Efficient Data Structures and Algorithms for Range-Aggregate problems

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

MS by research
in
Computer science

by

Jatin Agarwal
201007003
`jatin.agarwal@research.iiit.ac.in`

Center for Security,Theory, and Algorithmic Research
International Institute of Information Technology
Hyderabad - 500 032, INDIA
July 2013

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled “Efficient Data Structures and Algorithms for Range-Aggregate problems” by Jatin Agarwal, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Adviser: Dr. Kannan Srinathan
Date	Co-Adviser: Dr. Kishore Kothapalli

To My Parents, My Guide, My Friends and ‘The Truth’

Acknowledgments

This thesis would not have been possible without the help of many people due to various reasons. First and foremost, I would like to express my sincere gratitude to my advisor Dr. Kannan Srinathan for continuous encouragement in exploring various research areas irrespective of his own interest and providing financial support to the best of his capacity. He also embedded 'Art of Abstract Thinking' in me that helped me to tear apart physical realities and understand core concepts. I would also like to express my sincere gratitude to my co-advisor Dr. Kishore Kothapalli for his continuous efforts in keeping track of my efforts and directing me towards more practical possibilities. I am very thankful to both of them for all the professional and informal discussions that enhanced my learning and changed my perception. I would also like to thank Dr. Ashok Kumar Das for providing me clarity on core concepts of Discrete Mathematics and Cryptography during my association with him as teaching assistant.

Besides my advisor, I should sincerely thank four of my lab mates at CSTAR, more than any one else. 1. Ananda Swarup Das, for introducing me to Computational Geometry and encouraging me during my early research career. 2. K Anil Kishore, for being my best friend in lab and making me a more patient person with regard to decision making. 3. Nadeem Moidu, for rigorous brain-storming discussions and sharing of ideas across our research discipline. 4. V. Dhrma Teja for being available all the time for both technical and informal discussions.

I thank all my fellow lab mates in CSTAR: Pruthvi, Ravi Kishore, Chiranjeev, Akshat, Madhu, Manan, Rohan, Aman, Dharmeet, Sankalp, Tushar, Rohan, Rajeev, Niraj and Piyush for changing my perception about life through many valuable discussions. Special thanks to Sankalp for helping me in editing and documenting content of all my technical writings.

I would also like to thank all my friends from MS 2010 batch for being there and helping me whenever I needed them. Special thanks to my friends Vikram, Nikhil and Srk who kept "my drive to pursue random ideas" always in check. I would also like to thank my other thesis committee members: Dr. Suresh Purini and Dr. Vikram Pudi for giving valuable feedback and suggestions on future extensions of my work. Last but no the least, I would like to thank my family: my parents Smt. Madhu Agarwal and Sri. Ashok Agarwal, my sisters Nidhi Agarwal, Kanika Gupta and brother-in-law Mayank Chawla for always being very supportive.

Abstract

Computational Geometry is regarded as an emerging subfield of computer science. In computational geometry the *standard range searching* problems are studied widely because they have many real-world applications in spatial informatics, VLSI design, geographical information systems, information retrieval, databases, computer vision, computer graphics and other areas. In the *standard range searching* problem on an object set S we either report points or count the number of points in a given query region q. But just reporting and counting points in the query region is not sufficient for real world applications like data mining where data analysis is the major concern. In order to support applications which require data analysis to capture significant features we compute an aggregation function over the given query region like Sum, Max, Average, etc. Range Queries with such aggregate functions are called range-aggregate queries. But to capture intricate geometric properties on the data set we do geometric aggregation. Some of the examples of geometric aggregation functions are skyline, convex hull, closest pair, Voronoi diagram, triangulation and so on.

However, advances in the creation and archival of digital information have led to an information explosion and therefore the number of objects inside a query range can be huge. In such cases, reporting a sample of the result set is preferred. In this work we present solutions to two such geometrical problems which give such samples. Both skyline and convex hull concepts have wide applications to other problems like outlier detection, clustering and so on. Abstractly both skyline and convex hull concepts provide some type of ordering in higher dimensions. Intuitively we all know that ordering on a set of objects makes it more efficient for answering relevant queries by capturing significant features.

Let S be a set of n points in the two dimensional plane. A point $p = (p_x, p_y)$ is said to *dominate* another point $q = (q_x, q_y)$ if $p_x > q_x$ and $p_y > q_y$. Points *not* dominated by any point in the set are called *maximal* or *skyline points*. We preprocess S into a data structure such that we can efficiently compute maximal points or skyline for any given query q. We propose a linear space data structure which takes $O(\log^{\epsilon} n + k)$ query time for any given query q in the *word RAM* model where k is the total number of points that are reported. We solve this problem for 2-sided quadrant queries and 3-sided range queries. We not only report points inside the given query region but we also count the number of points in $S \cap q$.

The convex hull of any point set P in the Euclidean plane is the smallest convex polygon that contains P. Given a set of n points $P = \{p_1, \ldots, p_n\}$ in $\mathbb{R}^2$, we preprocess P into a data structure such that we can efficiently answer convex hull queries for any given range query $q = [x_{lt}, x_{rt}] \times [y_b, y_t]$. We also propose an $O(n \log^2 n)$ space data structure which takes $O(\log^3 n + h)$ time to answer any given query q where h is the total number of points on the convex hull. We also propose a similar data structure for related problems like counting number of points on the convex hull, finding the area of the convex hull and the perimeter of the convex hull. These related problems seem trivial once we report points on the convex hull but reporting points of convex hull takes $O(\log^3 n + h)$ time where $h \leq n$ is the total number of points on the convex hull. We propose *output-sensitive* algorithmic techniques to find the number of points on the convex hull and its area and perimeter without having to compute the convex hull which makes our results independent of h. So that it takes $O(\log^3 n)$ time to answer these queries.

Related Publications

- *On Generalized Planar Skyline and Convex Hull Range Queries* (Accepted as full paper), Nadeem Moidu, Jatin Agarwal, Sankalp Khare, Kannan Srinathan, Kishore Kothapalli. *Conference: The 8th Workshop on Algorithms and Computation(WALCOM 2014)*
  The work done in this paper is a joint work with Nadeem Moidu and Sankalp Khare is part of their M.S. dissertations.

- *Improved bounds for Smallest Enclosing Disk Range Queries*(Accepted as full paper), Sankalp Khare, Jatin Agarwal, Nadeem Moidu, and Kannan Srinathan *Conference: To appear in the proceedings of the 16th Japan Conference on Discrete and Computational Geometry and Graphs (*$JCDCG^2$2013*)*
  Work done in this paper is a joint work with Sankalp Khare, and is part of his M.S. dissertation.

- *Planar Convex Hull Range Query and Related Problems*(Accepted as full paper), Nadeem Moidu, Jatin Agarwal and Kishore Kothapalli *Conference: To appear in proceedings of the 25th Canadian Conference on Computational Geometry(CCCG 2013)*
  Work done in this paper is a joint work with Nadeem Moidu, and is part of his M.S. dissertation.

- *Efficient Range Reporting of Convex Hull*(Submitted as e-print), Sankalp Khare, Jatin Agarwal, Nadeem Moidu, Kishore Kothapalli and Kannan Srinathan *Conference: Computing Research Repository(*$CoRR$*)*
  The work done in this paper is presented in Chapter 4 and Chapter 5.

- *On Counting Range Maxima Points in Plane* (Accepted as full paper), Anil Kishore Kalavagattu, Jatin Agarwal, Ananda Swarup Das, and Kishore Kothapalli. *Conference: The 23rd International Workshop on Combinatorial Algorithms(IWOCA 2012)*
  The work done in this paper is a joint work with A. K. Kalavagattu, and is part of his M.S. dissertation.

- *Range Aggregate Maximal Points in the Plane.* (Accepted as full paper), Ananda Swarup Das, Prosenjit Gupta, Anil Kishore Kalavagattu, Jatin Agarwal, Kannan Srinathan, Kishore Kothapalli. *Conference: The 6th Workshop on Algorithms and Computation(WALCOM 2012)*
  The work done in this paper is a joint work with A. S. Das, and is part of his Ph.D. dissertation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter we provide a brief overview of computational geometry by explaining different types of problems and different models of computation. We explain the importance of output sensitive results to get a better understanding of the work presented in this thesis. An overview of problems studied in this work is provided in Section 1.2 and Section 1.3. We show that the problems solved are non-trivial in nature by discussing the difficulty level of the problems. Finally, we conclude this chapter with a brief outline of the thesis.

1.1 Computational Geometry: A brief overview

Computational geometry, as name indicates, is the field of solving geometrical problems using computation. It emerged as a branch of computer science in the 1970s. It is concerned with the design and analysis of algorithms for computing properties of geometric objects in euclidean space. The simplicity of the problem statements coupled with elegant and generalized solutions make it a very interesting field of research. Most geometrical problems are either in the $2D$-plane or in the $3D$-space. Most of the geometrical problems in literature are discrete in nature and the role of geometry and numerical computations is fairly small. In computational geometry our main interest is in the combinatorial elements of the problems unlike analytic or differential geometry. For a discrete algorithm designer, computational geometry is easy to start with and fun to study. Computational geometry plays a fundamental role in various application domains like geographical information systems(GIS), VLSI design [18, 17], geometric modeling, computer vision [36], computer graphics, robotics, computational topology and others. For a taste of the ubiquity, practical relevance and beauty that characterize this field refer [40, 1]. In order to understand the practical importance of computational geometry refer [34].

1.1.1 Types of Problems

Problems considered in the field of computational geometry can be classified into two categories: (a) *single-shot* problems and (b) *multi-shot* problems. Furthermore, there are static and dynamic versions in each of these categories. If the given input for a problem remains unchanged, it is a *static* input problem. In a *dynamic* input setting, objects can either be removed from the input set or new ones added over time. Note that all problems solved in this thesis are on a *static input* set of objects.

In *single-shot* problems, for any given input set we just compute the required output in a single instance with no further computations. However, in *multi-shot* problems we preprocess the input set into a data structure such that multiple successive queries may be answered efficiently.

Here some classic examples of single-shot problems which are also fundamental problems in computational geometry.

- **Convex Hull:** Given a set S of points in a plane, find the smallest convex polygon containing all the points of S. Convex hull is the most fundamental structure in computational geometry. It is the simplest shape approximations for a set of points. Chan's Algorithm [10] is an optimal,

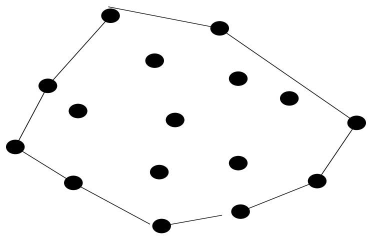

Figure 1.1 Convex hull of a finite set of points

output-sensitive algorithm to compute the convex hull of a finite set of points in 2 or 3 dimensional space in $O(n \log h)$ time, where h is the number of points on the boundary of the convex hull.

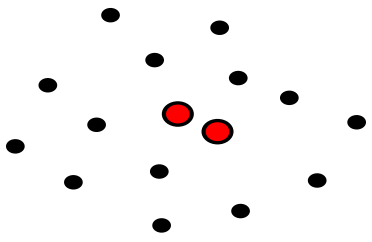

Figure 1.2 Closest pair of a finite set of points

- **Closest pair problem:** The problem of finding the closest pair of points was among the first problems studied at the beginning of the systematic study of the computational complexity of geometric algorithms [33]. Given a set S of n points in d-dimensional euclidean space, find a pair of points with the smallest distance between them. For a set of 2-dimensional points in a plane, a simple *divide-and-conquer* technique exist to obtain an $O(n \log n)$ time algorithm.

- **Voronoi Diagram** Given a set S of n points on a plane find the Voronoi diagram for the given set S. A Voronoi diagram partitions space into subregions(cells) such that each cell contains the one point from the set S, this point is called seed. Adjacent cells are separated from each other by the perpendicular bisector of seeds. The voronoi diagram is one of the most widely

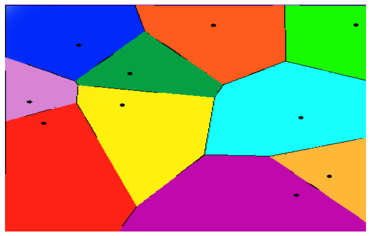

Figure 1.3 Voronoi Diagram of a finite set of points

used concepts both in theory and practice. Voronoi diagram has huge applications in geometry, computational chemistry, epidemiology, ecology, machine learning, databases, climatology, etc. Voronoi diagram also acts as the preprocessed data structure for a wide range of problems in *multi-shot* scenario.

In a *multi-shot* scenario, the input set is preprocessed into a data structure and output depends on the query. We make multiple queries for a *multi-shot* problem, so we try to ensure that each query is answered efficiently at the cost of space and preprocessing. Typically in *multi-shot* problems a trade-off between space complexity and time complexity is observed. We can impose several restrictions on the query and based on this restrictions a subset of the input set get selected as the output. *Generalized intersection searching (GIS)* [24] problems constitute an important class of *multi-shot* problems. *Standard range (intersection) searching* problems have been studied extensively in the literature of computational geometry. In general, a range searching problem consists of preprocessing a set S of input objects such that given a query object q, we need to determine which objects of P intersect with q. If the query requires us to report all the objects of $P \cap q$, it is called a *reporting query*. If only the number of objects in $P \cap q$ is of interest, it is called a *counting query*. Typical object types in a plane include points, lines, polygons, line segments etc. and query object types q include axis parallel rectangles, half spaces, circles, spheres etc. For an excellent survey on range searching and related problems refer [3]. Now we give some examples of range searching problems.

- **Orthogonal Range Reporting:** Given a set S of n points in $\mathbb{R}^2$, we preprocess S into a data structure such that we can efficiently **report** points inside given orthogonal range query q.

- **Orthogonal Range Counting:** Given a set S of n points in $\mathbb{R}^2$, we preprocess S into a data structure such that we can efficiently **count** the number of points inside given orthogonal range query q.

Note that both problems handled in this thesis are on orthogonal range querying in a plane. Thus our focus in the following sections will be to give a brief overview of *output-sensitivity* and models of computation.

1.1.2 Output Sensitivity

All algorithms proposed in this thesis are *output sensitive* in nature. An *output sensitive algorithm* is one which attempts to be more efficient for small output sizes and running time is described as an asymptotic function of both input size and output size. For example consider the points in an array A of size n, given a query $q \in R$ such that we have to report all elements of $A \geq q$. If the query is made only once then a linear scan of array A is the most efficient way of answering it. But if multiple queries are made, spending $O(n)$ time for each query is inefficient, so we preprocess A into a data structure A' such that each query is answered efficiently. So we preprocess A either into a balanced binary tree T or a sorted array A'. Now if a query is made we either perform a binary search in A' or search in T to find the first element $e \geq q$ and it takes $O(\log n)$ time. Now we also have to report all the elements $\geq q$. Let the number of elements in $A' \geq q$ be k. So the total query time is $O(\log n + k)$ where k is the output size for any given query q. It means query time is a function of both input size and output size and the algorithm proposed is *output sensitive*.

1.1.3 Models of computation

There are many different model of computations based on storage assumptions. For example some of them are *Turing Machine*, *Comparison-based Model*, *Pointer Machine Model*, *word-RAM model*, *External Memory*, *Cell-Probe model*, *Distributed Models*, *Parallel Models* and *streaming model*. Our focus in this section will be on the *Pointer Machine* and *word-RAM models* because the solutions we discuss are on these models. The results discussed in Chapter 2 are in the *word RAM model* and those

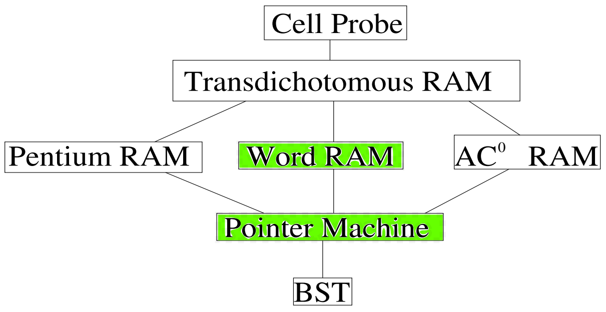

Figure 1.4 Hierarchy of the Computation models

discussed in Chapters 3 and 4 are in the *pointer machine model*. In both models, memory is seen as a collection of words of some specific size and a fixed cost is assigned for each basic operation. We also make the *fixed universe assumption*, i.e. we assume that points lie on a $n \times n$ grid and that point coordinates are integers in $[1, n]$. We can reduce the more general case to this one by reduction to rank space [21].

A sorting lower bound of $O(n \log n)$ time has been proved on any given set of n numbers in the *comparison based model*. In this type of modeling, comparison between values of elements is the most fundamental operation based on which the rest of the algorithm or data structure is constructed. Algorithms that access input only for comparison are said to be *Comparison-based Algorithms*. No manipulation or computation is allowed on the input.

The *word-RAM model* attempts to realistically model a computer based on the word size. It assumes unlimited memory of w-bit words and reading/writing of a word takes $O(1)$ time. It is also possible to perform all arithmetic and logical operations in $O(1)$ time. Generally in the *word-RAM* model the word size w is assumed to be greater than or equal to $\log n$ where n is the size of the input. It means based on the problem size n, the word size of the computer we use also changes. It is obvious that if $w < \log n$ then we cannot store a pointer to input in a single word and this holds on almost any every computer.

The *cell probe model* at the top as shown in Figure 1.4 is the strongest model, where we only pay for accessing memory (reads or writes). Memory cells have some size w, which is a parameter of the model. The model is non-uniform, and allows memory reads or writes to depend arbitrarily on past cell probes. Though not realistic to implement, it is good for proving lower bounds.

In the next couple of sections we provide relevant definitions and problem statements needed to understand this thesis. Finally we discuss the significance of our solutions by understanding the difficulties faced and the strategies used in solving the problems that we dealt with.

1.2 Problem 1: Sub-logarithmic Range Maxima

Previously we have defined problems on orthogonal range searching which either report or count the number of objects inside $S \cap q$. But for real-world applications just reporting and counting of objects inside a given orthogonal range is not sufficient. The need for aggregation of objects in applications like information retrieval(IR), on-line analytical processing (OLAP), geographical positioning systems, spatial informatics [39] was felt. Here one needs to compute an aggregate function of the objects in $S \cap q$. Aggregate functions computed in a given range query are referred as *Range-Aggregate queries* [35]. For example functions like SUM, MAX, MEDIAN, AVERAGE ... etc. Another example of aggregation can be *top*- k, where the goal is to report the k most significant points from $S \cap q$. All these aggregations are for weighted points where each point is assigned with a specific weight as part of the input. There is another type of aggregation called geometric aggregation where the nature of the aggregation function is geometrical. Some examples of geometrical aggregate [23, 26] function are skyline, convex hull, width [11], diameter [19], closest pair [38], smallest enclosing disk and many more. Thus geometric aggregate functions capture very intricate geometrical properties that provide better analysis of the given input. In this thesis we look at *Range maxima* in a plane and *Range Convex hull* in a plane as aggregate function.

In this section we provide brief a overview of problems discussed in Chapter 2. We also provide the necessary definitions, notations and formal statements. A brief account of difficulties faced while solving these problems and strategies used to encounter these difficulties is provided.

1.2.1 Definitions

Let S be a set of n points in the two dimensional plane. A point $p = (p_x, p_y)$ is said to *dominate* [37] another point $q = (q_x, q_y)$ if $p_x > q_x$ and $p_y > q_y$. Points *not* dominated by any point in the set are called *maximal* or *skyline points*. We use $x(p)$ to represent the x-coordinate of a point p and $y(p)$ to represent its y-coordinate. We also denote points with minimum x-coordinate, maximum x-coordinate, minimum y-coordinate and maximum y-coordinate of point set $P \cap q$ by x_{min}, x_{max}, y_{min} and y_{max} respectively. We define many types of queries based on different range restrictions on both x-coordinate

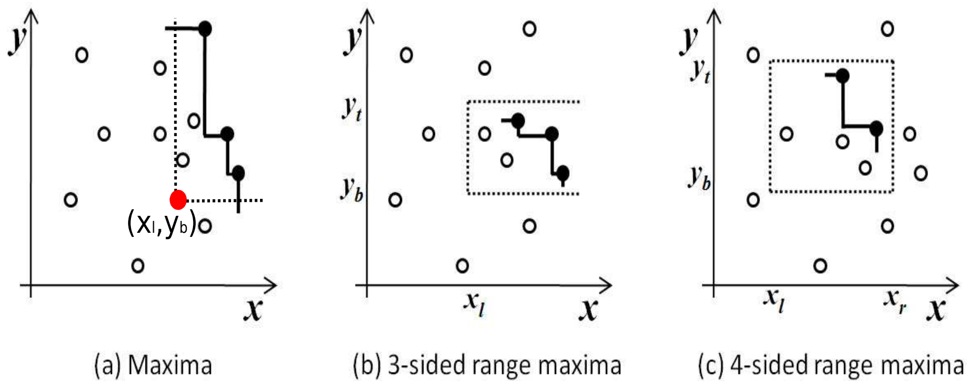

Figure 1.5 Types of queries

and y-coordinate. A quadrant query $q = [x_l, +\infty] \times [y_b, +\infty]$ is unbounded on the right towards x_{max} and unbounded on the top towards y_{max} as shown in Figure 1.5(a). The problem of finding maximal points in a quadrant query is called 2-*sided range maxima*. A 3-sided range query $q = [x_l, +\infty] \times [y_b, y_t]$ is unbounded on the right towards x_{max} as shown in Figure 1.5(b). The problem of finding maximal points in this 3-sided range query is called 3-*sided range maxima*. An orthogonal range query or 4-sided range query $q = [x_l, x_r] \times [y_b, y_t]$ is an axis parallel rectangle bounded on all four sides as shown in Figure 1.5(c). For each of these three different queries we compute skyline shown as filled points connected using orthogonal (staircase) lines as shown in Figure 1.5. Note that we solve only 2-sided range maxima and 3-sided range maxima as defined above.

1.2.2 Planar Range Maxima

Sorting is the most commonly studied problem in computer science. In one dimensional space if we want to provide some type of ordering on objects, we just sort them and things become simpler. Similarly given 2-dimensional object set, we want to provide some ordering on them to get a simpler view of whole set. Skyline and Convex Hull are two such mechanism to get a simpler view of point set in 2-dimensional space. Now we discuss an algorithm to report skyline or maximal points. We use a

Algorithm 1: *Sweep-line approach for Skyline*

```
  Data: Point set $S$ of size $n$
  Result: Maximal points or skyline points
1 Sort the point set $S$ into decreasing order of $y$-coordinates, denoted by $< p_1, p_2, \ldots, p_n >$.;
2 $x_{inf} \leftarrow -\infty$;
3 for $i = 1$ to $n$ do
4     if $x(p_i) > x_{inf}$ then
5         Report the point $p_i$;
6         $x_{inf} \leftarrow x(p_i)$;
7     end
8 end
```

sweep-line [7] approach to find skyline points. Consider a set S of n points in 2-dimensional plane. In step 1 of Algorithm 1 we sort the points in set P in decreasing order of y-coordinates into a sequence $\langle p_1, p_2, \ldots, p_n \rangle$. In step 2 we initialize an invariant x_{inf} of x-coordinate to minus infinity. In steps 3-7 we run a loop to check condition on each point of the n points. In every iteration of the loop we check whether the x-coordinate of point p_i is greater than the invariant x_{inf}. If above condition is true then we assign the x-coordinate of point p_i to the invariant x_{inf}. Traversing all points in decreasing

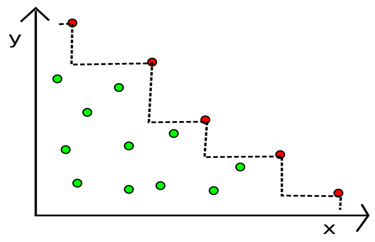

Figure 1.6 Skyline of a finite set of points

order of their y-coordinates can be thought of as a horizontal line sweep from y_{max} while maintaining only the current maximum x-coordinate with each sweep. During a sweep, whenever we find a point x-coordinate greater than the invariant, we report it and reset the invariant to the current maximum x. Darkened points along the staircase indicate maximal points as shown in Figure 1.6. This algorithm takes $O(n \log n)$ time to compute the skyline, dominated by sorting step. If the points were already sorted, it would have taken $O(n)$ time. For a one time query this seems to be optimal but when multiple queries are made we need an efficient data structure to answer these queries. In this thesis we are inter-

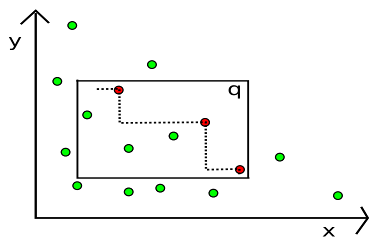

Figure 1.7 Range Maxima for a given query

ested in the query version of this problem. For the same set of points we compute skyline on a given a query range q. Observe that the skyline for a given query q, as shown Figure 1.7, is different from the skyline obtained for the entire set S in Figure 1.6. In the query version of the problem we only consider points common to set S and the query rectangle q. In Chapter 2 we provide an optimal solution for both the reporting and counting versions of the skyline problem in the word-RAM model. We restrict our scope to quadrant queries and 3-sided range queries and conjecture that such optimal bounds may also exist for the corresponding 4-sided orthogonal range query.

1.2.3 Formal Problem Statements

Given a set S of n points in a plane and an orthogonal query Q of the form $[a, b] \times [c, d]$ where $[a, b]$ is a range on the x-axis and $[c, d]$ is a range on the y-axis, report the skyline points in $S \cap Q$.

Problem 1: **Report** the skyline inside $S \cap Q$. Here $Q = [a, +\infty] \times [c, +\infty]$.

Problem 2: **Count** the number of points on the skyline of $S \cap Q$. Here $Q = [a, +\infty] \times [c, +\infty]$.

Problem 3: **Report** the skyline inside $S \cap Q$. Here $Q = [a, +\infty] \times [c, d]$.

Problem 4: **Count** the number of points on the skyline of $S \cap Q$. Here $Q = [a, +\infty] \times [c, d]$.

1.2.4 Difficulties and Strategies used

Given a *multi-shot* problem with input size n and output size k we have to find an data structure that maintains a proper trade-off between space and query time. An ideal solution for such a problem is to preprocess it into a linear data structure of size $O(n)$ such that any query takes polylogarithmic time. We already proposed a logarithmic time solution for the skyline problem [29] in the *pointer-machine model*. In our search for a sublogarithmic solution in the word-RAM, the major difficulty was to reduce search space. Our solution uses recently proposed data structures in [12, 33] that are efficient with respect to space. These data structures take sublogarithmic time to answer range successor queries.

1.3 Problem 2: Planar Range Convex hull

1.3.1 Definitions

Given a set of n points $P = \{p_1, \ldots, p_n\}$ in $\mathbb{R}^2$. We use $x(p)$ to represent the x-coordinate of a point p and $y(p)$ to represent its y co-ordinate. The convex hull of any point set S in the Euclidean plane is the smallest convex polygon that contains P. Henceforth, when we refer to *convex hull points*, we essentially mean the points on the boundary of the convex hull. The *convex hull* of any set P is the intersection of all convex sets that contain P, or the smallest convex set that contains S. We are given a set P of n points in $\mathbb{R}^2$ and a query $q = [x_{lt}, x_{rt}] \times [y_b, y_t]$. We find the convex hull on point set $P \cap q$, denoted by $ch(P \cap q)$. We also denote points with minimum x-coordinate, maximum x co-ordinate, minimum y co-ordinate and maximum y co-ordinate of point set $P \cap q$ by x_{min}, x_{max}, y_{min} and y_{max} respectively.

1.3.2 Planar Convex hull

Informally a convex hull is the minimum convex polygon that contains all given points. For example given a set of nails on a board if we leave a rubber band around the nails then we get a polygon containing all the nails inside a convex boundary. The points touching the rubber band are called *convex hull points*. Consider a set P of points as shown in Figure 1.8. We will look at some convex hull algorithms in Section 2.2 of Chapter 2. In this thesis we are interested in the query version of the convex hull problem. We refer to this problem as the *planar range convex hull* problem in the rest of this thesis. For any given orthogonal range query q we need to find a convex hull inside $P \cap q$ as shown in Figure 1.8.

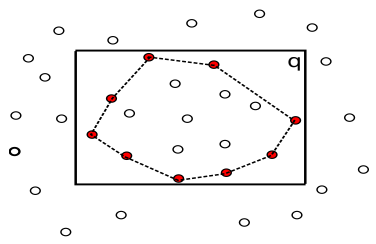

Figure 1.8 Range Convex hull for a given query

1.3.3 Formal Problem Statements

We are given a set P of n points in $\mathbb{R}^2$ and a query $q = [x_{lt}, x_{rt}] \times [y_b, y_t]$. We wish to pre-process P into a data structure such that given an orthogonal query region q, we can efficiently

Problem 1: **Report** the points on convex hull of $P \cap q$.

Problem 2: **Count** the number of points on the convex hull of $P \cap q$.

Problem 3: Find the **area** of the convex hull of $P \cap q$.

Problem 4: Find the **perimeter** of the convex hull of $P \cap q$.

1.3.4 Difficulties and strategies

A brute force solution for finding the convex hull for a given orthogonal range query q is to first report points $k \leq n$ inside $P \cap q$ using a standard range reporting technique. It takes $O(\log n + k)$ time to report points inside $P \cap q$. Now we can run Chan's algorithm [10] on this k points to obtain a convex hull in time $O(k \log h)$ where $h \leq k$ is number of the points on the convex hull inside $P \cap q$. Therefore total time taken for such a strategy would be $O(\log n + k) + O(k \log h) = O(\log n + k \log h)$. If $k = O(n)$ then this solution is as bad as $O(n \log h)$. For a single query this seems to be the optimal solution but if multiple queries are made this clearly seems inefficient.

For the first time in literature Brass et al. [8] a provided solution for the *planar range convex hull* problem. They proposed a data structure of size $O(n \log^2 n)$ with a query time of $O(\log^5 n + h)$. We improved their result by providing a data structure of $O(n \log^2 n)$ space with a query time of $O(\log^3 n + h)$. We know that $ch(P \cap q) \subseteq SL(P \cap q)$ where $ch(P \cap q)$ is set of the convex hull points inside $P \cap q$ and $SL(P \cap q)$ is the set of skyline points inside $P \cap q$ respectively. A logarithmic $O(\log n + k)$ query time solution for planar range skyline problem exists as explained in [30, 29]. Intuitively we thought that a similar bound must even exist for the *Planar range convex hull problem*. So the major difficulty that we faced in reducing the existing bound from $O(\log^5 n + h)$ to $O(\log^2 n + h)$ was how to process a given set of $O(\log^2 n)$ regions such that the query time gets reduced. In our technique we preprocess the given set of points into a range tree and store some extra information at each node of the secondary tree. So given any orthogonal query it gets divided into $O(\log^2 n)$ regions. Our technique is similar to *Graham Scan method* and is used to process information stored at nodes representing $O(\log^2 n)$ regions.

1.4 Flow of the thesis

In this chapter we gave a brief introduction to the field of computational geometry and an overview of the problems we solve and their significance. In Chapter 2 we discuss fundamental data structures and algorithmic techniques required to understand the work presented in this thesis. We propose an optimal solution for problems on *Planar Range Skyline* in Chapter 3. In Chapter 4, we propose the solution to *Planar Range Convex hull*. In Chapter 5, we propose solutions for several related problems to *Planar Range Convex hull* like counting, area and perimeter. We summarize the proposed solutions, give details of possible extensions to the problems and conclude in Chapter 6.

Chapter 2

Fundamental Data Structures and Algorithmic Techniques

The fundamental problems and properties of geometric structures have lead to the emergence of the field called computational geometry. In the past four decades it has emerged as a sub-field of algorithm design and has developed many data structures and algorithmic techniques of its own. The most fundamental problem in computational geometry is that of range searching which has led to the invention of many new data structures. Some commonly used data structures for range searching are segment trees, interval trees, range trees, k-d trees, priority search trees etc. In this work, we use range trees extensively equipped with skyline and convex hull concepts. In this chapter we provide a brief study of the basic data structures and algorithmic techniques employed in this thesis.

2.1 Range Trees

The standard range tree is the most commonly used data structure for handling orthogonal range queries on a d-dimensional point set. It was proposed by J. L. Bentley [6]. The main idea behind a range tree was to successively divide or decompose given range query into many ranges in each dimension.

For example consider a 2-dimensional point set P of size n such that given any orthogonal query $q = [x_{lt}, x_{rt}] \times [y_b, y_t]$ where (x_{lt},x_{rt}) and (y_b,y_t) are x-coordinates and y-coordinates of the query q as shown in the Figure 2.1. Consider the point set P and sort it in non-decreasing order based on x-coordinates. Based on median of x this sorted point set get divided it into two groups such that all points on the left group forms a left subtree and points on right group forms a right subtree. Recursively each subtree get divided into left and right subtree till each point is represented as the leaf of the tree. Now firstly we build a 1-dimensional range tree T_x based on the x-coordinates of the point set P then we build secondary structures at each internal node of the primary tree T_x to get a 2-dimensional range tree.

Some properties of 1-dimensional range tree T_x of n points that makes it efficient for range searching are:

1. Height of such a tree is $O(\log n)$ because range is halved at each internal node during construction of the range tree. A one dimensional range tree is similar to a balanced binary search tree.

2. Points rooted at any internal node of such a range tree represent a contiguous range of values. Now given a range query $[x_{lt}, x_{rt}]$ such that $x_{lt} \leq x_{rt}$, it gets divided into at most $O(\log n)$ internal nodes in the tree that cover exactly the given range. These nodes are called canonical nodes. If more than two canonical nodes are chosen at the same level of the range tree, then two of them must have a common parent and in this case we select their parent as the canonical node instead of its two children. Thus, there can be at most two canonical nodes at each of the $O(\log n)$ levels of the tree.

3. Searching either x_{lt} or x_{rt} on 1-d range tree would take $O(\log n)$ time. Therefore, it takes $O(\log n)$ to find all canonical nodes for any given orthogonal range query.

For a two dimensional range tree, we already built the primary range tree T_x whose leaves are associated with the points sorted by x-coordinate as explained above. For each internal node v, let $A(v)$ be the set of points associated with the leaves of the sub-tree rooted at v. The secondary structure associated with the each internal node v is a 1-dimensional range tree on set $A(v)$ based on sorted y-coordinates. Each level of tree T_x contains all the points of the set P exactly once. Hence the space required for a two dimensional range tree is $O(n \log n)$. The construction of a $2d$-range tree takes $O(n \log n)$ time because of sorting separately on x and y coordinates.

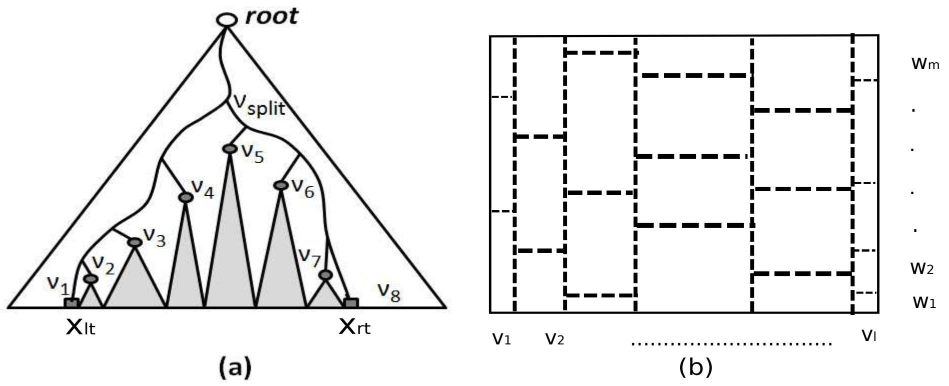

Figure 2.1 Standard Range Tree and Partition of orthogonal range query into $O(\log^2)$ disjoint regions

Given an orthogonal query $q = [x_{lt}, x_{rt}] \times [y_b, y_t]$, we can identify the $O(\log n)$ canonical nodes by considering only range $[x_{lt}, x_{rt}]$ on x-coordinate in time $O(\log n)$. For reporting the points contained in the query q, we do a binary search on secondary trees stored at each of the canonical nodes. It means we do $O(\log n)$ binary searches. Therefore total query time for reporting points inside q is $O(\log^2 n + k)$, where k is the size of the output. For counting the number of points, the total time complexity is

$O(\log^2 n)$. For higher dimensional points, we can extend the tree by recursively storing the additional structures, one for each of the d dimensions. The space required for a d-dimensional range tree is $O(n \log^{d-1} n)$ and the query time for orthogonal range reporting is $O(\log^d n + k)$. The query time can be further improved by a $O(\log n)$ factor using a technique called *fractional cascading*.

2.2 Convex hull and Graham's scan Method

Finding the convex hull of point set is one of the most fundamental problem in literature of computational geometry. Convex hull is the closed convex polygon containing the whole point set P. It contains only the *extreme points* (vertices), that is points on the boundary of the convex hull and it can be represented by counter-clockwise enumeration of its vertices. Let P be a set of n points on the plane. Based on different geometrical properties of the convex hull, many different algorithms have been proposed to compute the convex hull. For example a brute-force algorithm that takes $O(n^3)$ time, Quick hull algorithm in [5] similar to Quick sort takes $O(n^2)$ time, Gift-Wrapping Algorithm (Jarvis's March) in [13, 28] similar to selection sort takes $O(nh)$ time where $h \leq n$ is the number of points on the convex hull, Graham's scan in [22] that takes $O(n \log n)$ time, Divide-and-Conquer strategy in [4] similar to merge sort takes $O(n \log n)$ time and Chan's Algorithm in [10, 31] that takes $O(n \log h)$ time where $h \leq n$ is the number of points on the convex hull.

Graham's scan is the method for finding the convex hull for a given set of points in $O(n \log n)$ time. To appreciate the incremental construction approach of the Graham's scan algorithm first we look at the brute force method of constructing the convex hull. Each ordered pair of points (p, q) is thought to be a half-plane. We check whether the remaining $n - 2$ points other than p and q lie towards the right of the directed line from p to q. If that is the case for a given pair, we know that points p and q are on the convex hull otherwise pair (p, q) is discarded. This condition is checked for all possible ordered pairs. For a given set of n points there exist $O(n^2)$ ordered pairs and for each pair we check the above condition which takes $O(n)$ time. Therefore the total time taken to report points on the convex hull is $O(n^3 + h)$ where $h \leq n$ is the total number of points on the convex hull.

Graham's Scan is a method of computing the convex hull of a finite set of points. It is a classic example of the incremental construction approach for designing an algorithm. Consider a point set P on a plane as input to Graham's Scan method. In step 1 of the above algorithm we sort point set P into a sorted sequence $\langle p_1, p_2, \ldots, p_n \rangle$ based on x-coordinates and it takes $O(n \log n)$ time. In step 2 pushing point p_1 and p_2 into empty stack U takes $O(1)$ time. In step 3 we run a *for* loop $n - 2$ times to make a scan on the whole point set. In each scan we check the size of the stack U and orientation of the points p_i, $first(U)$ and $second(U)$ where p_i is the point in i^{th} scan, $first(U)$ is the topmost element of stack U and $second(U)$ is the second element of the stack U. For a stack U of $size(U) \geq 2$ and negative orientation of the points we pop a element from the stack U and continue the check. If the condition

Algorithm 2: *Graham's Scan*

```
Data: Point set $P$ of size $n$
Result: Counterclockwise enumeration of points on the convex hull
1 Sort the point set $P$ into increasing order of $x$-coordinates, denoted by $< p_1, p_2, \ldots, p_n >$.;
2 Push $p_1$ and then $p_2$ unto stack $U$;
3 for $i = 3$ to $n$ do
4     while $size(U) \geq 2$ and $orient(p_i, first(U), second(U)) \leq 0$ do
5         pop($U$).;
6     end
7     push $p_i$ onto $U$ ;
8 end
```

in the while loop fails then we break the loop. In step 7 we push p_i onto the stack U. Careful analysis of the steps 3-8 will show that each point of the sequence $\langle p_1, p_2, \ldots, p_n \rangle$ gets pushed onto stack U exactly once. If a element gets popped out from the stack U, its never gets pushed into it again. So the total number of push and pop operations on stack U are at most $O(n)$. Therefore time taken in steps 3-8 is $O(n)$. Hence the time complexity of Graham's Scan depends upon the sorting step which takes $O(n \log n)$ time.

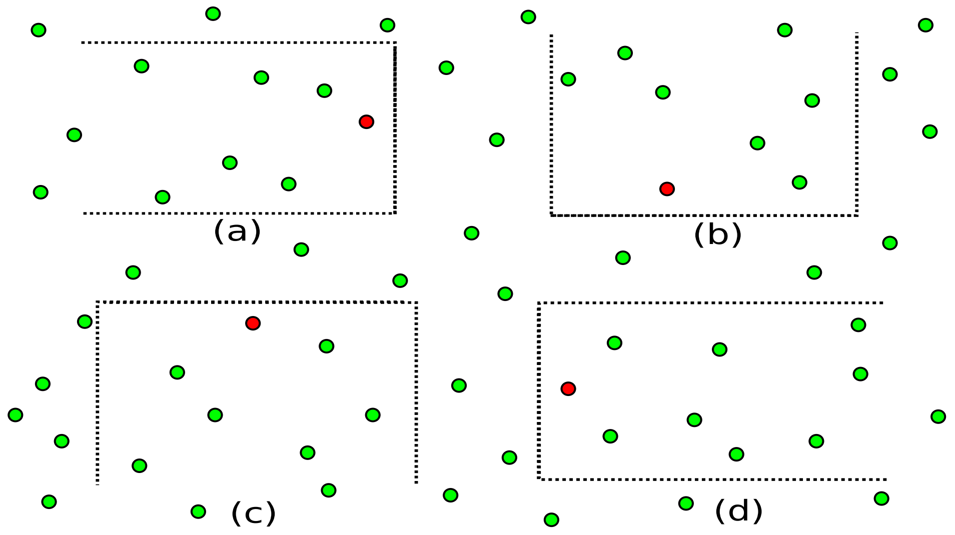

Figure 2.2 Four different types of range successor queries possible in a plane (a)Point with the largest x co-ordinate in $S \cap q$ (b)Point with the smallest y co-ordinate in $S \cap q$ (c)Point with the largest y co-ordinate in $S \cap q$ (d)Point with the smallest x co-ordinate in $S \cap q$

2.3 Range successor queries and related data structures

In this section we discuss the data structure explained in Sections 2 and 3 of Nekrich et. al.'s paper [33]. We use this data structure in Chapter 3 to derive a sub-logarithmic solution for the *planar range maxima* problem.

Let S be set of n points as shown in Figure 2.2. The answer to an orthogonal *range successor query* $q = [a, +\infty] \times [c, d]$ is the point with the smallest x-coordinate (darkened point) in $S \cap q$, as shown in Figure 2.2(d). Previously a lot of work has been done on range successor queries [12, 41]. The range tree is the most widely used data structure for various types of orthogonal range reporting problems. Its leaves contain the x-coordinates of points. Each internal node v is associated with an augmented set $A(v)$ that contains all points whose x-coordinates are stored in the subtree rooted at v. Points in set $A(v)$ are sorted by their y-coordinates. We will denote the i^{th} point in $A(v)$ by $A(v)[i]$ and $A(v)$ will denote the sorted list of points $A(v)[i], A(v)[i+1], \ldots, A(v)[j]$. We have seen earlier in section 2.1 that 2-dimensional range trees take $O(n \log n)$ space but this can be reduced to $O(n)$ space by storing compact representations of sets $A(v)$.

Now we define two operations $noderange(c, d, v)$ and $point(v, i)$ to support our data structure. We also do analysis on their space and time complexities. Given a range $[c, d]$ and a node v, $noderange(c, d, v)$ finds the range $[c_v, d_v]$ such that $p(y) \in [c, d]$ if and only if $p \in A(v)[c_v, d_v]$ for any $p \in A(v)$. Given an index i and a node v, $point(v, i)$ returns the coordinates of point $A(v)[i]$.

Lemma 1. *[12, 14, 33] There exists a compact range tree that uses $O(nf(n))$ space and supports operations $point(v, i)$ and $noderange(c, d, v)$ in $O(g(n))$ and $O(g(n) + \log\log n)$ time respectively for,*

1. *$f(n) = O(1)$ and $g(n) = O(\log^{\epsilon} n)$ from [14]*
2. *$f(n) = O(\log\log n)$ and $g(n) = O(\log\log n)$ from [12]*
3. *$f(n) = O(\log^{\epsilon} n)$ and $g(n) = O(1)$ from [14]*

Lemma 2. *[33] There exists a data structure that uses $O(n)$ space and answers orthogonal range successor queries in $O(log^{\epsilon} n)$ time.*

We use the above lemmas directly in the next chapter while providing solutions for the problems.

2.4 Summary of the Chapter

In this chapter we tried out best to cover all the fundamental concepts that will be needed to understand Chapters 3, 4 and 5. The *standard range tree* explained in Section 2.1 is the basic building block of the data structures proposed in Chapters 4 and 5. The Convex hull and Graham's scan concepts explained in Section 2.2 are needed to understand the differences between our method in Section 4.2.2 of Chapter 4 and Brass et. al.'s technique [8]. A good understanding of *range successor queries* and related data structures is needed to appreciate the data structures and techniques proposed in Chapter 3.

Chapter 3

Range Maxima with sublogarithmic query time and linear space

3.1 Introduction

In this chapter we describe a method for efficient reporting and counting of skyline points for a given orthogonal query on a set of points in two dimensions. We consider 2-sided quadrant queries and 3-sided range maxima queries.

Let S be a set of n points in the two dimensional plane. A point $p = (p_x, p_y)$ is said to *dominate* another point $q = (q_x, q_y)$ if $p_x > q_x$ and $p_y > q_y$. Points *not* dominated by any point in the set are called *maximal* or *skyline points*. In this paper, we study the following problem. Given a set S of n points in a plane and an orthogonal query Q of the form $[a, b] \times [c, d]$ where $[a, b]$ is a range on the x-axis and $[c, d]$ is a range on the y-axis, report the skyline points in $S \cap Q$. We specifically consider the case where the query is of the form $[a, \infty] \times [c, \infty]$, called a 2-sided quadrant query, and of the form $[a, \infty] \times [c, d]$, called a 3-sided query. Figure 3.1 shows an example of each.

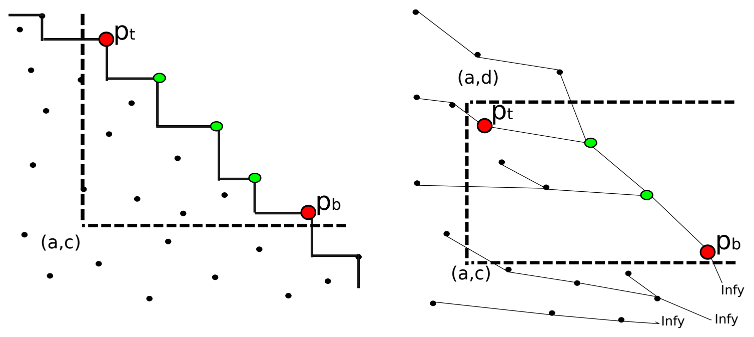

Figure 3.1 (*a*) 2-sided query unbounded on top and right (*b*) 3-sided query unbounded on the right

Reporting and counting the points in an orthogonal range has been a problem of interest in several previous works [29, 16, 15, 9]. In [29], the authors proposed a static data structure of size $O(n \log n)$

that supports reporting maximal points in a given orthogonal query rectangle in $O(\log n + k)$ time, where k is the size of the output. Brodal et al. [9], designed a dynamic data structure of size $O(n \log n)$ and worst case query time of $O(\log^2 n + k)$. In [15], Das et al. provided the first sub-logarithmic bounds: $O(n\frac{\log^3 n}{\log \log n})$. Nekrich and Navarro [33] describe a linear space data structure capable of answering orthogonal range successor queries in $O(log^\epsilon n)$ time where $0 < \epsilon < 1$. Henceforth, we will refer to the data structure proposed in [33] as $\mathcal{D}$ and it also explained briefly in section 2.3 of Chapter 2.

Using the range successor data structure $\mathcal{D}$, we show an $O(n)$ space and $O(\log^\epsilon n + k)$ query time algorithm for reporting the points in a 2-sided query region. The corresponding counting problem can be solved using $O(n)$ space and $O(\log^\epsilon n)$ query time. Our ideas are then extended to the case of a 3-sided query with the same bounds on the space and query time.

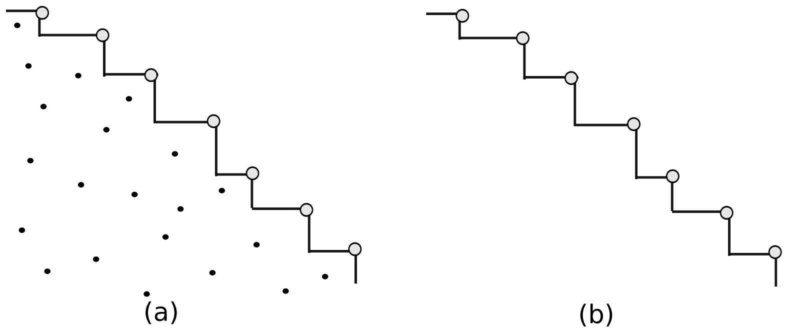

Figure 3.2 (a)Single Shot Skyline for given set of points(b)Skyline is stored in Array A_M and construct data structure on $\mathcal{D}$ on A_M

3.2 2-sided Range Reporting and Counting

Figure 3.1(a) shows a 2-sided query unbounded towards the top and right. During preprocessing, we first compute the skyline or maximal points on the whole set S using standard sweep line approach as shown in the Figure3.2 (a). Let us call this set M and store its points in an array A_M in non-decreasing order of the x-coordinate values of the points in M as shown in Figure 3.2 (b). We now build the data structure $\mathcal{D}$ for the points in M using [33]. Specifically, $\mathcal{D}$ can answer range successor queries and uses $O(n)$ space. With each point stored in $\mathcal{D}$, we also store its index from A_M.

Given a query Q of the form $[a, \infty] \times [c, \infty]$ we proceed as shown in the Figure 3.3(b). Clearly, the maximal points of the points in $S \cap Q$ is a subset of M. If we can identify the maximal point p_t in $S \cap Q$ with the largest y-coordinate, then we can traverse the remaining maximal points in $S \cap Q$ efficiently. To end this, we perform a range successor query unbounded towards bottom on $\mathcal{D}$ as shown in the Figure 3.3(a). This query allows us to find p_t and its index i of A_M in $O(\log^\epsilon n)$ time. The rest of the maximal points in $S \cap Q$ can be reported easily by traversing A_M as shown in Figure 3.3(b). Therefore, we have the following theorem.

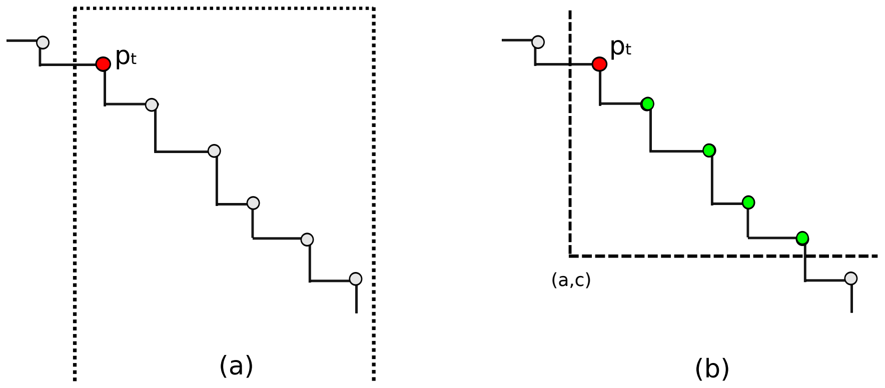

Figure 3.3 (a)Range Successor query to find point p_t on $\mathcal{D}$ (b)2-sided Range Reporting query need only point p_t for reporting

Theorem 1. *Given a set S of n points in $\mathbb{R}^2$, we can pre-process S into a data structure of size $O(n)$ in time $O(n \log n)$ such that, given an 2-sided query $Q = [a, \infty] \times [c, \infty]$, we can report the maximal points of $S \cap Q$ in time $O(\log^{\epsilon} n + k)$, where k is the number of points reported.*

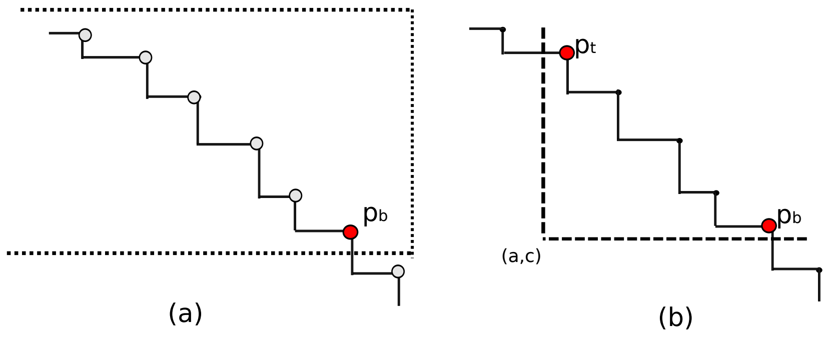

Figure 3.4 (a)Range Successor query to find point p_b on $\mathcal{D}$ (b)2-sided Range counting query needs both points p_t and p_b to find the count

To solve the corresponding counting problem, we also need to identify the maximal point p_b with the largest x-coordinate in query region $S \cap Q$ and its index j in A_M as shown in the Figure 3.4(b). To this end, we make another range successor query unbounded on left on $\mathcal{D}$ as shown in the Figure 3.4(a). Using the indices in A_M of the points p_t and p_b, the number of maximal points in $S \cap Q$ is then easily obtained as $j - i + 1$. Therefore, we have the following theorem.

Theorem 2. *Given a set S of n points in $\mathbb{R}^2$, we can pre-process S into a data structure of size $O(n)$ in time $O(n \log n)$ such that, given an 2-sided query $Q = [a, \infty] \times [c, \infty]$, we can count the maximal points of $S \cap Q$ in time $O(\log^{\epsilon} n)$.*

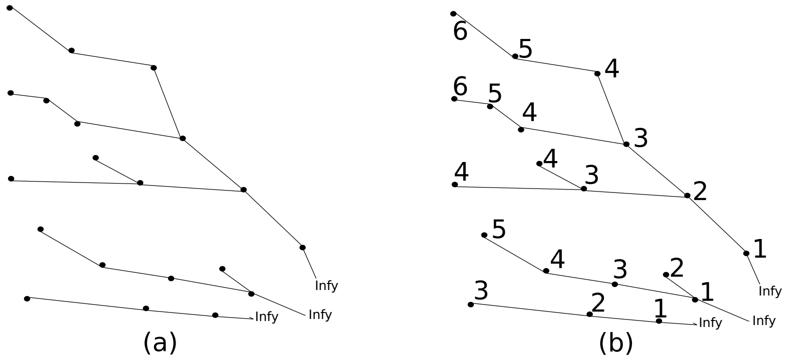

Figure 3.5 (*a*)Preprocessing for a given set of points for reporting queries (*b*)Preprocessing for a given set of points for counting queries

3.3 3-sided Range Reporting and Counting

We now extend our ideas to the case of a 3-sided query. We deal with 3-sided queries of the kind $[a, \infty] \times [c, d]$. During pre-processing, we build and populate the data structure $\mathcal{D}$ (see [33]) for the point set S. In addition, for each point $p \in S$, we store the point $p' \in S$ with the largest y-coordinate in the south-east quadrant of p. This is illustrated in Figure 3.5(a). Let us call the point p' as $nxt(p)$.

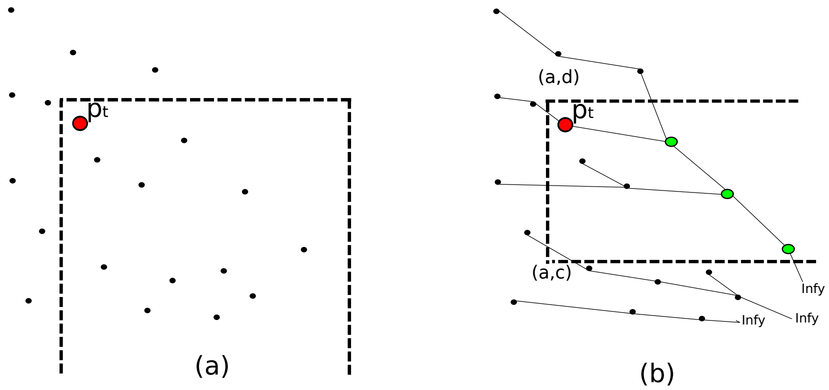

Figure 3.6 (*a*)Range Successor query to find point p_t on $\mathcal{D}$ (*b*)Range Successor query to find point p_b on $\mathcal{D}$

A 3-sided query Q to report the maximal points in $S \cap Q$ is done as shown in Figure 3.6(b). We locate the maximal point p_t in $S \cap Q$ with the largest y-coordinate by performing a range successor query unbounded on bottom on $\mathcal{D}$ as shown in the Figure 3.6(a). The remaining maximal points in $S \cap Q$ can then be identified using the array nxt as shown in the Figure 3.6(b). Therefore, we have the following theorem.

Theorem 3. *Given a set S of n points in $\mathbb{R}^2$, we can pre-process S into a data structure of size $O(n)$ in time $O(n \log n)$ such that, given an 3-sided query $Q = [a, \infty] \times [c, d]$, we can report the maximal points of $S \cap Q$ in time $O(\log^{\epsilon} n + k)$, where k is the number of points reported.*

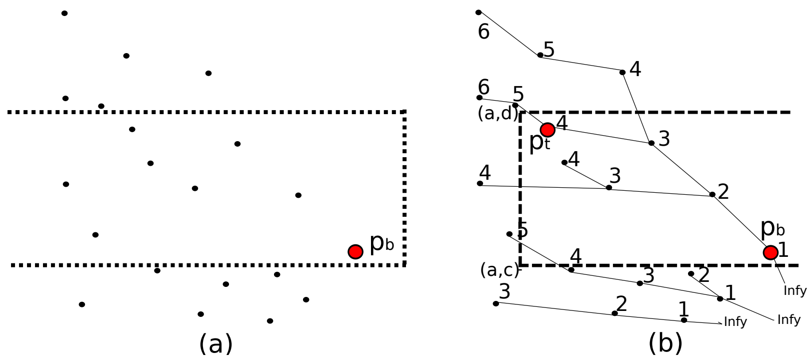

Figure 3.7 (a)2-sided Range Reporting query need only point p_t (b)3-sided Range counting query needs both point p_t and p_b

By additionally storing the count of number of points hanging from each point[29] as shown in the Figure 3.5(b). We store this count information in array $count$. We count the number of maximal points in $S \cap Q$ by making a range successor query to find point p_b as shown in Figure 3.7(a). The overall count in $S \cap Q$ is given by $count[p_t] - count[p_b] + 1$ which takes $O(1)$ time. The size of array $count$ is $O(n)$ and query time is dominated by range successor query which takes $O(\log^\epsilon n)$ time. Therefore, overall space used for the data structure remains at $O(n)$ and overall query time remains $O(\log^\epsilon n)$.

Theorem 4. *Given a set S of n points in $\mathbb{R}^2$, we can pre-process S into a data structure of size $O(n)$ in time $O(n \log n)$ such that, given an 3-sided query $Q = [a, \infty] \times [c, d]$, we can count the number of maximal points of $S \cap Q$ in time $O(\log^\epsilon n)$.*

In this work, we studied the problem of reporting and counting maximal points in a given 2-sided and 3-sided range query in the *word-RAM* model. It will be interesting to know whether our techniques can be extended to 4-sided query. So we conjecture that 4-sided range maximal reporting and counting problem done with similar bounds.

Conjecture 1. *Given a set S of n points in $\mathbb{R}^2$, it may be possible to pre-process S into a data structure of size $O(n)$ in time $O(n \log n)$ such that, given an 4-sided orthogonal query $Q = [a, b] \times [c, d]$, we can report and count the number of maximal points of $S \cap Q$ in time $O(\log^\epsilon n + k)$ and $O(\log^\epsilon n + k)$ respectively.*

3.4 Summary of the Chapter

In this chapter we discussed the *Planar Range Skyline* problem for 2-sided quadrant range maxima queries and 3-sided range maxima queries in the *word-RAM* model. We proposed a linear space data structure $\mathcal{D}$ to answer any 2-sided quadrant range maxima query, taking $O(\log^\epsilon n + k)$ and $O(\log^\epsilon n)$

time for the reporting and counting problems respectively. The proposed data structure is similar to earlier data structures proposed in [12, 14, 33, 29]. In Section 3.3 we extended our main idea to 3-sided range queries using the already proposed data structure $\mathcal{D}$. Thus we established that the reporting and counting versions of 3-sided range queries can be answered in $O(\log^{\epsilon} n + k)$ and $O(\log^{\epsilon} n)$ time respectively. We concluded with a conjecture for the 4-sided orthogonal range maxima problem.

Chapter 4

Reporting Convex hull in an Orthogonal Range

In this chapter we consider the problem of reporting convex hull points in an orthogonal range query in two dimensions. Formally, let P be a set of n points in $\mathbb{R}^2$. A point lies on the convex hull of a point set S if it lies on the boundary of the minimum convex polygon formed by S. In this thesis, we are interested in finding the points that lie on the boundary of the convex hull of the points in P that also fall with in an orthogonal range $[x_{lt}, x_{rt}] \times [y_b, y_t]$. We propose a $O(n \log^2 n)$ space data structure that can support reporting points on a convex hull inside an orthogonal range query, in time $O(\log^3 n + h)$. Here h is the size of the output. In this chapter we improve the existing results of (Brass et al. 2013) [8] that builds a data structure that uses $O(n \log^2 n)$ space and has a $O(\log^5 n + h)$ query time.

4.1 Introduction

Range searching is one of the most commonly studied topics in spatial databases and computational geometry. Informally, range searching can be stated as follows: Given a point set P we wish to pre-process P into a data structure such that given any rectangular query region q, we can efficiently report the points in $P \cap q$ or count their number in $P \cap q$. Range aggregate geometric querying is a variant of range searching where we calculate a geometrical function f inside $P \cap q$. In this chapter we efficiently compute $f(P \cap q)$ where the function f is to compute the convex hull for the point set $P \cap q$. Range aggregate querying has been studied recently by Kalavagatttu et al.[29], Das et al. [16] and Brodal et al. [9] for maximal points.

In this chapter, we use the concept of convex hull query for reporting the boundary points. Reporting of convex points can be useful in situations where we need a shape with minimum area/perimeter for a set of points. Therefore, it may have applications in spatial databases [39], computer graphics, and computer vision [36]. In this chapter, we propose a data structure to solve the problem of reporting

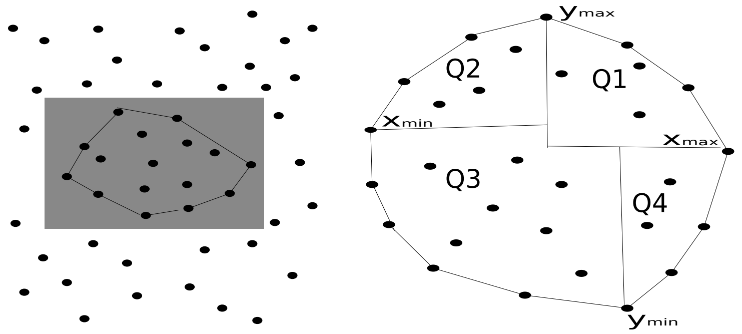

Figure 4.1 (a) Convex hull inside an orthogonal range query (the shaded region is the query). (b) Convex hull with four monotone chains

convex hull points in an orthogonal query rectangle. We also study the problem of counting the number of points on the convex hull inside $P \cap q$, and computing its area/perimeter.

4.1.1 Problem Definitions

In this section, we formally define the problems and related terminology. Given a set of n points $P = \{p_1, \ldots, p_n\}$ in $\mathbb{R}^2$. We use $x(p)$ to represent the x co-ordinate of a point p and $y(p)$ to represent its y co-ordinate. The convex hull of any point set S in the Euclidean plane is the smallest convex polygon that contains S. Henceforth, when we refer to convex hull points, we essentially mean the points on the boundary of the convex hull. We are given a set P of n points in $\mathbb{R}^2$ and a query $q = [x_{lt}, x_{rt}] \times [y_b, y_t]$. We find the convex hull on the point set $P \cap q$, denoted by $ch(P \cap q)$. We also denote points with minimum x co-ordinate, maximum x co-ordinate, minimum y co-ordinate and maximum y co-ordinate of point set $P \cap q$ by x_{min}, x_{max}, y_{min} and y_{max} respectively.

4.1.2 Previous Work

The convex hull for a static data set of two-dimensional points can be computed in $O(n \log h)$ time [10] where h is the number of points on the convex boundary. Gupta et al. discusses non-intersecting queries on aggregated geometric data [25]. Brass et al. gave the first solution for *Problem 1* that takes $O(n \log^2 n)$ space, $O(n \log^3 n)$ preprocessing time and $O(\log^5 n + h)$ query time [8]. For any given orthogonal range query on a standard 2d-range tree they identify $O(\log^2 n)$ disjoint canonical convex hulls. There are $O(\log^4 n)$ pairs of disjoint convex hulls and they compute the tangent for each pair of disjoint convex hulls. Computing tangents between each pair of local convex hulls takes $O(\log n)$ time [32]. They use a method similar to the gift-wrapping algorithm. Therefore, total query time is $O(\log^5 n + h)$. The rest of the chapter is organized as follows. In Section 4.2, we give the details of the preprocessing and the query algorithm for reporting convex points inside an orthogonal range query.

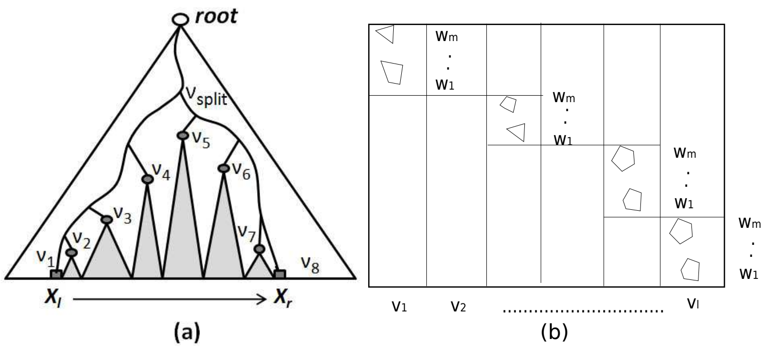

Figure 4.2 $(a.)$Range tree with the canonical nodes highlighted $(b.)$ Processing an orthogonal range query

4.2 Our Solution

In this section we explain our solution to the problem of reporting the convex hull points inside $P \cap q$. In Section 4.2.1 we describe how to construct the required data structure of size $O(n \log^2 n)$ in time $O(n \log^3 n)$ on a static point set P. In Section 4.2.2 we explain how to report points on the convex hull inside $P \cap q$ for a monotonic chain from x_{max} to y_{max}.

Points on the convex hull can be broken into four monotone chains, namely maximal chain from x_{max} to y_{max}, maxY-minX chain from y_{max} to x_{min}, minimal chain from x_{min} to y_{min} and minY-maxX chain from y_{min} to x_{max} as shown in Figure 4.1(b). Area of such a convex hull gets divided into four quadrants Q1, Q2, Q3 and Q4 as shown in Figure 4.1. In this chapter we present an algorithm to report subset of the maximal chain from x_{max} to y_{max} that lie on the convex hull. A similar approach can be applied for the other monotone chains.

4.2.1 Preprocessing

Our solution uses a $2d$-range tree as described in [20]. Each internal node of the primary tree T_x represents a horizontal range $[x_i, x_j]$ for $i \neq j$. The set of points rooted at an internal node v are represented by $S(v)$. For each internal node v of T_x we have a secondary(binary) tree $T_y(v)$ such that leaves of tree $T_y(v)$ store the points in $S(v)$ in non-decreasing order of their y-coordinate. Each internal node of the secondary tree $T_y(v)$ represents a vertical range $[y_i, y_j]$ for $i \neq j$. Given a query $q = [x_{lt}, x_{rt}] \times [y_b, y_t]$, we search x_{lt} and x_{rt} in the primary tree to get canonical nodes $v_1, v_2, \ldots, v_i, \ldots, v_l$. as shown in Figure 4.2. Here $l = O(\log n)$ is the height of the primary tree. Therefore, the range query $[x_{lt}, x_{rt}]$ is divided into $l = O(\log n)$ canonical subsets $S(v_1), S(v_2), \ldots, S(v_l)$. Similarly we search for y_b and y_t in the secondary tree $T_y(v_i)$ for $i = 1, 2, 3, \ldots, l$ to get canonical nodes $w(i,1), w(i,2), \ldots, w(i,j), \ldots, w(i,m)$ Here $m = O(\log |S(v_i)|)$ is the height of the secondary tree. Therefore, in every secondary tree, the range query $[y_b, y_t]$ gets divided into $m = O(\log n)$ canonical subsets $S(w(i,1))$, $S(w(i,2))$,..., $S(w(i,m))$. For each internal node w of the secondary tree, we compute the convex hull over the set $S(w)$. We store points of this convex hull in an array ch_w in the anti-clockwise direction starting from the point with the maximum y-coordinate. We also find the point $pmax_{w(i,j)}$ with maximum x co-ordinate from the set $S(w(i,j))$ and store it at each internal node w. Therefore, any given query $q = [x_{lt}, x_{rt}] \times [y_b, y_t]$ gets divided into $O(\log^2 n)$ disjoint canonical subsets and for each of these subsets we have stored a convex hull $ch_{w(i,j)}$ on the set $S(w(i,j))$.

It takes $O(|S(w)| \log |S(w)|)$ time to compute convex hull ch_w over set $S(w)$ and $O(|S(w)|)$ space to store ch_w at internal node w of secondary tree $T_y(v)$. Therefore it takes $\sum_{\forall w \in T_y(v)} O(|S(w)| \log |S(w)|) = O(|T_y| \log^2 |T_y|)$ preprocessing time and $\sum_{\forall w \in T_y(v)} O(|S(w)|) = O(|T_y| \log |T_y|)$ space for secondary tree $T_y(v)$. For any given query $[x_{lt}, x_{rt}]$ on the primary tree T_x we have $l = O(\log n)$ secondary trees. Therefore it takes $\sum_{\forall v \in T_x} O(|T_y(v)| \log^2 |T_y(v)|) = O(n \log^3 n)$ preprocessing time and storage space of $\sum_{\forall v \in T_x} O(|T_y(v)| \log |T_y(v)|) = O(n \log^2 n)$.

Lemma 3. *Given a set P of n points in $\mathbb{R}^2$, we can pre-process P into a data structure that takes $O(n \log^2 n)$ storage space and $O(n \log^3 n)$ pre-processing time (as explained above).*

4.2.2 Query Algorithm

In this algorithm we use two stacks, a hull stack HS and a tangent stack TS. An element of hull stack HS is a pointer to some canonical convex hull $ch(w(i,j))$. An element of tangent stack TS is a tuple $t = (i_1, i_2)$ where i_1 and i_2 are indices of points on two different convex hulls C_1 and C_2 as shown in Figure 4.3. Given an orthogonal range query $q = [x_{lt}, x_{rt}] \times [y_b, y_t]$, the query algorithm for reporting convex hull points in $P \cap q$ is as follows:

1. Express the range of x co-ordinates in $[x_{lt}, x_{rt}]$ as the disjoint union of l = $O(\log n)$ canonical subsets. Let the canonical nodes be $v_1, v_2, \ldots, v_l$ from left to right in that order, as shown in Figure 4.2(a).

2. Consider a node v_l of the primary tree T_x. Make a range query $[y_b, y_t]$ on the associated secondary tree $T_y(v_l)$ to find canonical nodes $w(i,1), w(i,2)$, ..., $w(i,m)$. Do a linear search on $x(pmax_{w(i,1)})$, $x(pmax_{w(i,2)})$, ..., $x(pmax_{w(i,m)})$ to find point X_{maxw} with maximum x co-ordinate. Then traverse the canonical nodes from right to left, starting from v_l back to v_1, as shown in Figure 4.2(a). Initialize $i \leftarrow l$ and $y_{low} \leftarrow y(X_{maxw})$.

3. Consider a node v_i of the primary tree T_x. For each internal node v_i of T_x there is an associated secondary tree $T_y(v_i)$. Make a range query $[y_{low}, y_t]$ on secondary tree $T_y(v_i)$ to find canonical nodes $w(i,1), w(i,2), \ldots, w(i,m)$, as shown in the Figure 4.2(b) where $m = O(\log |T_y(v_i)|)$.

4. Consider the array $ch_{w(i,1)}$. If the array $ch_{w(i,1)}$ is empty then go to step 5, otherwise set $y_{low} \leftarrow y(ch_{w(i,m)}[1])$ and then traverse the canonical nodes from bottom to top, starting from $w(i,1)$ back to $w(i,m)$ as shown in Figure 4.2(b) as follows:

   for $j \leftarrow 1$ to m call Algorithm Merge($ch_{w(i,j)}$,HS,TS) (see Section 2.2.1).

5. At this point, we have processed the nodes $v_l, v_{l-1}, \ldots, v_i$. Set $i \leftarrow i-1$ and if $i \geq 1$, move to the node v_i and goto step 3, else exit.

6. Call Algorithm Report(HS,TS)(see Section 2.2.2) to report the points on the convex monotone from maximum x to maximum y inside $P \cap q$.

4.2.2.1 Merge Algorithm

In this section we explain the Algorithm *Merge()* used for merging canonical convex hulls for any given query. Note that the algorithm proposed is similar to the Graham's scan algorithm [22] where we combine points which are sorted on x co-ordinate. Instead of points we have disjoint convex hulls sorted on non-increasing x co-ordinate in the stack HS. In line 1 of the algorithm, a while loop checks whether

Algorithm 3: *Merge()*

```
   Data: $ch_{w(i,j)}$, $HS = \phi$, $TS = \phi$
   Result: Updated Stacks $HS$ and $TS$
1  while $size(HS) \geq 2$ do
2      $C_1 \leftarrow secondtop(HS)$;$C_2 \leftarrow top(HS)$;$C_3 \leftarrow ch_{w(i,j)}$;
3      find points $P_a$ and $P_b$ on tangent $T_1$=($i_1$,$i_2$) joining hulls $C_1$ and $C_2$;
4      find points $P_c$ and $P_d$ on tangent $T_2$=($i_3$,$i_4$) joining hulls $C_2$ and $C_3$;
5      if $orient(p_a, p_b, p_d) \leq 0$ then
           /*(refer figure 4.3($b$))                                   */
6          pop($HS$); pop($TS$);
7      else
           /*(refer figure 4.3($a$))                                   */
8          push($T_2$); break;
9      end
10 end
11 push $ch_{w(i,j)}$ onto stack $HS$;
```

hull stack HS has sufficient elements to continue. In line 2 we store hulls at $secondtop(HS)$, $top(HS)$ and $ch(w(i,j))$ in variables C_1, C_2 and C_3. In line 3 we find points $p_a \leftarrow C_1[i_1]$ and $p_b \leftarrow C_2[i_2]$ incident with tangent T_1=(i_1,i_2) between hulls C_1 and C_2 as shown in the Figure 4.3. Computing such a tangent takes $O(\log(n_1 + n_2))$ points where $n_1 = |C_1|$ and $n_2 = |C_2|$ are sizes of the convex hulls.

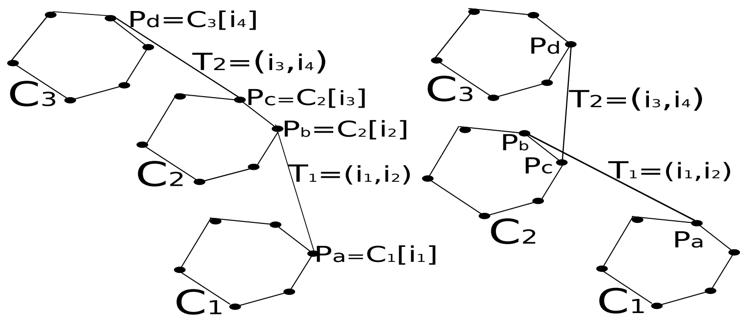

Figure 4.3 (a)Positive Orientation (b) Negative Orientation

Similarly in line 4 we find points $p_c \leftarrow C_2[i_3]$ and $p_d \leftarrow C_3[i_4]$ incident with tangent T_2=(i_3,i_4) between hulls C_2 and C_3 as shown in the Figure 4.3. In line 5 we compute the orientation of the points p_a, p_b and p_d using function $orient()$(a primitive opeation in computational Geometry). If the orientation of the points is clockwise (negative) as shown in Figure 4.3(b) then we pop out an element from the stack HS and the stack TS in line 6, else we push tangent T_2 into stack TS and break out of the loop in line 8. Every time an element gets pushed onto the stack we iterate the above procedure on the top two elements of stack HS and ch_w to discard canonical set of convex hulls $ch(w(i,j))$ that do not contribute points to $ch(P \cap q)$. For every call to Algorithm *Merge()* there can be zero or more pop() operations on stack HS. An element once popped out of the stack HS never gets pushed in again. Therefore, the total number of push and pop operations is $O(|HS|)$. For each pop and push operation on the stack HS, the required tangent can be computed in $O(\log n)$ time. Finding the tangent between any two canonical convex hulls takes $O(\log(n_1 + n_2)) = O(\log n)$ time [32]. Therefore, the merging algorithm takes no more than $O(|HS| \cdot \log n)$ time where $|HS|$ is size of the stack. We end this section with the following lemma.

Lemma 4. *Given the stack HS, the stack TS and the hull $ch_{w(i,j)}$ as input to* **Algorithm Merge()***, it takes $O(|HS| \cdot \log n)$ time where $|HS|$ is size of the stack HS.*

4.2.2.2 Reporting algorithm

In this section we explain the algorithm *Report()* used for reporting points on the convex hull. This reporting algorithm is the last step of the query algorithm. We report points from canonical convex hulls stored in stack HS using information about tangents stored in the stack TS. In step 1 of the Algorithm 4 we pop out a hull from the stack HS and index i is assigned with the size of array ch. In line 2 we report the point $ch[i]$ with maximum y co-ordinate. In line 3 we check the while loop condition, if the stack is not empty then we enter the while loop. In line 4 we pop out tangent t from the stack TS and in line 5 we assign j with the first index i_2 of the tangent. In line 6 we report points

```
Algorithm 4: Report()
  Data: Stack $HS$,Stack $TS$
  Result: Convex hull points in $P \cap q$
1 $ch \leftarrow pop(HS)$; index $i \leftarrow size(ch)$;
2 Report the point $ch[i]$ with maximum $y$ co-ordinate;
3 while $size(HS) > 0$ do
4     tangent $t \leftarrow pop(TS)$;
5     index $j \leftarrow t(i_2)$;
6     Report points from $ch[i]$ to $ch[j]$;
7     $i \leftarrow t(i_1)$;
8     array $ch \leftarrow pop(HS)$;
9 end
10 Report points from ch[$i$] till $x_{max}$ in array $ch$;
```

from ch[i] to ch[j]. In line 7 we assign i with the second index i_1 of tangent t and in line 8 we pop out a canonical convex hull from the stack TS.

In the algorithm *Report()* we iterate till the stack HS is empty and in each iteration of the while loop we report a set of points. Let total number points reported by Algorithm *Report()* be h. It takes $O(|HS|)$ time for the algorithm *Report()* because while loop gets iterated $|HS|$ times. Therefore, it takes $O(|HS| + h)$ time for algorithm *Report()*. We end this section with the following lemma.

Lemma 5. *Given the stack HS and the stack TS as input to* **Algorithm Report()***, it takes $O(|HS|+h)$ time where $|HS|$ is size of the stack HS and h is the number of points reported.*

4.2.3 Putting Everything Together

We will now prove the following theorem:

Theorem 5. *Given a set P of n points in $\mathbb{R}^2$, we can pre-process P into a data structure of size $O(n \log^2 n)$ in time $O(n \log^3 n)$ such that, given an orthogonal range query $q = [x_l, x_r] \times [y_b, y_t]$, we can report the points of the convex hull inside $P \cap q$ in time $O(\log^3 n + h)$, where h is the number of convex hull points reported.*

Proof : Lemma 3 for preprocessing indicates the storage space used by our data structure and the preprocessing time to build it. Now we analyze the complexity of our query algorithm explained in Section 4.2.2. In step 1 it takes $O(\log n)$ time to find $l = O(\log n)$ canonical nodes because the height of primary tree T_x is $O(\log n)$. In step 2 it takes $O(\log |S(v_l)|)$ time to find canonical subset of nodes $w(l,1), w(l,2), \ldots, w(l,m)$. It takes $m = O(\log |S(v_l)|)$ time to perform a linear search on $x(max_{w(i,1)})$, $x(max_{w(i,2)})$, …, $x(max_{w(i,m)})$ to find point p_{xmax} with maximum x co-ordinate. In step 3 we consider each v_i from $i \leftarrow l$ to 1. In i^{th} iteration of step 3 we spend $O(\log |S(v_i)|) = O(\log n)$

time finding the canonical subset of nodes $w(i,1), w(i,2), \ldots, w(i,m)$ by making an updated query on the associated secondary tree $T_y(v_i)$. Therefore, the total time spent in finding all canonical sets of nodes for any given query is $O(\log|S(v_1)|) + O(\log|S(v_2)|) + \ldots + O(\log|S(v_l)|)$ for $i = 1, 2, \ldots, l$ then total time is $O(l \cdot \log n) = O(\log^2 n)$. In step 4 at most $m = O(\log^2 n)$ calls are made to Algorithm *Merge()*. Therefore total calls made to algorithm *Merge()* is $m \cdot l = O(\log^2 n)$. From Lemma 4 we know that time taken for Algorithm *Merge()* is $O(|HS| \cdot \log n)$ Therefore, it takes $O(\log^3 n)$ time to find updated HS that contains convex hulls that contribute at least one point to $ch(P \cap q)$. In step 6 of the query algorithm a call to the algorithm Report(HS,TS) is made. According to Lemma 5 it takes $O(|HS|+h)$ time where $|HS|$ is size of the stack HS and h is the number of points reported. Therefore, it takes $O(|HS| + h)$ time for step 6 to execute where $|HS| = O(\log^2 n)$. Recall that time taken for step $3 - 5$ is $O(\log^3 n)$. Therefore it takes $O(\log^3 n) + O(\log^2 n + h) = O(\log^3 n + h)$ to report the points of $ch(P \cap q)$ in the first quadrant Q1 (see Figure 4.1).

4.3 Summary of the chapter

In this chapter we discussed the reporting version of *planar range convex hull problem*. We proposed a data structure with $O(n \log^2 n)$ space build upon the *standard range tree* to report the convex hull points using the query algorithm based technique similar to Graham's scan for merging local convex hulls at canonical nodes. This query algorithm takes $O(\log^3 n)$ time for finding such a convex hull. Now we report points of this convex hull which takes $O(\log^3 n + h)$ time. The results proposed in this chapter are tabulated in Table 6.2. The work presented in this chapter is also available at [2].

Chapter 5

Finding count, area and perimeter of convex hull in an orthogonal range

In this chapter we show that data structure proposed in previous chapter can be modified slightly to solve other related problems. For instance, for counting the number of points on the convex hull in an orthogonal query rectangle, we propose an $O(n \log^2 n)$ space data structure that can be queried upon in $O(\log^3 n)$ time. We also propose a $O(n \log^2 n)$ space data structure that can compute the $area$ and $perimeter$ of the convex hull inside an orthogonal range query in $O(\log^3 n)$ time.

In this chapter we present solution to related problems such as counting the number of convex hull points and finding the area/perimeter of the convex hull inside $P \cap q$. One may argue that it is not necessary to study these problems separately once we have reported the points of $ch(P \cap q)$ in query time $O(\log^3 n + h)$ using the algorithm described in Section 4.2.2 where h is the total number of points on the convex hull. However, h can be as large as $O(n)$ for some queries. If these problems can be solved independent of $O(h)$ then the running time is unaffected by output size. With a few clever modifications on the data structure explained in Section 4.2.1 and the algorithm proposed in Section 4.2.2 we can achieve results that are independent of output size for the above mentioned problems.

5.1 The Counting Problem

Most of the preprocessing of the point set P for the counting problem is the same as the preprocessing described in Section 4.2.1. In addition to it, at each internal node w of every secondary tree $T_y(.)$ we store three indices $index_{xmax}$, $index_{xmin}$ and $index_{ymin}$ where $ch_w[index_{xmax}]$, $ch_w[index_{xmin}]$ and $ch_w[index_{ymin}]$ are points with maximum x co-ordinate, minimum x coordinate and minimum y co-ordinate. To get the stack HS which contains all the canonical convex hulls that contribute to convex hull $ch(P \cap q)$ we use same Query Algorithm of Section 4.2.2. Below is the algorithm for counting the number of points on the convex hull of set $S(P \cap q)$. To count the points in $ch(P \cap q)$, algorithm counting() can be used as subroutine in step 6 of the query algorithm. Algorithm *Counting()* is similar

Algorithm 5: Counting()

```
   Data: stack $HS$,stack $TS$
   Result: $count$ of the points of $ch(P \cap q)$ from maximum $y$ to maximum $x$
1  array $ch \leftarrow pop(HS)$; index $i \leftarrow size(ch)$;
2  $count \leftarrow 1$;
3  while $size(HS) > 0$ do
4      tangent $t \leftarrow pop(TS)$;
5      index $j \leftarrow t(i_2)$;
6      $count \leftarrow count + (j - i + 1)$;
7      $i \leftarrow t(i_1)$;
8      array $ch \leftarrow pop(HS)$;
9  end
10 $count \leftarrow count + (index_{xmax} - i + 1)$;
```

in spirit to the Algorithm *Report()* of Section 4 In Step 2 of this algorithm $count$ is initialized to 1 because we count all the points from point $ch[j]$ till the point with maximum y co-ordinate in $P \cap q$. In Step 6 the variable $count$ gets updated with the number of points that the current canonical hull contributes to the output. This is computed as the difference $i - j + 1$ in each iteration of while loop. This process is repeated until the stack HS is empty. In step 10 $count$ gets updated with the difference $index_{xmax} - i + 1$ after exiting from the while loop, where $index_{xmax}$ is the index of point with the maximum x co-ordinate in the array ch.

It can be seen that Algorithm *Counting()* runs in $O(|HS|) = O(\log^2 n)$ query time to count points of the convex hull $ch(P \cap q)$ from maximum x to maximum y. Similar counting algorithms can be used to find the count of points of the convex hull $ch(P \cap q)$ for the other three monotone chains (see also Figure 4.1).

Theorem 6. *Given a set P of n points in $\mathbb{R}^2$, we can pre-process P into a data structure of size $O(n \log^2 n)$ in time $O(n \log^3 n)$ such that, given an orthogonal range query $q = [x_l, x_r] \times [y_b, y_t]$, we can count the points of the convex hull inside $P \cap q$ in time $O(\log^3 n)$.*

5.2 The Perimeter Problem

For computing the perimeter of the convex hull of the points in $P \cap q$ on a two-dimensional point set P and an orthogonal range q, we perform a preprocessing phase that is similar to the one described in Section 4.2.1. In addition, at every internal node w of every secondary tree, we store an auxiliary array P_w that stores the cumulative perimeter. $P_w[i] = \sum_{1 \leq j \leq i} dist(j, j+1)$ where $dist(j, j+1)$ is the distance between points $ch_w[j]$ and $ch_w[j+1]$ respectively. Below Algorithm *Perimeter()* used for finding the perimeter of the convex hull $ch(P \cap q)$ from maximum x to maximum y.

Algorithm 6: Perimeter()

```
   Data: $HS$,$TS$
   Result: $perimeter$ of $ch(P \cap q)$ from maximum $y$ to maximum $x$
1  convex hull $ch \leftarrow pop(HS)$; index $i \leftarrow 1$;
2  $perimeter = 0$;
3  while $size(HS) > 0$ do
4      tangent $t \leftarrow pop(TS)$;
5      index $j \leftarrow t(i_2)$;
6      point $a = ch[j]$;
7      $perimeter \leftarrow perimeter + (P_w[i] - P_w[j])$;
8      $i \leftarrow t(i_1)$;
9      $ch = pop(HS)$;
10     point $b = ch[i]$;
11     $perimeter = perimeter + dist(a, b)$;
12 end
13 $perimeter \leftarrow perimeter + (P_w[i] - P_w[index_{xmax}])$;
```

Algorithm *Perimeter()* is similar in spirit to the reporting algorithm (Algorithm Report) of Section 5.1. In Step 2 of this algorithm, the variable $perimeter$ is initialized to 0. In Step 6 we store point $ch[i_1]$ of some hull C_1 into point a. In Step 7 the variable $perimeter$ is updated with difference $P_w[i] - P_w[j]$ in each iteration of while loop. This process is repeated until the stack HS is empty. In Step 13 the variable $perimeter$ is updated with difference $P_w[i] - P_w[index_{xmax}]$ after exiting from while loop where $index_{xmax}$ is index of point with the maximum x co-ordinate in array ch. All other steps are similar to counting algorithm in section 5.1

It can be noticed that the Algorithm Perimeter runs in $O(|HS|) = O(\log^3 n)$ time to find perimeter of the convex hull $ch(P \cap q)$ from maximum x to maximum y. A similar algorithm can be used to find the perimeter of the convex hull $ch(P \cap q)$ for other three monotone chains. (See also Figure 4.1.)

Theorem 7. *Given a set P of n points in $\mathbb{R}^2$, we can pre-process P into a data structure of size $O(n \log^2 n)$ in time $O(n \log^3 n)$ such that, given an orthogonal range query $q = [x_l, x_r] \times [y_b, y_t]$, we can compute the perimeter of the convex hull inside $P \cap q$ in time $O(\log^3 n)$.*

5.3 The Area Problem

During the preprocessing phase, an auxiliary array A_w is stored at each internal node of every secondary tree. Each element of such an array stores the cumulative area $A_w[i] = \sum_{3 \le j \le i} aot(1, j-1, j)$ where $aot(1, j-1, j)$ is the area of the triangle between points $ch_w[1]$, $ch_w[j-1]$ and $ch_w[j]$ respectively. $A_w[1] = 0$ because it represents the area of point $ch[1]$ and $A_w[2] = 0$ because it is the area of the line joining points $ch[1]$ and $ch[2]$. Below is an algorithm for computing the area of the convex hull $(P \cap q)$.

Algorithm 7: Area()

```
Data: Stacks $HS$,$TS$,$AR = \phi$
Result: Area of the convex hull of Q1 in figure 4.1
$area_{Q1} = 0$;
point $fixedPoint$ array $ch \leftarrow pop(HS)$; index $i \leftarrow 1$;
$push(ch[i])$ onto $AR$;
$y(fixedPoint) = y(ch[1])$;
while $size(HS) > 0$ do
    tangent $t \leftarrow pop(TS)$;
    index $j \leftarrow t(i_2)$;
    $push(ch[j])$ onto $AR$;
    $area_{Q1} \leftarrow area_{Q1} + (A_w[i] - A_w[j]) - aot(1, i, j)$;
    $i \leftarrow t(i_1)$;
    $push(ch[i])$ onto $AR$;
    ch=pop($HS$);
end
$j \leftarrow index_{xmax}$;
$push(ch[j])$ onto $AR$;
$area_{Q1} \leftarrow area_{Q1} + (A_w[i] - A_w[j]) - aot(1, i, j)$;
$x(fixedPoint) = x(ch[j])$;
point $k1 \leftarrow pop(AR)$;
while $size(AR) > 0$ do
    point $k2 \leftarrow pop(AR)$;
    $area_{Q1} \leftarrow area_{Q1} + aot(fixedPoint, k1, k2)$;
    $k1 \leftarrow k2$;
end
```

The algorithm *Area()* finds the area of quadrant Q1 (Figure 4.1). Similarly we can find areas of the other quadrants Q2,Q3 and Q4. The area of $ch(P \cap q)$ can be obtained using the following formula.
Area(ch($P \cap q$) = $area_{Q1} + area_{Q2} + area_{Q3} + area_{Q4} - (|x(y_{max}) - x(y_{min})| * |y(x_{max}) - y(x_{min})|)$
where x_{min}, x_{max}, y_{min} and y_{max} are the points with minimum x, maximum x, minimum y and maximum y in $P \cap q$.

Theorem 8. *Given a set P of n points in $\mathbb{R}^2$, we can pre-process P into a data structure of size $O(n \log^2 n)$ in time $O(n \log^3 n)$ such that, given an orthogonal range query $q = [x_l, x_r] \times [y_b, y_t]$, we can compute the area of the convex hull inside $P \cap q$ in time $O(\log^3 n)$.*

5.4 Summary of the chapter

In this chapter we discussed problems related to *Planar Range Convex hull* like finding the *count* (i.e. the number of convex hull points), the *area* and the *perimeter* of the convex hull. For all these problems, we proposed several $O(n \log^2 n)$ space data structures which take $O(n \log^3 n)$ time to answer any orthogonal range query. All these data structures are very similar except the information stored in the internal nodes of the respective secondary trees is different based on the nature of the problem being solved. For example to compute *area* we store the cumulative areas at each internal node of every secondary tree. Table 6.2 summarizes the results presented in this chapter. The work presented in this chapter is also available at [2].

Chapter 6

Conclusion

We conclude by summarizing the results presented in this thesis. We also look at future work that can be done towards improving these results. In Chapter 2 we described *standard range trees* which we use to build our solution by storing additional information at each internal node of the tree. We also studied Graham's scan algorithm which is used to find the convex hull for a finite set of points in time $O(n \log n)$ where n is size of the input point set. We also studied the compact representation of range trees and related data structures which we employ to solve problems. In Chapter 3 we proposed a data structure

Query Type	**Query Time**	**Storage Space**	**Theorems**
2-sided Range Reporting	$O(\log^{\epsilon} n + k)$	$O(n)$	Theorem 1
2-sided Range Counting	$O(\log^{\epsilon} n)$	$O(n)$	Theorem 2
3-sided Range Reporting	$O(\log^{\epsilon} n + k)$	$O(n)$	Theorem 3
3-sided Range Reporting	$O(\log^{\epsilon} n)$	$O(n)$	Theorem 4

Table 6.1 Our Results of Planar Range Skyline Problem

for the range skyline problem that takes $O(n)$ space to answer reporting queries in $O(\log^{\epsilon} n + k)$ time and counting queries in $O(\log^{\epsilon} n)$ time. It takes $O(n)$ to preprocess this data structure in the *word-RAM*. We solved the range skyline problem for 2-sided quadrant queries and 3-sided range queries unbounded on the right. In Table 6.1 we present a summary of the results corresponding to each type of problem based on either the type of the problem (*reporting or counting*) or restrictions on range queries. Note that all results summarized in Table 6.1 are in the *word-RAM* model.

Query Type	**Query Time**	**Storage Space**	**Theorems**
Reporting	$O(\log^3 n + h)$	$O(n \log^2 n)$	Theorem 5
Counting	$O(\log^3 n)$	$O(n \log^2 n)$	Theorem 6
Area	$O(\log^3 n)$	$O(n \log^2 n)$	Theorem 7
Perimeter	$O(\log^3 n)$	$O(n \log^2 n)$	Theorem 8

Table 6.2 Our Results of Planar Range Convex hull and Related problems

In Chapter 4 we proposed a data structure that takes $O(n \log^2 n)$ space to answer reporting queries in $O(\log^3 n + h)$ time. It takes $O(n \log^3 n)$ time to preprocess this data structure. In Chapter 5 we proposed several similar data structures each with little modification by storing varied information in each structure based on the nature of the problem. In Table 6.2 we list out summarized results corresponding to the type of problem based on the type of the query like reporting, counting, finding area/perimeter. All results in Table 6.2 are in the *pointer machine model*. It will be interesting to see whether the query time and storage space can be improved by a couple of logarithmic factors. Also of interest will be to study other problems like smallest enclosing disk in a range, diameter, width and other problems capturing some relevant geometrical features from the point set.

Some extensions for both the range maxima and the range convex hull problem for future work are:

- **Generalization:** There has been extensive study on the generalized intersection searching problem, where inputs are categorized [27]. Categorization can be made by assigning colors to each point. Now the problem is to report range skyline such that points reported or counted all have distinct colors which means, a dominating point from each category. Similarly it will interesting to find a range convex hull on specified set of colors.
- **Other models of Computation:** It will be worthwhile to study both *planar range maxima* and *planar range convex hull* in the *cell-probe* model to understand lower bound. An analysis of the planar range convex hull in the *word-RAM* should also be fruitful. The most interesting possibility will be to find an linear space data structure with efficient query time either in *word-RAM* model or in *cell-probe* model. Studying these problems for *I/O-efficient storage devices*, *external memory* model and cache-oblivious model will be have considerable practical significance.
- **Higher Dimensions:** Both the problems that we studied in this thesis are on a planar point set. It will be interesting to see whether output sensitive solutions can be framed for these problems in dimensions greater than two. Range Convex hull in 3-dimensions (space) will be an interesting problem to study.
- **Dynamism:** Both the problems that we studied in this thesis are on a *static* point set. In the dynamic setting new points can be inserted, existing points can be deleted or both. Recently there have been very good results on the dynamic range maxima reporting problem in a plane [9]. The counting version of this problem could also be examined. In our knowledge no work has been done on the dynamic planar range convex hull problem. It should be an utter delight for the researcher to work on the dynamic version of this problem.

Bibliography

[1] Computational geometry. In *Computational Geometry*, pages 1–17. Springer Berlin Heidelberg, 2008.

[2] J. Agarwal, N. Moidu, K. Kothapalli, and K. Srinathan. Efficient range reporting of convex hull. *CoRR*, abs/1307.5612, 2013.

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

[4] A. M. Andrew. Another efficient algorithm for convex hulls in two dimensions. *Inf. Process. Lett.*, 9(5):216–219, 1979.

[5] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM TRANSACTIONS ON MATHEMATICAL SOFTWARE*, 22(4):469–483, 1996.

[6] J. L. Bentley. Decomposable searching problems. *Inf. Process. Lett.*, 8(5):244–251, 1979.

[7] J. L. Bentley. Multidimensional divide-and-conquer. *Commun. ACM*, 23(4):214–229, 1980.

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

[9] G. S. Brodal and K. Tsakalidis. Dynamic planar range maxima queries. In *ICALP (1)*, pages 256–267, 2011.

[10] T. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, 16(4):361–368, 1996.

[11] T. Chan. A fully dynamic algorithm for planar width. In *in Proc. 17th ACM Sympos. Comput. Geom*, pages 172–176, 2002.

[12] T. M. Chan, K. G. Larsen, and M. Patrascu. Orthogonal range searching on the ram, revisited. *CoRR*, abs/1103.5510, 2011.

[13] D. R. Chand and S. S. Kapur. An algorithm for convex polytopes. *J. ACM*, 17(1):78–86, Jan. 1970.

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

[15] A. Das, P. Gupta, and K. Srinathan. Counting maximal points in a query orthogonal rectangle. In S. Ghosh and T. Tokuyama, editors, *WALCOM: Algorithms and Computation*, volume 7748 of *Lecture Notes in Computer Science*, pages 65–76. Springer Berlin Heidelberg, 2013.

[16] A. S. Das, P. Gupta, A. K. Kalavagattu, J. Agarwal, K. Srinathan, and K. Kothapalli. Range aggregate maximal points in the plane. In *WALCOM*, pages 52–63, 2012.

[17] A. S. Das, P. Gupta, and K. Srinathan. Data structures for reporting extension violations in a query range. In *CCCG*, pages 129–132, 2009.

[18] A. S. Das, P. Gupta, and K. Srinathan. Detecting vlsi layout, connectivity errors in a query window. In *CCCG*, 2011.

[19] P. Davoodi, M. Smid, and F. Walderveen. Two-dimensional range diameter queries. In D. Fernández-Baca, editor, *LATIN 2012: Theoretical Informatics*, volume 7256 of *Lecture Notes in Computer Science*, pages 219–230. Springer Berlin Heidelberg, 2012.

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

[21] H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, STOC '84, pages 135–143, New York, NY, USA, 1984. ACM.

[22] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inf. Process. Lett.*, 1(4):132–133, 1972.

[23] P. Gupta. Algorithms for range-aggregate query problems involving geometric aggregation operations. In X. Deng and D.-Z. Du, editors, *Algorithms and Computation*, volume 3827 of *Lecture Notes in Computer Science*, pages 892–901. Springer Berlin Heidelberg, 2005.

[24] P. Gupta, R. Janardan, and M. H. M. Smid. Further results on generalized intersection searching problems: Counting, reporting, and dynamization. In *WADS*, pages 361–372, 1993.

[25] P. Gupta, R. Janardan, and M. H. M. Smid. Efficient non-intersection queries on aggregated geometric data. *Int. J. Comput. Geometry Appl.*, 19(6):479–506, 2009.

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

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

[28] R. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2(1):18 – 21, 1973.

[29] A. Kalavagattu, J. Agarwal, A. Das, and K. Kothapalli. On counting range maxima points in plane. In S. Arumugam and W. Smyth, editors, *Combinatorial Algorithms*, volume 7643 of *Lecture Notes in Computer Science*, pages 263–273. Springer Berlin Heidel, 2012.

[30] A. K. Kalavagattu, A. S. Das, K. Kothapalli, and K. Srinathan. On finding skyline points for range queries in plane. In *CCCG*, 2011.

[31] D. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM Journal on Computing*, 15(1):287–299, 1986.

[32] D. G. Kirkpatrick and J. Snoeyink. Computing common tangents without a separating line. In *WADS*, pages 183–193, 1995.

[33] Y. Nekrich and G. Navarro. Sorted range reporting. In *SWAT*, pages 271–282, 2012.

[34] M. Overmars. Designing the computational geometry algorithms library cgal. In M. Lin and D. Manocha, editors, *Applied Computational Geometry Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 53–58. Springer Berlin Heidelberg, 1996.

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

[36] S. Saha, A. Das, and B. Chanda. An automatic image segmentation technique based on pseudo-convex hull. In P. Kalra and S. Peleg, editors, *Computer Vision, Graphics and Image Processing*, volume 4338 of *Lecture Notes in Computer Science*, pages 70–81. Springer Berlin Heidelberg, 2006.

[37] S. Saxena. Dominance made simple. *Inf. Process. Lett.*, 109(9):419–421, Apr. 2009.

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

[39] Y. Tao and D. Papadias. Range aggregate processing in spatial databases. *IEEE Trans. Knowl. Data Eng.*, 16(12):1555–1570, 2004.

[40] G. T. Toussaint. Basic introduction to computaitonal geometry.

[41] C.-C. Yu, W.-K. Hon, and B.-F. Wang. Improved data structures for the orthogonal range successor problem. *Computational Geometry*, 44(3):148 – 159, 2011.