set of points, S, into a data structure such that given a query region q, the value of an *aggregate function* f, over the region $S \cap q$, can be computed efficiently. When compared to standard range queries, a range-aggregate query provides a more informative summary of the output.

Aggregate functions can be numeric, for example *count*, *sum* (of weights), etc. or geometric, such as *width*, *closest/farthest pair*, *maximal/dominating chain* etc. Geometric aggregate functions are, in general, non-decomposable, i.e., the desired result $f(S \cap q)$ cannot be derived efficiently from the results obtained by applying f on partitions of $S \cap q$. This calls for the development of more sophisticated techniques to answer geometric range aggregate queries.

All previously known non-trivial results for the *smallest enclosing disk* range-aggregate query problem were approximation versions [25] [24] . Brass et al. [6] presented the first *exact*, non-trivial solution. Their solution requires $O(n \log^2 n)$ preprocessing time and $O(n \log^2 n)$ space to build a data structure which answers queries in $O(\log^9 n)$ time. We improve upon their result, presenting a method that utilizes the same amount of time and space for preprocessing, but answers queries in $O(\log^6 n)$ time.

3.2 Preliminaries

We assume that the points in S are in general position. Let q be an orthogonal query range of the form $[a_x, b_x] \times [a_y, b_y]$, where $a_x < b_x$ and $a_y < b_y$.

Proposition 1. Given a set of points, S, in the plane, the smallest enclosing disk, $\mathrm{sed}(S)$, is the minimal radius disk such that every point $p \in S$ lies on or within the boundary of $\mathrm{sed}(S)$.

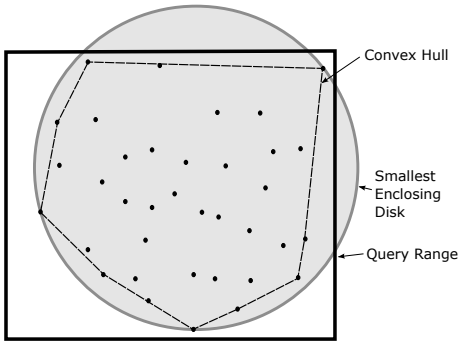

Figure 3.1: Smallest Enclosing Disk and Convex Hull in a query range

3.3 Previous Work

In Section 5.1 of [6], the authors demonstrate how, for a set of points in the plane, the problem of computing the radius of the smallest enclosing disk can be transformed to the problem of finding the minimum vertical distance between a convex polyhedron and a paraboloid in the dual space. The methods used to achieve this transformation are well studied in the literature (Section 5.7 of [26]), but for completeness we will outline the procedure here.

The remainder of section 4.2 covers relevant parts of the solution by Brass et al [6].

3.3.1 The lifting map:

A lifting map *lifts* points in two dimensions into the three dimensional space by assigning them a z co-ordinate which is a function of their x and y co-ordinates. We define our lifting map so that a point in $\mathbb{R}^2$ is mapped to a point on the paraboloid $P : z = x^2 + y^2$ in $\mathbb{R}^3$:

$$z = f(x, y) = x^2 + y^2$$

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

3.3.2 The duality transform

A duality transform defines a mapping between any two sets of parameterized geometric objects. It provides an alternative way to view the same information. We employ the following duality transform, which maps a non-vertical plane H to a point H^*, in $\mathbb{R}^3$:

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

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

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

A point $p : (x, y)$ in $\mathbb{R}^2$ lies on, inside or outside the circle C if and only if the point $p_\uparrow : (x, y, x^2 + y^2)$ lies on, below or above H_C, respectively.

3.3.3 The Dual Problem

Using the lifting map, our point set S is mapped to the set $S_\uparrow$ defined as:

$$S_\uparrow = \{p_\uparrow \,|\, p \in S\}$$

Let C be a circle with center (a, b) and radius r, such that it encloses all points in the set S. Then all points in $S_\uparrow$ lie on or below the plane H_C. Define the plane H'_C:

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

Clearly H'_C is (a) parallel to H_C, and (b) tangent to the paraboloid P. Also, the vertical distance between the two planes, H_C and H'_C is r^2.

If C were the smallest enclosing disk for the set S, then either two or three points in S would lie on the boundary of C and their lifted counterparts in $S_\uparrow$ would lie on H_C, i.e., either an edge (in the two-point scenario) or a face (in the three-point scenario) of the *upper* convex hull of $S_\uparrow$ would lie on H_C (since all points in $S_\uparrow$ lie below H_C, in the negative z-direction). This brings us to the following observation:

Observation 1. The square of the radius of $\mathrm{sed}(S)$ 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 (positive z-direction) of $S_\uparrow$, and
3. H' is tangent to the paraboloid P.

$$G^* = \{H^* \mid H \text{ is a non-vertical plane on or above the convex hull of } G\}$$

The following observation then holds:

Observation 2. For a set of points G in $\mathbb{R}^3$, the set G^* is convex and unbounded in the positive z-direction.

For a set P' defined as follows,

$$P' = \{H^* \mid H \text{ is a non-vertical plane on or below the paraboloid } P\}$$

Let P^* be the boundary of P'. Then we can make the following observation, similar to Observation 2:

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

We now have the following geometric constructs:

- $S_\uparrow$, in the primal space, is the set of points obtained by lifting the input point set S.
- $S^*_\uparrow$, in the dual space, is the set of points obtained by applying the duality transform on $S_\uparrow$.
- P^*, in the dual space, is the paraboloid obtained by applying the dual transform on the paraboloid P.

We define a set of points B^*, in the dual space, as follows:

$$B^* = \{H^* \mid H \text{ is a non-vertical plane containing some face of the upper convex hull of } S_\uparrow\}$$

Furthermore, we define $\mathcal{B}^*$ as follows:

$$\mathcal{B}^* = \{p^* \mid p^* \text{ is a point in the dual space which lies in the region } \textit{vertically above} \text{ the lower convex hull of } B^*\}$$

In other words, all points in $\mathcal{B}^*$ project down (negative z-direction) to some point on the lower convex hull of B^*. Clearly, $\mathcal{B}^*$ is a convex polyhedron unbounded in the positive z-direction, such that (a) $\mathcal{B}^*$ is fully contained within $S^*_\uparrow$, and (b) $\mathcal{B}^*$ and P^* are disjoint.

3.3.4 Finding the radius of the smallest enclosing disk

Consider the planes H and H', as defined in Observation 1. In the dual space, the point H^*, obtained by applying the duality transform on H, lies on the boundary of $S^*_\uparrow$, and the point $(H')^*$ lies on the paraboloid P^*. Since H must contain either an edge or a face of the upper convex hull of $S_\uparrow$, this implies that H^* is either an edge or a vertex of the lower convex hull of B^*, in other words, H^* is either an edge or a vertex of $\mathcal{B}^*$. Thus we have the following:

Observation 4. Let S be a set of n points in the plane. The radius of $\mathrm{sed}(S)$ is equal to the minimum vertical distance between $\mathcal{B}^*$ and P^*.

For a point set S and query region q in a standard range tree, $S \cap q$ can be expressed as the union of the points rooted at canonical nodes $v_1, v_2, \ldots, v_m$ which fall in the desired range. For all canonical nodes v_i, let $S(v_i)$ be the set of points rooted at the node. We define $S_\uparrow(v_i)$, $S^*_\uparrow(v_i)$ and $\mathcal{B}^*(v_i)$, as described above. For the disk to contain all points in $S \cap q$, the associated plane H must be such that the points in $S_\uparrow(v_i)$, for *all* canonical nodes, lie on or below it. In other words, the plane H associated with $\mathrm{sed}(S \cap q)$ must lie on or above the convex hull of the *union* of all $S_\uparrow(v_i)$. In the dual space equivalent, what this means is that the point H^* must lie on the boundary of the *intersection* of all $S^*_\uparrow(v_i)$. By definition of $\mathcal{B}^*$, this implies that H^* must be a point on the intersection of all $\mathcal{B}^*(v_i)$.

Thus, the problem of computing $\mathrm{sed}(S \cap q)$ is equivalent to that of computing the minimum vertical distance between

- the paraboloid P^*, and
- the intersection of the convex polyhedra $\mathcal{B}^*(v_1), \mathcal{B}^*(v_2), \ldots, \mathcal{B}^*(v_m)$. We call it $\mathcal{B}^*(S \cap q)$.

3.4 Our Solution

A standard two dimensional range tree partitions q into $O(\log^2 n)$ rectangular sub-regions. The points in each region, via lifting and duality transforms, are mapped to a convex polyhedron in the dual space. To construct the polyhedron $\mathcal{B}^*(S \cap q)$ the authors, in [6], take the intersection of all these $O(\log^2 n)$ polyhedra.

We propose a solution that allows us to discard a large set of canonical nodes in the query range while preserving nodes whose points contribute to the smallest enclosing disk. More precisely, our solution reduces the number of canonical nodes (and corresponding polyhedra $\mathcal{B}^*(v_i)$) under consideration to $O(\log n)$. This reduces the search-space, and therefore query time, significantly.

3.5 Data Structures and Query Algorithm

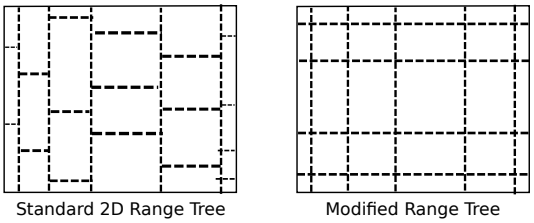

Figure 3.2: Space partitioning by the standard v/s modified range tree

We use a modified version of the two dimensional range tree, described in [23], to partition the orthogonal range into a grid, as shown in Figure 3.2. The idea used is similar to the one used in [2] to enhance kinetic *kd*-trees.

Given a point set P, we first construct a tree T_x, which is a one-dimensional range tree, on the x co-ordinates of the points. For any node v of this tree, the set of all points rooted at v is called the *canonical subset* of v. We denote the canonical subset of a node v by $P(v)$. The standard two-dimensional range tree associates each node v with a secondary tree $T_y(v)$, which is a balanced binary search tree built on the y co-ordinates of the points in $P(v)$. Our data structure differs in the method of building these secondary trees.

We build a one-dimensional range tree, T_y, based on the y co-ordinates of the entire point set P. We call this the *template tree*. For each node v, we reduce the template tree into a structure that we call the *contracted tree* for the set $P(v)$. To do so, we perform the following two operations on the template tree T_y (Figure 3.3):

- First, all subtrees that do not have a leaf in $P(v)$ are removed.
- Subsequently all nodes that have a single child are *contracted*, i.e., the child of such a node is directly connected to its parent, and the node itself is removed. This process is carried out recursively until no nodes with a single child remain.

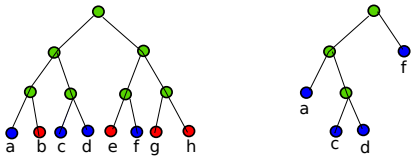

Figure 3.3: Template tree and Contracted tree for nodes a, c, d and f.

Notice that every range associated with a node of such a contracted tree is also present in T_y, as we are merely eliminating ranges that do not correspond to points in $P(v)$. Since each range in T_y corresponds to a horizontal section of the plane, the contracted tree for any node v represents a subset of these horizontal sections. By using the space partitioning produced by T_y, we can partition the space into a tiled grid, such that all space partitions induced by the contracted secondary trees will automatically be aligned with this grid. This produces the tiled space partitioning shown in Figure 3.2.

Consider the tiled alignment generated by the modified range tree. For each query we can identify blocks (tiles) that (a) together capture all points on the boundary of the point-set, and (b) therefore must contain the bounding points of $\mathrm{sed}\,(S \cap q)$. We call these blocks *candidate blocks*. These blocks alone are sufficient for computing both the convex hull and the smallest enclosing disk.

Lemma 2. The set $\mathcal{C}$ of candidate blocks can be computed in $O(\log n)$ steps, and $|\mathcal{C}| = O(\log n)$.

Algorithm 1 Finding candidate blocks in the top-right region

Initialize: $B \leftarrow$ empty list

Set: $b_x \leftarrow$ block enclosing the point with max x value

Set: $b_y \leftarrow$ block enclosing the point with max y value

1: B.append(b_x)
2: **while** $b_x \neq b_y$ **do**
3: **if** $\exists\ b_{x'} \mid b_{x'}$ is non-empty and lies vertically above b_x, in the same column **then**
4: $b_x \leftarrow b_{x'}$
5: **else**
6: $b_x \leftarrow$ left-neighbour of b_x
7: **end if**
8: B.append(b_x)
9: **end while**
10: **return** B (List of Candidate Blocks)

Proof. The set $\mathcal{C}$ of candidate blocks can be partitioned into four continuous chains based upon the extreme points in each orthogonal direction. We demonstrate how to compute the upper-right quarter (dark shaded blocks in Figure 3.4):

We begin at the block containing the point with maximum x co-ordinate. We then find the first non-empty block in the positive y direction in the same column. If such a block is found, we move to that block. If not, we move one block to the left and repeat the process until the block enclosing the point with the maximum y co-ordinate is reached. In each step we move either towards the top or towards the left, therefore we may move a maximum of $O(\log n)$ steps in both directions. All non-empty blocks visited in this process constitute the max x to max y chain of candidate blocks. Algorithm 1 describes the process rigorously.

The remaining chains can be computed in a similar manner and the results combined to get the set $\mathcal{C}$. □

In place of the polyhedron $\mathcal{B}^*$ of Section 3.3.3, we can now use a convex polyhedron $\mathcal{B}^{**}$, defined as the intersection of these $O(\log n)$ candidate blocks/nodes (as opposed to $O(\log^2 n)$ required previously).

At each candidate canonical node v, we store the *hierarchical representation* [10] of the corresponding convex polyhedron $\mathcal{B}_v$, which can be constructed in $O(|\mathcal{B}_v|)$ time and requires an equivalent amount of space. Therefore the entire tree requires $O(n \log^2 n)$ time and space.

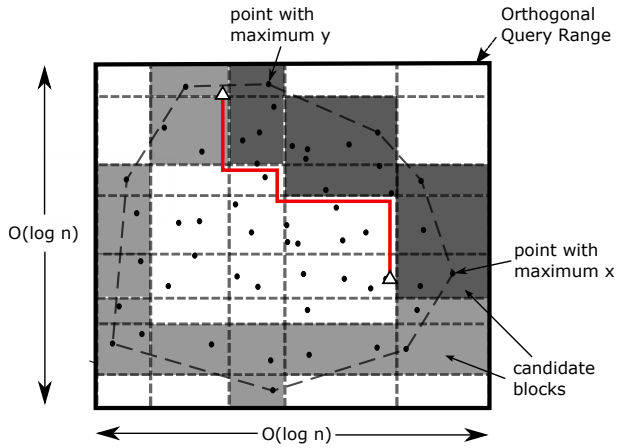

Figure 3.4: Candidate blocks

Using hierarchical representations of polyhedra and the data structures described in Section 5.2 of [6], we can compute the distance between the paraboloid P^* and a convex polyhedron A in its hierarchical representation, $hier(A)$, in $O(\log |A|)$ time.

Lemma 3 (Eppstein, 1992. [11], as given in [6])**.** 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.

We have $m = O(\log n)$ polyhedra and $\gamma = O(\log n)$, as explained above, therefore the minimum vertical distance between the paraboloid P^* and the convex polyhedron $\mathcal{B}^{**} = \bigcap_{i \in \mathcal{C}} \mathcal{B}_i$ can be computed in $O(\log^6 n)$. Thus, the radius of $\mathrm{sed}\,(S \cap q)$ can also be computed within the same time bound.

Theorem 4. *Let S be a set of points in the plane. We can construct an $O(n \log^2 n)$ size data structure in $O(n \log^2 n)$ time, such that for any axis-parallel query rectangle q, the radius of the smallest enclosing disk for the points lying in $S \cap q$ can be computed in $O(\log^6 n)$ time.*

Chapter *4*

Generalized Planar Convex Hull Range Queries

4.1 Introduction

In the generalized version of a problem, points are associated with colors. Colors capture the idea of membership, dividing objects into groups based on some common property. Such categorization has practical applications in databases, spatial information systems and other areas where objects are separable into classes and queries involve membership checking in these classes.

We present here the first solution for the generalized (colored) range-query version of the classic problem in computational geometry – *Convex Hull* – in the two dimensional setting, for a static set of points. Generalized intersection searching problems are broadly divided into two kinds, *reporting* and *counting*. In the former, the goal is to report the distinct colors whose points fall in the query range, while in the latter the goal is to count the number of such colors. Our solution is for the reporting version of Convex Hull.

4.1.1 Generalized Planar Range Convex Hull Queries

In the planar convex hull range query problem, given a set of points P and a query region q, the goal is to find the convex hull of the points in $P \cap q$. We present a solution for generalized *reporting* of the convex hull in a orthogonal range q, for which we must preprocess the points in a way such that we can report the distinct colors that constitute the convex hull of the points in $P \cap q$.

4.2 Previous Work

Brass et al. [6] studied the range convex hull problem and presented a solution using range trees and a method similar to the gift wrapping algorithm [26]. Moidu et al. [23] presented a more efficient solution, using a novel approach using a modified version of the range tree. The colored version of the range convex hull problem has also not been studied before.

4.3 Preliminaries

We assume that all points are distinct and have integer co-ordinates. The more general setting can be transformed to one with integer co-ordinates by reduction to rank space using standard methods [7][4]. Let $\mathcal{C}$ be the set of all colors. Let $n = |P|$. Clearly, $|\mathcal{C}| \leq n$ (if $|\mathcal{C}| = n$, then the problem becomes a case of standard intersection searching, without colors). We encode the colors as integers from 1 to n for notational and operational convenience.

For both the problems under consideration, let c be the number of distinct colors intersected by the query range. Our answer, therefore, will have size $O(c)$. To be output sensitive, we would therefore like our solutions to take time which is a function of c (and not a function of the total number of points on the maxima or convex hull). This rules out any method involving the computation of all maximal or convex hull points followed by some processing on them.

All queries studied in the following sections are orthogonal and axis-parallel, unless explicitly specified otherwise. All results are in the pointer machine model.

4.3.1 Generalized Reporting in One Dimension

Given a set of points in one dimension where each point has a color (not necessarily unique) associated with it, we want to preprocess the point-set such that given a query range we can efficiently report the distinct colors in that range.

Gupta et al. [16] showed a transformation which reduces the generalized one dimensional range reporting problem into the standard grounded range reporting problem in two dimensions and solved the problem using a *priority search tree* (PST) [22]. Thus, for a static set of points in one dimension, the c distinct colors intersected by a query range can be reported using a $O(n)$ space and $O(n \log n)$ preprocessing time data structure $\mathcal{D}$ which answers queries in $O(\log n + c)$ time per query. We use this result in our solutions for both maxima and convex hull.

4.3.2 Heavy-light Decomposition

Heavy-light decomposition is a technique which allows us to break down a rooted tree into a set of mutually disjoint paths. It was first used in literature by Tarjan [33] while the exact phrase was coined by Sleator and Tarjan when they used the technique in their analysis of link-cut trees [31][32].

Let T be a rooted n-ary tree. Let $\text{size}(v)$ be the number of nodes in the subtree rooted at a node v. An edge (p, q), where q is a child of p, is labeled *heavy* if $\text{size}(q) > \frac{1}{2} \cdot \text{size}(p)$ and *light* otherwise. A tree with edges labeled in this manner is said to be *decomposed*. The following properties can be shown easily for a heavy-light decomposed tree:

- At most 1 edge from a node to its children can be heavy.
- Each connected set of heavy edges forms a vertex-disjoint path. We call such a path a *heavy path*.

- A path (v, u) in the tree, where u is an ancestor of v, will consist of $O(\log n)$ light edges and $O(\log n)$ heavy paths.

4.4 Generalized Range Convex Hull Queries

We present an output sensitive algorithm to report the distinct colors on the convex hull of the points lying in a query range. We use the modified two dimensional range tree proposed by Moidu et al. [23] together with the generalized one dimensional range reporting structure of Section 4.3.1.

4.4.1 Preprocessing

We build a modified range tree $\mathcal{R}$, as described in [23], on the set of points P and supplement it with additional preprocessed data structures to support generalized range reporting of the convex hull.

Constructing $\mathcal{R}$ takes $O(n \log n)$ time. Within $\mathcal{R}$, we call the primary range tree, built using the x co-ordinates of the points, T_x. Each vertex v_i of T_x has a secondary tree associated with it, which is built using the y co-ordinates of the points rooted at v_i. We call this tree $T_y(v_i)$. At each canonical node in every secondary tree $T_y(v_i)$, we precompute the convex hull of the points rooted at $T_y(v_i)$.

Lemma 5. The convex hulls of the points rooted at the canonical nodes in $\mathcal{R}$ can be computed in a total of $O(n \log^2 n)$ time.

Proof. Consider a node v_x in the primary tree T_x. Let the number of points rooted at v_x be n_{v_x}. Therefore the total number of points in the tree $T_y(v_x)$ will also be n_{v_x}.

Instead of computing the convex hull of the points rooted at each node of $T_y(v_x)$ ab initio, we will proceed in a bottom-up fashion starting at the deepest (lowermost) level, merging the convex hulls of the children of each canonical node to get the convex hull of the points at the canonical node itself. Merging of every pair of child hulls to form the parent hull takes $O(\log (n_1 + n_2))$ time using the method described by Kirkpatrick and Snoeyink [21] to compute outer tangents for disjoint convex polyhedra. Here n_1 and n_2 are the number of points in the respective child convex hulls. Clearly, even in the worst case, $n_1 + n_2 \leq n_{v_x}$ (for tree $T_y(v_x)$). Therefore each merge step takes at most $O(\log n_{v_x})$ time. Notice that the total number of merge operations required to compute the convex hulls for all canonical nodes in $T_y(v_x)$ is the same as the total number of canonical nodes (non-leaf nodes) present in $T_y(v_x)$, i.e. $O(n_{v_x})$. Therefore, the total preprocessing time required to compute the convex hull of each canonical node in a secondary tree $T_y(v_x)$ will be $O(n_{v_x} \log n_{v_x})$, where $n_{v_x} = |T_y(v_x)|$.

Consider the set S_d of all nodes lying at depth d in the primary tree T_x. The time required to preprocess the secondary trees corresponding to each node $s \in S_d$ will be $O(n_s \log n_s)$, where n_s is the

number of nodes rooted at s. The total time required to preprocess all nodes in S_d will be

$$\sum_{s \in S_d} O(n_s \log n_s) \leq \sum_{s \in S_d} O(n_s \log n) = O(\log n \sum_{s \in S_d} n_s) = O(n \log n) \tag{4.1}$$

There are a total of $\log n$ levels in the tree T_x. Therefore, preprocessing the entire tree at the cost of $O(n \log n)$ per level will take $O(n \log^2 n)$ time. □ □

Once the convex hulls of all canonical nodes have been computed, we preprocess each canonical convex hull (convex hull of the points rooted at the canonical node) for generalized range reporting. To do this, we first linearize the list of convex hull points and then preprocess it using the data structure $\mathcal{D}$ described in Section 4.3.1. At each canonical node, we store the convex hull points in counter-clockwise order in an array. We store a pointer from each convex hull point to its index in the array, allowing lookups in both directions (array index to convex hull point and vice-versa). Using the array indices as one dimensional co-ordinates, we preprocess the array for generalized one dimensional range reporting using the data structure $\mathcal{D}$.

Lemma 6. The canonical convex hulls in $\mathcal{R}$ can be preprocessed for generalized range reporting in $O(n \log^3 n)$ time.

Proof. The data structure $\mathcal{D}$ takes $O(n \log n)$ time to build. We build one instance of $\mathcal{D}$ for every canonical node in $\mathcal{R}$. By a similar analysis as was performed in proving Lemma 5, it can be shown that the preprocessing of all canonical convex hulls in $\mathcal{R}$ will take $O(n \log^3 n)$ time. □ □

Lemma 7. The data structure $\mathcal{R}$ occupies $O(n \log^2 n)$ space.

Proof. The modified range tree of [23] utilizes $O(n \log n)$ space, as it only stores a constant amount of extra information per node. Our data structure $\mathcal{R}$, however, stores two additional items at each canonical node – the canonical convex hull and the data structure for generalized reporting. Both these structures require $O(n)$ space. Analysis similar to what is shown in the proof of Lemma 5 will show that the total space requirement of the data structure $\mathcal{R}$ is $O(n \log^2 n)$. □ □

4.4.2 Query

Moidu et al., in [23], show that using their modified range tree, given a query region q, in $O(\log^2 n)$ time we can

1. Identify $O(\log n)$ canonical nodes whose convex hulls, when merged, form the convex hull of the points in $P \cap q$. They call these nodes *candidate* nodes (or blocks).

2. Find, for each candidate node, the start and end points of a continuous segment of its convex hull which, after merging, becomes part of the convex hull of $P \cap q$. Let us call these points p_s and p_e.

For every candidate node n_c, once we obtain the points p_s and p_e, we can find the corresponding indices l and r, using the arrays populated in the preprocessing stage for generalized reporting (refer Section 4.4.1). We then do the query $[l, r]$ on the preprocessed generalized reporting data structure $\mathcal{D}$. For each of the $O(\log n)$ candidate nodes, the corresponding instance of $\mathcal{D}$ outputs the distinct colors present in the points which that node contributes to the convex hull of $P \cap q$. The set of colors returned by querying each candidate node can at most be of size c, where c is the total number of distinct colors in the convex hull of $P \cap q$. However, the same color can be output by more than one candidate node.

To ensure that each color in the maxima is reported only once, we leverage the fact that colors are encoded as integers from 1 to $\mathcal{C}$. For every query we initialize a bit-array B, of size $\mathcal{C}$, with all $B[i]$ set to 0. As colors are reported from $\mathcal{T}$, for every reported color k we check the value of $B[k]$. If $B[k] = 0$, we output color k and set $B[k]$ to 1, else we do not output color k and move on. Since there are a total of $O(c)$ colors output by $\mathcal{T}$, the aggregation process will not be a dominating factor in the query time.

Lemma 8. The data structure $\mathcal{R}$ answers range queries in $O(\log^2 n + c \log n)$ time.

Proof. For every range query of the form $[x_l, x_h] \times [y_l, y_h]$, our query algorithm performs a one dimensional generalized range reporting query on the instance of the data structure $\mathcal{D}$ stored in each of the $O(\log n)$ candidate canonical nodes. $\mathcal{D}$ has a query time of $O(\log n + c)$, where c is the number of colors reported. Hence our method requires $O(\log^2 n + c \log n)$ time per query. □ □

Thus we have the following result:

Theorem 9. *Let P be a set of colored points in two dimensions. P can be preprocessed into a $O(n \log^2 n)$ space and $O(n \log^3 n)$ preprocessing time data structure such that given an orthogonal range query q, the c distinct colors in the convex hull of $P \cap q$ can be reported in $O(\log^2 n + c \log n)$ time.*

Chapter *5*

Future Work

5.1 Smallest Enclosing Disk Range Query

We hope there can be improvements in the query efficiency for this problem, either by improving the solution proposed here, or using an alernate approach.

5.2 Generalized Convex Hull

Designing output sensitive algorithms for generalized intersection searching problems is an challenging problem since the query time must depend on the number of colors and not on the number of points in the result. The problem discussed in the preceding sections presents interesting possibilities in the design of solutions for generalized geometric range aggregate query problems. An immediate open question is that of proposing a more efficient solution – Improvements in the runtime by a logarithmic factor should be possible.

Furthermore, we have not tackled the counting version of convex hull where the result is simply the number of colors. Developing efficient algorithms for the counting version remains an open line of pursuit. All results here are for a static set of points. Developing solutions for the dynamic case is still an open problem. Solutions to this problem in higher dimensions are also yet to be proposed.

Bibliography

[1] *Proceedings of the 22nd Annual Canadian Conference on Computational Geometry, Winnipeg, Manitoba, Canada, August 9-11, 2010*, 2010.

[2] M. A. Abam, M. de Berg, and B. Speckmann. Kinetic kd-trees and longest-side kd-trees. *SIAM J. Comput.*, 39(4):1219–1232, 2009.

[3] P. K. Agarwal, S. Govindarajan, and S. Muthukrishnan. Range searching in categorical data: Colored range searching on grid. In R. H. Möhring and R. Raman, editors, *ESA*, volume 2461 of *Lecture Notes in Computer Science*, pages 17–28. Springer, 2002.

[4] S. Alstrup, G. S. Brodal, and T. Rauhe. New data structures for orthogonal range searching. In *FOCS*, pages 198–207. IEEE Computer Society, 2000.

[5] P. Bozanis, N. Kitsios, C. Makris, and A. K. Tsakalidis. New upper bounds for generalized intersection searching problems. In Z. Fülöp and F. Gécseg, editors, *ICALP*, volume 944 of *Lecture Notes in Computer Science*, pages 464–474. Springer, 1995.

[6] P. Brass, C. Knauer, C. S. Shin, M. Smid, and I. Vigan. Range-aggregate queries for geometric extent problems. In *CATS: 19th Computing: Australasian Theory Symposium*, 2013.

[7] B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17(3):427–462, 1988.

[8] A. S. Das, P. Gupta, A. K. Kalavagattu, J. Agarwal, K. Srinathan, and K. Kothapalli. Range aggregate maximal points in the plane. In M. S. Rahman and S.-I. Nakano, editors, *WALCOM*, volume 7157 of *Lecture Notes in Computer Science*, pages 52–63. Springer, 2012.

[9] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 2008.

[10] D. P. Dobkin and D. G. Kirkpatrick. Determining the separation of preprocessed polyhedra - a unified approach. In M. Paterson, editor, *ICALP*, volume 443 of *Lecture Notes in Computer Science*, pages 400–413. Springer, 1990.

[11] D. Eppstein. Dynamic three-dimensional linear programming. *INFORMS Journal on Computing*, 4(4):360–368, 1992.

[12] T. Gagie, J. Kärkkäinen, G. Navarro, and S. J. Puglisi. Colored range queries and document retrieval. *Theor. Comput. Sci.*, 483:36–50, 2013.

[13] P. Gupta, R. Janardan, and M. Smid. Computational geometry: Generalized intersection searching. In *Handbook Of Data Structures And Applications, D. Mehta and S. Sahni (Editors), Chapman & Hall/CRC, Boca Raton, FL*, pages 1–17. CRC Press, 2005.

[14] P. Gupta, R. Janardan, and M. H. M. Smid. Efficient algorithms for generalized intersection searching on non-iso-oriented objects. In *Symposium on Computational Geometry*, pages 369–378, 1994.

[15] P. Gupta, R. Janardan, and M. H. M. Smid. Algorithms for generalized halfspace range searching and other intersection searching problems. *Comput. Geom.*, 5:321–340, 1995.

[16] P. Gupta, R. Janardan, and M. H. M. Smid. Further results on generalized intersection searching problems: Counting, reporting, and dynamization. *J. Algorithms*, 19(2):282–317, 1995.

[17] P. Gupta, R. Janardan, and M. H. M. Smid. Algorithms for generalized halfspace range searching and other intersection searching problems. *Comput. Geom.*, 6:1–19, 1996.

[18] P. Gupta, R. Janardan, and M. H. M. Smid. A technique for adding range restrictions to generalized searching problems. *Inf. Process. Lett.*, 64(5):263–269, 1997.

[19] R. Janardan, P. Gupta, Y. Kumar, and M. H. M. Smid. Data structures for range-aggregate extent queries. In *CCCG*, 2008.

[20] R. Janardan and M. A. Lopez. Generalized intersection searching problems. *Int. J. Comput. Geometry Appl.*, 3(1):39–69, 1993.

[21] D. Kirkpatrick and J. Snoeyink. Computing common tangents without a separating line. In *Algorithms and Data Structures*, volume 955 of *Lecture Notes in Computer Science*, pages 183–193. Springer Berlin Heidelberg, 1995.

[22] E. M. McCreight. Priority search trees. *SIAM J. Comput.*, 14(2):257–276, 1985.

[23] N. Moidu, J. Agarwal, and K. Kothapalli. Planar convex hull range query and related problems. In *CCCG*. Carleton University, Ottawa, Canada, 2013.

[24] Y. Nekrich and M. H. M. Smid. Approximating range-aggregate queries using coresets. In *CCCG* [1], pages 253–256.

[25] F. Nielsen and R. Nock. A fast deterministic smallest enclosing disk approximation algorithm. *Inf. Process. Lett.*, 93(6):263–268, 2005.

[26] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1998.

[27] S. Rahul, H. Bellam, P. Gupta, and K. Rajan. Range aggregate structures for colored geometric objects. In *CCCG* [1], pages 249–252.

[28] S. Rahul, A. S. Das, K. S. Rajan, and K. Srinathan. Range-aggregate queries involving geometric aggregation operations. In N. Katoh and A. Kumar, editors, *WALCOM*, volume 6552 of *Lecture Notes in Computer Science*, pages 122–133. Springer, 2011.

[29] R. Sharathkumar and P. Gupta. Range-aggregate proximity detection for design rule checking in vlsi layouts. In *CCCG*, 2006.

[30] Q. Shi and J. JáJá. Optimal and near-optimal algorithms for generalized intersection reporting on pointer machines. *Inf. Process. Lett.*, 95(3):382–388, 2005.

[31] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. In *STOC*, pages 114–122. ACM, 1981.

[32] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.

[33] R. E. Tarjan. Applications of path compression on balanced trees. *J. ACM*, 26(4):690–715, 1979.