Algorithms and Data Structures for Geometric Intersection Query Problems

A THESIS
SUBMITTED TO THE FACULTY OF THE DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING
OF THE UNIVERSITY OF MINNESOTA
BY

Saladi Rahul

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILISOPHY

Advisor: Prof. Ravi Janardan

September, 2017

Acknowledgements

My journey into theoretical computer science started at IIIT-Hyderabad. A special thanks to Prof. Prosenjit Gupta for introducing me to the world of computational geometry; and Prof. Rajan for being a great mentor during my stay at IIIT-H. Both of them truly cared for me and I will never forget that.

I am thankful to Prof. Ravi Janardan for taking me as a graduate student to his lab. More importantly, I would like to thank him for showing a lot of patience over the past six years. I have learnt many invaluable research skills from him. I will always remember two of his phrases "have blinders on.." and "you will *not* be judged by the number of problems you have attempted, but by the number of problems you *actually* solve". I had the habit of attempting too many problems at the same time and he had to invoke the above two phrases from time-to-time to get me focussed on finishing a particular problem.

I had a very fruitful collaboration and friendship with Yufei Tao. Our collaboration started with my month long trip to Hong Kong where my stay was generously taken care of. In my initial years, I was not sure if I was capable enough to do research in theoretical computer science, but Yufei (for some reason) believed that I could do it and always encouraged me.

Prof. Barna Saha visited Minnesota and taught a course on "Algorithmic Techniques for Big Data Analysis". This course taught me tools which made it easy for me to understand many modern concepts of computational geometry. Thanks a lot, Barna! My SODA paper would not have been possible without this course.

At Minnesota, I had fruitful interactions with Prof. John Gunnar Carlsson and Prof. Mohamed Mokbel. Thank you John for inviting me to Stanford for a week to work on our paper; and thank you Prof. Mokbel for inviting me to present my work to your

research group and to your class.

Timothy Chan and Sariel Har-Peled are two stalwarts in the field of computational geometry. It has been a great learning experience working with them over the past few months. I am really looking forward to start a postdoc with both of them at UIUC.

The University of Washington theoretical computer science group has been my second home for the past three years. They let me generously attend all their theory courses. I would like to thank the faculty (especially, Anna and Paul) for providing me a desk space to work. The theory lunches were a great place to meet everyone, have good food and listen to some nice theory. I made some good friends with the graduate students at UW (especially Siva).

In the summer of 2015, I got a chance to work as an intern at Microsoft Research, Redmond (MSR). The exposure to the research environment at MSR gave me a new perspective about conducting research. Thanks a lot to my mentors Ishai, Srikanth, Nikhil and Peter. This internship lead to friendship with Janardhan Kulkarni (a.k.a. Jana) who continues to be a mentor on the topics of research and life.

I was blessed with good friends and labmates during my stay at Minnesota and Seattle, especially the "310 Gang" (Raghu, Pavani, Anand, GV, Narayanaswamy(?) with its Wednesday nights at *Legends* and the fights over the *special ikea plates*), the "Chateau Gang" (Anuj, Ankush, and Pragya), the "Seattle Group" and the "Computational Geometry Lab" (Jie, Yuan, Yokesh, and Akash). I want to use this opportunity to say "Hii" to all my other friends in India, USA and other parts of the world. I cannot thank my wife, Haritha, enough for standing by me all this time. She had to make many sacrifices: quitting her job in India, moving to a new country, and facing financial troubles in the early days of my Ph.D. Finally, I would like to thank my family (my mom Surekha, my dad, and my brother Sidhu). Their love and affection ensured that I had a wonderful childhood.

Dedication

To my parents...I bow down to them.

Abstract

The focus of this thesis is the topic of *geometric intersection queries* (GIQ) which has been very well studied by the computational geometry community and the database community. In a GIQ problem, the user is not interested in the entire input geometric dataset, but only in a small subset of it and requests an *informative summary* of that small subset of data. Formally, the goal is to preprocess a set $\mathcal{A}$ of n geometric objects into a data structure so that given a query geometric object q, a certain *aggregation function* can be applied efficiently on the objects of $\mathcal{A} \cap q$. The classical aggregation functions studied in the literature are reporting or counting the objects of $\mathcal{A} \cap q$. In many applications the same set $\mathcal{A}$ is queried several times, in which case one would like to answer a query faster by preprocessing $\mathcal{A}$ into a data structure. The goal is to organize the data into a data structure which occupies *a small amount of space* and yet responds to any user query in *real-time*.

In this thesis the study of the GIQ problems was conducted from the point-of-view of a computational geometry researcher. Given a model of computation and a GIQ problem, what are the best possible upper bounds (resp., lower bounds) on the space and the query time that can be achieved by a data structure? Also, what is the *relative hardness* of various GIQ problems and aggregate functions. Here relative hardness means that given two GIQ problems A and B (or, two aggregate functions $f(\mathcal{A}, q)$ and $g(\mathcal{A}, q)$), which of them can be answered faster by a computer (assuming data structures for both of them occupy asymptotically the same amount of space)?

This thesis presents results which increase our understanding of the above questions. For many GIQ problems, data structures with optimal (or near-optimal) space and query time bounds have been achieved. The geometric settings studied are primarily *orthogonal range searching* where the input is points and the query is an axes-aligned rectangle, and the dual setting of *rectangle stabbing* where the input is a set of axes-aligned rectangles and the query is a point. The aggregation functions studied are primarily reporting, top-k, and approximate counting. Most of the data structures are built for the internal memory model (word-RAM or pointer machine model), but in some settings they are generic enough to be efficient in the I/O-model as well.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Designing efficient algorithms and data structures for geometric problems is an active topic of research in theoretical computer science and databases. The focus of this thesis will be one of its popular sub-topics called *geometric intersection queries* (GIQ). Real-world problems from diverse domains (such as Geographic Information System (GIS), robotics, spatial and temporal databases, and networking) can be modelled as GIQ problems. We present two motivating examples:

Finding nearby top-rated restaurants: Consider a tourist in the New York City who wants to locate the nearest restaurants on his smartphone. This can be modelled as an *orthogonal range searching* problem: Let $\mathcal{A}$ be a set of points in the plane and q be an axes-aligned rectangle (see Figure 1.1(a)). The points could represent restaurants and the query rectangle q can be an area in New York City, and the user wants to know all the points in $\mathcal{A}$ which lie inside q.

Modeling user preferences as hyper-rectangles: In the *rectangle stabbing* problem $\mathcal{A}$ is a set of axes-aligned rectangles and q is a point in the plane (see Figure 1.1(e)). Rectangle stabbing is useful in applications in which the *preferences of the users* can be modeled as hyper-rectangles in d-dimensional space. For example, consider a real-estate database that contains information on several thousand homes for sale in a large metropolitan area. A potential buyer can specify his preference as a two-dimensional rectangle: "I am looking for houses whose price is in the range \$200,000 to \$500,000, and whose age is in the range 3 to 10". Here price is the x-axis and age is the y-axis.

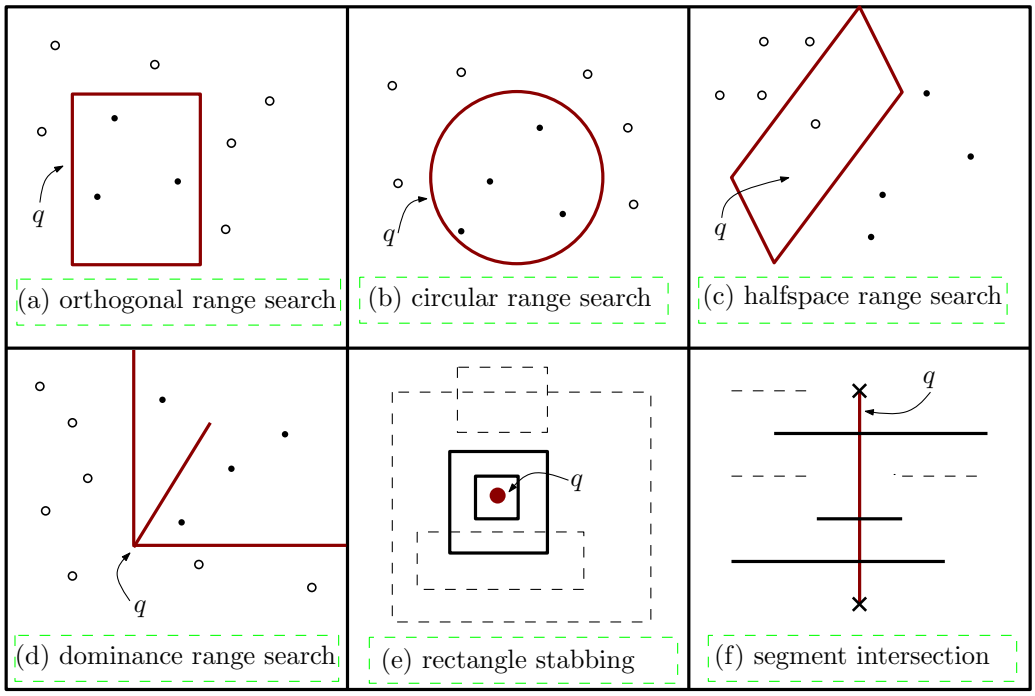


Figure 1.1: Various geometric settings in $\mathbb{R}^2$ and $\mathbb{R}^3$. (Figures in the thesis are best viewed in color.)

Each house can be modeled as a query point: (price, age). The output of the query will be the set of buyers interested in buying that house.

Notice that in the above GIQ problems, the user is not interested in the entire geometric dataset, but only in a small subset of it and requests an *informative summary* of that small subset of data. A formal definition is the following:

Geometric intersection query (GIQ). Preprocess a set $\mathcal{A}$ of n geometric objects into a data structure so that given a query geometric object q, a certain aggregation function can be applied efficiently on the objects of $\mathcal{A} \cap q$.

If we are interested in answering a single query, it can be done in linear time, by

simply checking for each object $p \in \mathcal{A}$ whether p lies in the query range q and then computing the aggregation function on the objects of $\mathcal{A} \cap q$. However, in many applications (such as the ones discussed above) the same set $\mathcal{A}$ is queried several times, in which case we would like to answer a query faster by preprocessing $\mathcal{A}$ into a data structure. The rationale here is that the cost of preprocessing will be more than compensated for by the savings in response time when answering hundreds of thousands of queries (as opposed to using a naïve query algorithm on un-preprocessed data).

The performance of the data structure is *primarily* measured by the following two parameters: (i) *query time*, the time taken to answer the query, and (ii) *size/space*, the size of the data structure. If the input data is dynamic, then *update time*, the time taken to handle insertion/deletion of an object is also of interest. A *secondary* measure is the *preprocessing time*, which is the time taken to build the data structure. The goal is to obtain good upper and/or lower bounds on these parameters, which is done via rigorous mathematical analysis.

1.1 Landscape of GIQ problems

Figure 1.2 is a succinct representation of the vast landscape of GIQ problems. It shows the different aspects that come into play, such as computational models, geometric setting, aggregation functions, fundamental data structures and algorithms, types of data, etc. We briefly touch upon some of these aspects.

1.1.1 Geometric setting

A GIQ problem is primarily determined by the geometric objects in $\mathcal{A}$ and q. Similar to the two scenarios presented at the beginning of the chapter (for studying orthogonal range searching and rectangle stabbing), there are real-world applications which have motivated the study of various other geometric settings. Please see Figure 1.1 which illustrates a few well-studied geometric settings.

1.1.2 Aggregation function

Aggregation functions studied in the literature can be classified into the following two categories:

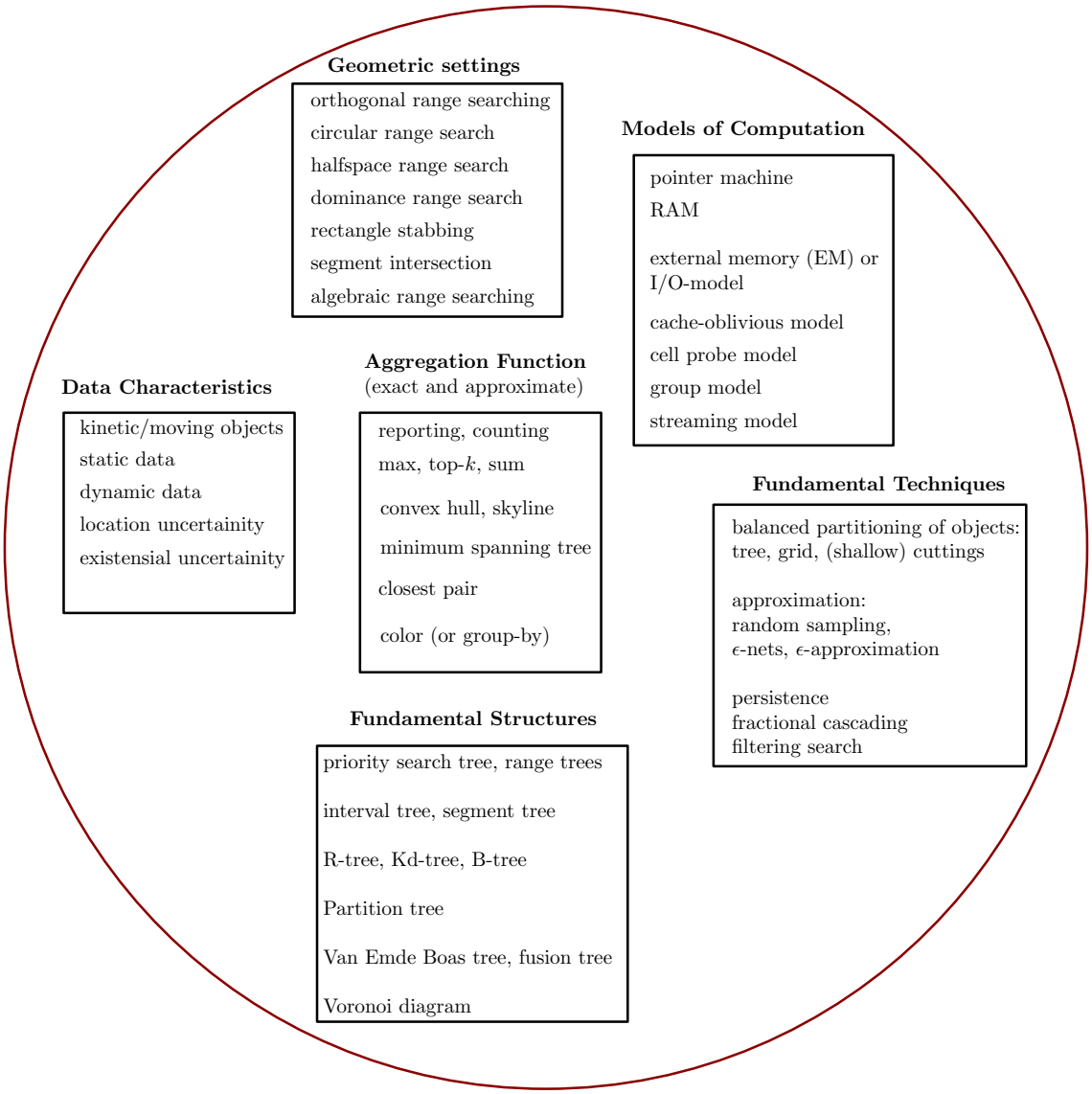


Figure 1.2: Landscape of geometric intersection queries (GIQ).

Classical aggregation functions: GIQ problems have been an active field of research since the 1970s. Some of the classical aggregation functions studied are *reporting*, *counting*, *max*, and *sum*. In a reporting query we report the objects in $\mathcal{A} \cap q$, in a counting query we report $|\mathcal{A} \cap q|$, in a max query we report the object in $\mathcal{A} \cap q$ with the largest-weight, and so on.

In spite of many decades of research on these aggregate functions, there are still open problems under various geometric settings. Solutions to these classical problems form the basis for building solutions for newer GIQ problems. References [1, 2] are our key contributions to this class of problems.

Modern aggregation functions: In recent years, there has been an explosion in the volume of digital data that is being generated and stored. Querying such large datasets with new aggregate functions provides users more insights into the data. Moreover, modern display devices such as smartphones have relatively low computing power and small screens. Displaying all the items on a small screen would lead to clutter, and hence, will not be informative. Our work in [3, 4, 5, 6, 7, 8] is part of a growing literature to address these challenges by proposing practically useful aggregation functions, and then providing efficient solutions for them. We have focussed on aggregate functions *top-k* where we report the k largest-weighted objects in $\mathcal{A} \cap q$, and *approximate counting* where we report an approximation of $|\mathcal{A} \cap q|$.

1.1.3 Models of Computation

Before designing a data structure, an algorithm designer has to fix a *model of computation*. It decides (a) *permissible operations*: operations which can be performed on the data (and operations which are not allowed), and (b) *data access*: where the data is stored and how it can be accessed. The solutions in this thesis have been built for the following three models.

Word-RAM model. In this model [9], we have a collection of cells, each of which is a w-bit word. Each cell can, therefore, store integer values in the range $\{0, \ldots, 2^w - 1\}$. Random access to any cell can be performed in constant time. Basic operations on words (which are performed in modern programming languages such as C, C++, or Java) take constant time. This includes arithmetic operations (such as $+, -, *, /, \%$), comparisons

$(<, >, =)$, and bitwise boolean operations (bitwise-AND, OR, and exclusive-OR). We assume that $w \geq \log U$ and $w \geq \log n$, so that the coordinate of any object fits in a single word and the *memory location* of any of the n objects also fits in a single word, respectively. The space of the data structure is measured in terms of the number of words/cells occupied.

I/O-model or external memory model. The dawn of big-data led to the introduction of *external memory (EM) or I/O-model* [10]. Many massive datasets cannot be completely stored in the main memory and hence reside in external devices such as hard disks. It has been observed that the major bottleneck is the time taken to access the disk: it takes orders of magnitude more time to access the disk than the time taken to perform computations in the main memory. Therefore, the main objective of this model is to minimize the number of I/Os (Input/Output) performed between the main memory and the external device. In this model, a machine is equipped with M words of main memory, and a disk that has been formatted into *blocks* of B words each (we assume $B \geq 64$). The values of M and B satisfy $M \geq 2B$. An I/O either reads a disk block into memory, or writes B words of memory into a disk block. The *query time* of an algorithm is measured in the number of I/Os performed, while the *space* of a structure is measured in the number of disk blocks occupied.

Pointer machine model. This model has been used extensively for proving several interesting lower bounds and upper bounds for range searching and related problems. Loosely speaking, in this model the data structure is modeled as a graph and one is not allowed to do a random access. Formally, as defined by Tarjan [11], in this model a data structure can be regarded as a directed graph, where each node stores $O(1)$ real values and $O(1)$ pointers to other nodes. Random access to a node is not allowed and only pointers can be used to access a node. We begin answering a query using a pointer to a *root node* of the data structure. The *query time* of an algorithm is the total number of nodes visited, whereas the *size* of a structure is the number of its nodes and edges. Further details can be found in Agarwal and Erickson [12].

1.1.4 Fundamental Structures and Techniques

We give the reader a quick tour of some of the fundamental structures and techniques which have been invented for answering GIQ problems in the last five to six decades. In the 1970's a balanced partition of the geometric objects or the underlying space was the key approach for build range searching data structures. This lead to the invention of fundamental structures such as range tree [13], Kd-tree [14], quadtree [15], B-tree [16], and priority search tree [17]. Most of these structures came with good theoretical guarantees. On the database side, the R-tree [18] and its numerous variants were invented in the 1980's to efficiently handle range queries on spatial data in external memory. The field of computational geometry started getting more sophisticated with the invention of powerful tools such as persistence [19, 20], fractional cascading [21], and filtering search [22].

In the late 1980's, invention of tools such as ε-sample [23], ε-nets [24], and moments technique [25] (all of which make use of random sampling) led to the emergence of modern computational geometry. It revolutionized many areas within computational geometry including GIQ problems. For example, it led to the efficient construction of cuttings and shallow cuttings for halfspaces which are used as tools for solving GIQ problems dealing with halfspaces.

In the meanwhile, different types of data structures were invented to efficiently perform basic operations on integer data. These data structures generously exploited the full power of the RAM model, which not only lets us do comparisons but also arithmetic operations and bitwise boolean operations. The van Emde Boas tree [26, 27] and the fusion tree [9] are two classic structures which perform predecessor/successor search on an integer data coming from a fixed universe.

The last two decades have witnessed sustained activity on GIQ problems. A lot of the focus has been on external memory structures and structures for integer data (word-RAM structures). Some of the key features of the newer data structures have been the use of trees with large fanout, the use of shallow cuttings to solve orthogonal GIQ problems (not just halfspace/algebraic GIQ problems), and the use of a stronger version of filtering search.

Typical space and query time bounds for GIQ problems have an exponential dependence on the dimension size, say d. For example, standard range trees occupy

$O(n \log^{d-1} n)$ space and answer an orthogonal range counting query in $O(\log^{d-1} n)$ time. If $d = \log n$, then the query time of this structure is greater than n, which is worse than a brute-force scan of the entire dataset. For $d = \Omega(\log n)$, the grand question is whether a sub-linear (i.e. $o(n)$) query time solution is possible using a reasonable size data structure, say $O(n^2)$ or $O(n^{d/2})$? Recently, there has been progress on this question by Chan [28].

1.1.5 Data Characteristics

Traditionally GIQ problems have been formulated to handle static data (where $\mathcal{A}$ is fixed and does not change) and dynamic data (where insertion/deletion of objects into $\mathcal{A}$ is permitted). With the availability of high-quality trajectory data and real-time tracking of vehicles, GIQ problems are also being studied for the setting where $\mathcal{A}$ represents trajectory data [29] and moving objects [30], respectively.

Typically, the data is assumed to be precise; however, some sources of data such as GPS or network sensors are imprecise, and over the last few years there are attempts being made to model this imprecise data and answer GIQ problems on them [31, 32, 33].

1.2 Contributions of the Thesis

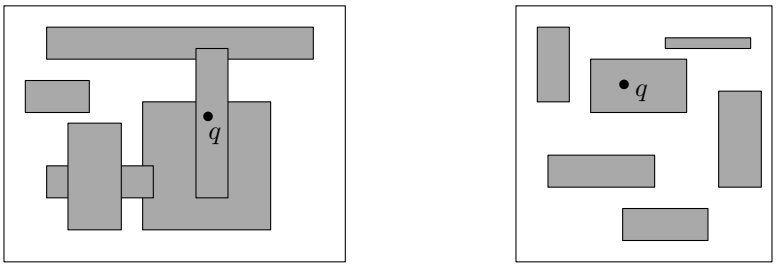
This thesis presents new results for various GIQ problems. The results presented in this thesis will be presented in three parts. Part-I of the thesis will focus on two fundamental problems in the field of computational geometry. These problems have been of interest to the community for the past four decades. Part-II and Part-III of the thesis study problems that have emerged within the past decade or so.

Part-I: Orthogonal point location in 3-d. In this problem, we preprocess a set of n axes-aligned *disjoint* boxes/hyper-rectangles in $\mathbb{R}^d$ into a data structure, so that the box (if any) containing a given query point can be reported efficiently (see Figure 1.3(b)). In 2-d the problem is well-understood; an optimal solution is known in various models of computation: for example, in the pointer machine model there is an $O(n)$ size data structure which can answer the query in $O(\log n)$ time.

This thesis presents an *optimal* solution for the orthogonal point location query in 3-d in the pointer machine model and the I/O model: a linear-space structure which can answer the query in $O(\log n)$ time and $O(\log_B n)$ I/Os, respectively. In the word-RAM model, we have succeeded in surpassing the $\log n$ barrier in the query time.

Part I: Rectangle stabbing in 3-d. In this problem, $\mathcal{A}$ is a set of n hyper-rectangles (possibly overlapping) that lie in $\mathbb{R}^d$, so that given a query point q, we can report all the rectangles in $\mathcal{A}$ containing q (see Figure 1.3(a)). Optimal solutions for rectangle stabbing in $\mathbb{R}^1$ and $\mathbb{R}^2$ were discovered as early as the 1980s: an $O(n)$ space data structure which can answer the query in $O(\log n + k)$ time, where k is the number of rectangles reported. The data structure in $\mathbb{R}^1$ is the classical interval-tree data structure and in $\mathbb{R}^2$ is the hive-graph data structure of Chazelle [22].

However, for the past three decades an optimal solution in $\mathbb{R}^3$ had been elusive. In the pointer-machine model, there was a known lower bound of $\Omega(\log^2 n + k)$ query time for a linear-space data structure in $\mathbb{R}^3$ [34]; whereas, the state-of-the-art linear-space data structure took $O(\log^4 n + k)$ time to answer a query [35]. This thesis presents an *almost* optimal solution: an $O(n \log^* n)$ size data structure which can answer the query in $O(\log^2 n \cdot \log\log n + k)$ time.



(a) Overlapping rectangles (b) Disjoint rectangles

Figure 1.3: (a) Rectangle stabbing problem shown in $\mathbb{R}^2$, (b) Orthogonal point location problem shown in $\mathbb{R}^2$.

Part-II: Top-k Geometric intersection queries (Top-k GIQ). We have done an extensive study of top-k GIQ problems. In a top-k GIQ problem, each object in $\mathcal{A}$ has a weight associated with it (which is determined by some ranking criteria) and the user

would want to know the k largest-weight objects of $\mathcal{A}$ intersected by q. For example, let $\mathcal{A}$ be a set of points in the plane and q be an axes-aligned rectangle (see Figure 1.4). The points could represent restaurants and the rating of each restaurant could be its weight. The query rectangle q can be the downtown area in New York City and the user might want to know the top-5 rated restaurants in that area. This setting is known as *top-k orthogonal range searching*.

Our work on top-k GIQ can be classified into two categories. In the first category the effort is to build an efficient solution for a particular geometric setting. For example, in [5] we present the first known optimal solution for top-k orthogonal range searching in $\mathbb{R}^2$ in the pointer machine model and an almost-optimal solution in the external-memory setting. Previous work on this problem could guarantee an optimal solution only when points lie in $\mathbb{R}^1$. Also, in an unpublished report [8] we present an optimal solution for top-k halfplane range searching problem in $\mathbb{R}^2$. In the interest of space, these results [5, 8] are omitted in the thesis.

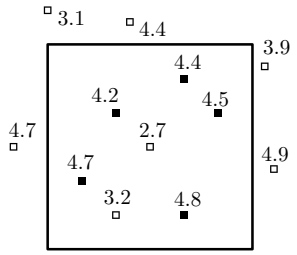


Figure 1.4: Top-5 rated restaurants shown as solid black squares.

In the second category [4, 6], the effort was in coming up with general techniques (which we will henceforth refer to as reductions) which can handle top-k GIQ problems for any combination of input objects and query object. The best way to describe these reductions would be "Top-k Indexes made Small and Sweet" (which was the title of the invited talk given by one my collaborators, Prof. Yufei Tao, at EDBT/ICDT 2016). The reductions are "small" because they are easy to implement, and they are "sweet" because they come with non-trivial theoretical guarantees in terms of space and query time bounds. These reductions have been discussed in full detail in this thesis.

Part-III: Approximate range counting. Let $\mathcal{A}$ be a set of n geometric objects in $\mathbb{R}^d$ which are segregated into disjoint groups (i.e., *colors*). Given a query $q \subseteq \mathbb{R}^d$, a color c *intersects (or, is present in)* q if any object in $\mathcal{A}$ of color c intersects q, and let k be the number of colors of $\mathcal{A}$ present in q. In the *approximate colored range-counting problem*, the task is to preprocess $\mathcal{A}$ into a data structure, so that for a query q, one can efficiently report the *approximate* number of colors present in q. Specifically, return any value in the range $[(1-\varepsilon)k, (1+\varepsilon)k]$, where $\varepsilon \in (0,1)$ is a pre-specified parameter.

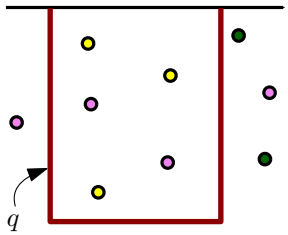


Figure 1.5: An instance of the colored setting.

These are known as GROUP-BY queries in the database literature. A popular variant is the *colored orthogonal range searching* problem: $\mathcal{A}$ is a set of n colored points in $\mathbb{R}^d$, and q is an axes-aligned rectangle. As a motivating example for this problem, consider the following query: "How many countries have employees aged between X_1 and X_2 while earning annually more than Y dollars?". An employee is represented as a colored point $(age, salary)$, where the color encodes the country, and the query is the axes-aligned rectangle $[X_1, X_2] \times [Y, \infty)$.

In [7], new results for approximate range counting are presented. Most of the results are obtained via reductions to the approximate uncolored version, and improved data-structures for them. A key contribution of this work is the introduction of *nested shallow cuttings* (which have stronger properties than the regular *shallow cuttings*) for rectangles in $\mathbb{R}^2$. Nested shallow cuttings will lead to improved solutions for other aggregate functions on rectangles in $\mathbb{R}^2$ as well. Similar to top-k GIQ, another important contribution is general reductions which can handle approximate counting query for any combination of input objects and query object. The most interesting reduction requires using two companion structures: (a) *reporting structure* (its objective is to report the k colors), and (b) *C-approximation structure* (its objective is to report any value z s.t. $k \in [z, Cz]$, where C is a constant). Significantly, unlike previous reductions [36, 37],

there is *no asymptotic loss* of efficiency in space and query time bounds w.r.t. to the two companion problems. To keep the thesis short, only a subset of the results from [7] have been included. These results are also cited in recent surveys [38, 39].

Flow of the thesis. As mentioned earlier, the main body of the thesis consists of three parts. Part-I will present solutions for orthogonal point location in 3-d and rectangle stabbing in 3-d. Part-II will present solutions for top-k GIQ problems. Part-III will present solutions for approximate range counting problems.

Part I

Orthogonal Point Location and Rectangle Stabbing in 3-d

Chapter 2

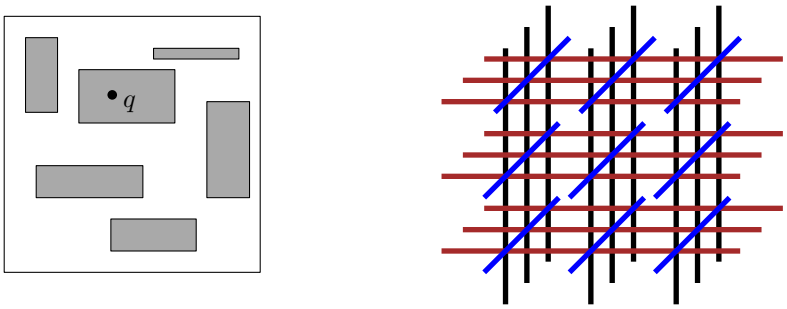
Orthogonal Point Location in 3-d

2.1 Problem Statement

Point location is a fundamental problem in the field of computational geometry. In this chapter we study the *orthogonal point location* problem. Formally, we want to preprocess a set of n axes-aligned disjoint boxes (hyperrectangles) in $\mathbb{R}^d$ into a data structure, so that the box in the set containing a given query point (if any) can be reported efficiently. See Figure 2.1(a). A special case of this problem is when the input boxes fill the entire space forming a *subdivision*. In this work we consider the general setting, where the entire space need not be filled by the boxes.

2.2 Previous Work

Orthogonal point location in 2-d. In the plane, the two versions of the problem are equivalent in the sense that any arbitrary set of n disjoint rectangles can be converted into a subdivision of $\Theta(n)$ rectangles via the vertical decomposition. Optimal solutions are known for this problem in all models we consider, namely, linear-space data structures with $O(\log n)$ query time [40, 41, 42, 19, 43] in the pointer machine model, $O(\log_B n)$ query cost [44, 45] in the I/O model with block size B, and $O(\log \log U)$ query time [46] in the word-RAM model with input coordinates in $[U] = \{0, 1, \ldots, U-1\}$. (The first two results actually hold for nonorthogonal point location.)



(a) (b)

Figure 2.1: (a) An illustration of an orthogonal point location query in $\mathbb{R}^2$. (b) A setting consisting of $\Theta(n)$ boxes which would require $\Omega(n^{3/2})$ additional boxes to fill the entire space.

Orthogonal point location in 3-d. In 3-d, the two versions are no longer equivalent, since there exist sets of n disjoint boxes that need $\Omega(n^{3/2})$ boxes to fill the entire space. See Figure 2.1(b). This makes the general setting (the focus of this thesis) potentially harder than the special case of a subdivision, as the latter allows for fast $O(\log^2 \log U)$ query time in the word-RAM model with $O(n \log \log U)$ space, as shown by de Berg, van Kreveld, and Snoeyink [47] (with an improvement by Chan [46]). For non-space-filling boxes that are *fat*, Iacono and Langerman [48] achieved fast $O(\log \log U)$ query time in the word-RAM model, using $O(n \log \log U)$ space (their result actually holds in any constant dimension). For general non-space-filling boxes in 3-d, however, the best known results are linear-space data structures with $O(\log^{3/2} n)$ query time by Rahul [1] in the pointer machine model, $O(\log_B^2 n)$ query cost by Nekrich [49] in the I/O model, and $O(\log n \log \log n)$ query time in the word-RAM model. (That last result was not stated explicitly before but can obtained by an interval tree augmented with Chan's 2-d orthogonal point location structure [46].)

2.3 Our results

Our main results are the first *optimal* data structures for orthogonal point location queries in 3-d in the (arithmetic) pointer machine model and the I/O model. In the word-RAM, we succeed in surpassing the $\log n$ barrier in the query time. We also obtain the first linear-space data structure for the case of subdivisions. See Table 2.1

for a comparison of our results with previous work.

Table 2.1: Orthogonal point location in 3-d using linear space in the pointer machine model, I/O model, and the word-RAM model.

Model	**Reference**	**Query Time**
Pointer Machine	Edelsbrunner, Haring, and Hilbert'86 [50]	$\log^2 n$
	Afshani, Arge, and Larsen (SoCG'10) [51]	$\frac{\log^2 n}{\log\log n}$
	Rahul (SODA'15) [1]	$\log^{3/2} n$
	New (Theorem 2.6.1)	$\log n$ (optimal)
I/O	Nekrich (LATIN'08) [49]	$\log_B^2 n$
	New (Theorem 2.6.1)	$\log_B n$ (optimal)
Word-RAM	Chan (SODA'11) [46]	$\log n \log\log n$
	New (Theorem 2.6.1)	$\log_w n$ $(\leq \frac{\log n}{\log\log n})$

In the pointer machine model, improvements in 3-d automatically lead to improvements in higher dimensions, by using interval trees, which impose a $\log n$ time-overhead per dimension: we get a linear-space data structure with $O(\log^{d-2} n)$ query time for d-dimensional queries, which is better than previous methods [50, 51], the best of which had $O(\log^{d-3/2} n)$ query time [1].

2.4 Techniques

The main virtue of this work is the simplicity of our method (especially when compared against previous methods such as [1]). Our solution combines two ideas: (a) a van Emde Boas style partition over a single dimension that reduces the problem to 2-d *rectangle stabbing emptiness* (also called *point enclosure emptiness*), i.e., store a set of possibly overlapping axes-aligned rectangles, so as to determine whether any of them contains a given query point, and (b) quickly shrinking the universe size by applying this partition in a *round-robin* fashion over all three dimensions. Note that the original van Emde Boas recursion was designed to obtain $O(\log\log U)$-like bounds, but we will use it to obtain logarithmic-like bounds, interestingly.

2.5 Preliminaries: On Two Subroutines

Our solution to 3-d orthogonal point location will require known data structures for *2-d orthogonal point location* and *2-d rectangle stabbing emptiness*.

Lemma 2.5.1. *Given n disjoint axes-aligned rectangles in $[U]^2$ ($n \leq U \leq 2^w$), there are data structures for point location with $O(\frac{n \log U}{w})$ words of space and*

- $O(\log n)$ *query time in the pointer machine model;*
- $O(\log_B n)$ *query cost in the I/O model;*
- $O(\min\{\log\log U, \log_w n\})$ *query time in the word-RAM model.*

Lemma 2.5.2. *Given n (possibly overlapping) axes-aligned rectangles in $[U]^2$ ($n \leq U \leq 2^w$), there are data structures for rectangle stabbing emptiness with $O(\frac{n \log U}{w})$ words of space and*

- $O(\log n)$ *query time in the pointer machine model;*
- $O(\log_B n)$ *query cost in the I/O model;*
- $O(\log_w n)$ *query time in the word-RAM model.*

For Lemma 2.5.1, such data structures for 2-d orthogonal point location can be found in [40, 41, 42, 19, 43] for the pointer machine model, [44, 45] for the I/O model, and [46] for the word-RAM model. For Lemma 2.5.2, 2-d rectangle stabbing emptiness (or more generally, rectangle stabbing counting) is known to be reducible to 2-d orthogonal range counting [52], and such data structures for 2-d orthogonal range counting can be found in [53] for the pointer machine model, [54] for the I/O model, and [55] for the word-RAM model.

All these known data structures technically require $O(n)$ words of space, or more precisely, $O(n \log U)$ bits of space. In the I/O model or word-RAM model, we can easily pack the data structures in $O(\frac{n \log U}{w})$ words of space without increasing the query cost when $\log U \ll w$. In the pointer machine model, we may not be able to pack the data structures in general, since if multiple "micro-pointers" are packed in a word, the model does not allow us to follow such a micro-pointer. Nevertheless, it is not difficult to modify the existing data structures to achieve the compressed space bound.

We now provide missing details on the subroutines for 2-d orthogonal point location and 2-d rectangle stabbing emptiness (Lemmas 2.5.1 and 2.5.2) in the pointer machine model. Existing methods already achieve $O(\log n)$ time and $O(n)$ space, but we want $O(\frac{n \log U}{w})$ words of space.

Proof of Lemma 2.5.1 for pointer machines. For 2-d orthogonal point location, one solution is via $(1/r)$-*cuttings* [56]: we can partition the plane into $O(r)$ disjoint rectangular cells, each intersecting $O(n/r)$ line segments (edges of the input rectangles), where we choose $r = \frac{\delta n \log U}{w}$ for a suitable constant $\delta > 1$.

We build a point location structure [40, 41, 42, 19, 43] for the $O(r)$ cells with $O(\log r)$ query time in the pointer machine model; the space usage of this structure in words is $O(r)$, which is within the allowed bound $O(\frac{n \log U}{w})$, so there is no need for bit packing here.

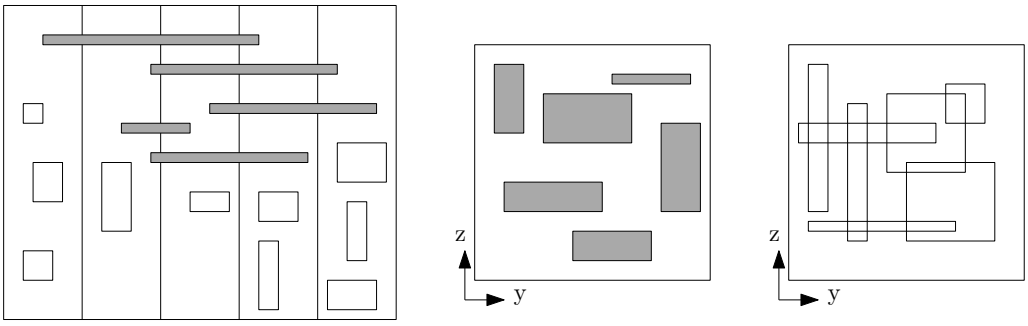
For each cell, we store the $O(n/r)$ line segments in a point location structure [41] with $O(\log(n/r))$ query time; the space usage of this structure in bits is $O((n/r) \log U)$, which is $O(w/\delta)$, so the entire structure can be packed in a single word. Although pointer chasing is not directly supported in the pointer machine model when multiple "micro-pointers" are packed in a word, we can simulate each pointer chasing step here in constant time by arithmetic operations and shifts within the word.

Given a query point q, we can first find the cell containing q in $O(\log r)$ time and then finish the query inside the cell in $O(\log(n/r))$ time. The overall query time is $O(\log n)$.

Proof of Lemma 2.5.2 for pointer machines. Rectangle stabbing emptiness in 2-d reduces to dominance range counting in 2-d [52]. Chazelle's *compressed range tree* structure [53] solves the latter problem with $O(n)$ words of space and $O(\log n)$ time in the pointer machine model. We observe that his data structure actually achieves $O(\frac{n \log U}{w})$ words of space, after minor modifications.

At each level of the range tree, Chazelle's structure stores lists consisting of a total of $O\left(\frac{n}{w}\right)$ words ($O\left(\frac{n}{w}\right)$ w-bit integers as well as $O(\frac{n}{w})$ pointers to words in lists at the next level). The total number of words over all levels of the tree is $O(\frac{n \log n}{w}) \leq O(\frac{n \log U}{w})$.

We shorten the tree by making the leaf nodes contain b points, where we choose



$\sqrt{U}$ slabs along x-axis (a)

Side boxes of a slab (b)

Bottom boxes of a slab (c)

Figure 2.2: Boxes obtained after partitioning along the x-direction.

$b = \frac{\delta w}{\log U}$ for a sufficiently small constant δ. This way, the space in words for the tree itself is $O(n/b) = O(\frac{n \log U}{w})$. Inside each leaf, we store the b points in another instance of Chazelle's structure; the space usage of this structure in bits is $O(b \log U)$, which is $O(\delta w)$, so the entire structure can be packed in a single word. Again, we can simulate each pointer chasing step here in constant time by arithmetic operations and shifts within the word.

To answer a dominance range counting query, we descend along a path in the compressed range tree, which requires $O(\log(n/b))$ time by following pointers in the lists stored at the path and doing various arithmetic operations and shifts on w-bit integers. At the leaf of the path, we can finish the query in $O(\log b)$ time. The overall query time is $O(\log n)$.

2.6 Orthogonal Point Location in 3-d

We are now ready to describe our data structure for 3-d orthogonal point location. We focus on the pointer machine model first.

2.6.1 Data Structure

At the beginning, we apply a rank space reduction (replacing input coordinates by their ranks) so that all coordinates are in $[2n]^3$, where n is the global number of input

boxes. Given a query point, we can initially find the ranks of its coordinates by three predecessor searches (costing $O(\log n)$ time in the pointer machine model).

We describe our preprocessing algorithm recursively. The input to the preprocessing algorithm is a set of n disjoint boxes that are assumed to be aligned to the $[U_x] \times [U_y] \times [U_z]$ grid. (At the beginning, $U_x = U_y = U_z = 2n$.)

Without loss of generality, assume that $U_x \geq U_y, U_z$. We partition the $[U_x] \times [U_y] \times [U_z]$ grid into $\sqrt{U_x}$ equal-sized vertical slabs perpendicular to the x-direction. See Figure 2.2. (In the symmetric case $U_y \geq U_x, U_z$ or $U_z \geq U_x, U_y$, we partition along the y- or z-direction instead.) We classify the boxes into two categories:

- *Bottom boxes.* For each slab, define its bottom boxes to be those that lie completely inside the slab.
- *Top boxes.* Top boxes intersect the boundary (vertical plane) of at least one slab. Each top box $\mathcal{B}$ is broken into three disjoint boxes:
  - *Left box.* Let s_L be the slab containing the left endpoint (with respect to the x-axis) of $\mathcal{B}$. The left box is defined as $\mathcal{B} \cap s_L$.
  - *Right box.* Let s_R be the slab containing the right endpoint of $\mathcal{B}$. The right box is defined as $\mathcal{B} \cap s_R$.
  - *Middle box.* The remaining portion of box $\mathcal{B}$ after removing its left and right box, i.e. $\mathcal{B} \setminus ((\mathcal{B} \cap s_L) \cup (\mathcal{B} \cap s_R))$.

We build our data structure as follows:

1. *Planar point location structure.* For each slab, we project its left boxes onto the yz-plane. The projected boxes remain disjoint, since they intersect a common boundary. We store them in a data structure for 2-d orthogonal point location by Lemma 2.5.1. We do this for the slab's right boxes as well.
2. *Rectangle stabbing structure.* For each slab, we project its bottom boxes onto the yz-plane. The projected boxes are not necessarily disjoint. We store them in a data structure for 2-d rectangle stabbing emptiness by Lemma 2.5.2.
3. *Recursive top structure.* We recursively build a *top structure* on all the middle boxes.

4. *Recursive bottom structures.* For each slab, we recursively build a *bottom structure* on all the bottom boxes inside the slab.

By translation or scaling, these recursive bottom structures or top structure can be made aligned to the $\left[\sqrt{U_x}\right] \times [U_y] \times [U_z]$ grid. In addition, we store the mapping from left/right/middle boxes to their original boxes, as a list of pairs (sorted lexicographically) packed in $O\left(\frac{n \log(U_x U_y U_z)}{w}\right)$ words.

2.6.2 Query Algorithm

The following lemma is crucial for deciding whether to query recursively the top or the bottom structure.

Lemma 2.6.1. *Given a query point (q_x, q_y, q_z), if the query with (q_y, q_z) on the rectangle stabbing emptiness structure of the slab that contains q_x returns*

- NON-EMPTY*, then the query point cannot lie inside a box stored in the top structure, or*
- EMPTY*, then the query point cannot lie inside a box stored in the slab's bottom structure.*

Proof. If NON-EMPTY is returned, then the query point is stabbed by the extension (along the x-direction) of a box in the slab's bottom structure and cannot be stabbed by any box stored in the top structure, because of disjointness of the input boxes. If EMPTY is returned, then obviously the query point cannot lie inside a box stored in the bottom structure. □

To answer a query for a given point (q_x, q_y, q_z), we proceed as follows:

1. Find the slab that contains q_x by predecessor search over the slab boundaries.
2. Query with (q_y, q_z) the planar point location structures at this slab. If a left or a right box returned by the query contains the query point, then we are done.
3. Query with (q_y, q_z) the rectangle stabbing emptiness structure at this slab. If it returns NON-EMPTY, query recursively the slab's bottom structure, else query

recursively the top structure (after appropriate translation/scaling of the query point).

In step 3, to decode the coordinates of the output box, we need to map from a left/right/middle box to its original box; this can be done naïvely by another predecessor search in the list of pairs we have stored.

2.6.3 Query Time Analysis

Let $Q(U_x, U_y, U_z)$ denote the query time for our data structure in the $[U_x] \times [U_y] \times [U_z]$ grid. Observe that the number of boxes n is trivially upper-bounded by $U_x U_y U_z$ because of disjointness. The predecessor search in step 1, the 2-d point location query in step 2, and the 2-d rectangle stabbing query in step 3 all take $O(\log n) = O(\log(U_x U_y U_z))$ time by Lemmata 2.5.1 and 2.5.2. We thus obtain the following recurrence, assuming that $U_x \geq U_y, U_z$:

$$Q(U_x, U_y, U_z) = Q\left(\sqrt{U_x}, U_y, U_z\right) + O\left(\log\left(U_x U_y U_z\right)\right).$$

If $U_x = U_y = U_z = U$, then three rounds of recursion will partition along the x-, y-, and z-directions and decrease U_x, U_y, and U_z in a round-robin fashion, yielding

$$Q(U, U, U) = Q\left(\sqrt{U}, \sqrt{U}, \sqrt{U}\right) + O\left(\log U\right),$$

which solves to $Q(U, U, U) = O(\log U)$. As $U = 2n$ initially, we get $O(\log n)$ query time.

2.6.4 Space Analysis

Let $s(U_x, U_y, U_z)$ denote the amortized number of words of space needed per input box for our data structure in the $[U_x] \times [U_y] \times [U_z]$ grid. The amortized number of words per input box for the 2-d point location and rectangle stabbing structures is $O\left(\frac{\log(U_x U_y U_z)}{w}\right)$ by Lemmata 2.5.1 and 2.5.2. We thus obtain the following recurrence, assuming that $U_x \geq U_y, U_z$:

$$s(U_x, U_y, U_z) = s\left(\sqrt{U_x}, U_y, U_z\right) + O\left(\frac{\log\left(U_x U_y U_z\right)}{w}\right).$$

Three rounds of recursion yield

$$s(U, U, U) = s\left(\sqrt{U}, \sqrt{U}, \sqrt{U}\right) + O\left(\frac{\log U}{w}\right),$$

which solves to $s(U,U,U) = O\left(\frac{\log U}{w}\right)$. As $U = 2n$ initially, the total space in words is $O\left(n\frac{\log n}{w}\right) \leq O(n)$.

Note that the above analysis ignores an overhead of $O(1)$ words of space per node of the recursion tree, but by shortcutting degree-1 nodes, we can bound the number of nodes in the recursion tree by $O(n)$.

2.6.5 Other Models

In the I/O model, the analysis is similar, with a modified recurrence for the query cost:

$$Q(U,U,U) = Q\left(\sqrt{U},\sqrt{U},\sqrt{U}\right) + O(\log_B U).$$

For the base case $U \leq B^{1/3}$, we have $Q(U,U,U) = O(1)$ trivially, since $n \leq U^3 \leq B$. Solving the recurrence yields $O(\log_B n)$ query cost. The space usage remains $O(n)$ words (i.e., $O(n/B)$ blocks).

In the word-RAM model, the analysis is again similar, with

$$Q(U,U,U) = Q\left(\sqrt{U},\sqrt{U},\sqrt{U}\right) + O(\log_w U).$$

For the base case $U \leq w$, we have $Q(U,U,U) = O(1)$ by switching to another known method: Orthogonal point location in 3-d reduces to 6-d dominance emptiness, for which there is a known method [57] with $O(n(\log_w n)^4)$ words of space and $O((\log_w n)^5)$ query time in the word-RAM. (The method in [57] can be modified to report a witness if the range is non-empty.) Since $n \leq U^3 \leq w^3$, we have $\log_w n = O(1)$, and so the space bound is $O(n)$ and query bound is $O(1)$ for the base case. Solving the recurrence yields $O(\log_w n)$ query time.

To summarize, we have obtained the following results:

Theorem 2.6.1. *Given n disjoint axes-aligned boxes in 3-d, there are data structures for point location with $O(n)$ words of space and*

- $O(\log n)$ *query time in the pointer machine model;*
- $O(\log_B n)$ *query cost in the I/O model;*
- $O(\log_w n)$ *query time in the word-RAM model.*

2.7 Extensions

Higher dimensions. The same approach can be extended to higher dimensions, reducing the complexity of d-dimensional orthogonal point location to that of $(d-1)$-dimensional box stabbing emptiness. However, known data structures for higher-dimensional box stabbing [34] requires superlinear space, whereas the simpler approach mentioned in the Introduction, of using interval trees to reduce the dimension, gives $O(\log^{d-2} n)$ query time while keeping linear space in the pointer machine model.

The case of 3-d subdivisions. Our approach can also be used to improve the space bound of de Berg, van Kreveld, and Snoeyink's point location structure [47] for 3-d orthogonal subdivisions, from $O(n \log\log U)$ space to $O(n)$, in the word-RAM model.

Theorem 2.7.1. *Given a subdivision formed by n disjoint (space-filling) axes-aligned boxes in 3-d, there is a data structure for point location with $O(n)$ words of space and $O(\log^2 \log n)$ query time in the word-RAM model.*

Proof. (Sketch) De Berg et al.'s method [47, Theorem 2.4] was already based on a van Emde Boas recursion, partitioning along the x-direction. They also used 2-d orthogonal point location structures during the recursion, but managed to avoid rectangle stabbing structures by exploiting the fact that the input is a subdivision. Roughly, for each slab, they took the "holes" formed by all middle boxes that intersect the slab, and filled the holes by taking the vertical decomposition of the yz-projection. The analysis followed by charging the complexity of the decomposition to vertices within the slab.

Our new change is to do the van Emde Boas recursion not just along the x-direction but along all three axis directions in a round-robin fashion. This leads to the same recurrence for space as in Section 2.6. The query time satisfies the following recurrence:

$$Q(U,U,U) = Q\left(\sqrt{U},\sqrt{U},\sqrt{U}\right) + O(\log\log U).$$

This leads to $O(\log^2 \log n)$ query time. □

2.8 Open questions

An intriguing question is to determine if our $O(\log_w n)$ query time bound for 3-d orthogonal point location for disjoint boxes is optimal, or if $(\log\log U)^{O(1)}$ bounds are at

all possible, in the word-RAM model. Also, we are not aware of any nontrivial lower bound for d-dimensional point location for disjoint boxes, to indicate that the number of logarithmic factors has to grow as d increases.

Chapter 3

Rectangle Stabbing in 3-d

3.1 Problem Statement

Rectangle stabbing is another well-studied problem in the field of computational geometry. This problem is also referred to as *orthogonal point enclosure query (OPEQ)* in the literature. We will present an *almost* optimal solution for this problem in 3-d in the pointer machine model.

In an $OPEQ$, we preprocess a set S of n axes-aligned rectangles in $\mathbb{R}^d$, so that given a query point $q \in \mathbb{R}^d$, we can efficiently report all the rectangles in S containing (or stabbed by) q. There are a lot of practical applications of this query in various domains such as GIS, recommender systems, networking etc. For example, on a flight booking website such as *kayak.com*, users can specify their preference as a d-dimensional rectangle: "I am looking for flights with price in the range \$100 to \$300 and with departure date in the range 1st March to 5th March". Here price is the x-axis and departure date is the y-axis. Given a particular flight, all the users whose preference match this flight can be found out by posing an $OPEQ$ with q *(price, departure date)* $\in \mathbb{R}^2$ as the query point.

3.2 Previous results

There are several ways of obtaining an optimal solution of $O(n)$ space and $O(\log n + k)$ query time in $\mathbb{R}^1$, where k is the number of intervals reported [58, 59, 60]. In $\mathbb{R}^2$ an

optimal solution of $O(n)$ space and $O(\log n + k)$ query time was obtained by Chazelle [22]. He introduced the hive-graph structure to answer the query. Later, another solution with optimal bounds was presented in [61] using a combination of persistence and interval tree.

By using segment trees [58, 60], we can generalize the optimal structure in $\mathbb{R}^2$ to higher dimensions. In $\mathbb{R}^d$ the space occupied will be $O(n \log^{d-2} n)$ and the query time will be $O(\log^{d-1} n + k)$. Afshani *et al.* [34] extend the above result for any parameter $h \geq 2$, to obtain an $O(nh \log^{d-2} n)$ space and $O(\log n \cdot (\log n / \log h)^{d-2} + k)$ query time solution. However, these structures occupy $\Omega(n \log n)$ space in $\mathbb{R}^3$. A natural question that arises is: Can an $O(n)$-space and $O(\log n + k)$-query time solution can be obtained in $\mathbb{R}^3$? The answer unfortunately is "no". Afshani *et al.* [51, 34] showed that with $O(n)$ space, the $OPEQ$ takes $\Omega(\log^2 n + k)$ time.

3.3 Our Results

We first introduce some notation to denote special kinds of rectangles in $\mathbb{R}^3$. A rectangle is called $(3 + k)$-sided if the rectangle is bounded in k out of the 3 dimensions and unbounded (on one side) in the remaining $3 - k$ dimensions. See Figure 3.1 for a 3-, 4-, 5- and 6-sided rectangle in $\mathbb{R}^3$. When rectangles are 3-sided, the $OPEQ$ can be answered in $O(\log n + k)$ query time and by using $O(n)$ space [35, 62]. However, when the rectangles are 4-sided, the best result one can achieve using existing techniques is $O(n)$ space and $O(\log^2 n + k)$ query time (see Theorem 3.4.1).

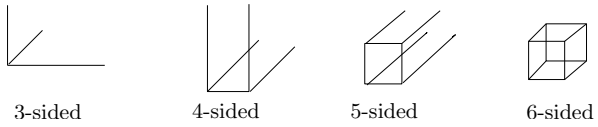


Figure 3.1: Different kinds of rectangles in three dimensional space.

The key result of our work is an almost optimal solution for 4-sided rectangles. Our first data structure uses $O(n \log^* n)$ space and answers the query in $O(\log n + k)$ time. Our second data structure uses $O(n)$ space and answers the query in $O(\log n \cdot \log^{(i)} n + k)$ time, for any constant integer $i \geq 1$. Here $\log^{(1)} n = \log n$ and $\log^{(i)} n = \log(\log^{(i-1)} n)$ when $i > 1$; $\log^* n$ is the iterated logarithm of n. At a high-level, the following are the

key ideas:

(a) As will be shown later, an $OPEQ$ for 4-sided rectangles can be answered by asking $O(\log n)$ $OPEQs$ on 3-sided rectangles. By carefully applying the idea of shallow cuttings, we succeed in answering each of the $OPEQ$ on 3-sided rectangles in "effectively" $O(\log^* n)$ time (ignoring the output term). The trick is to identify the most "fruitful" shallow cutting to answer each of the $OPEQ$ on 3-sided rectangles; this is achieved by reducing $O(\log n)$ point location queries to the problem of $OPEQ$ in $\mathbb{R}^2$. We believe this is a novel idea. This leads to a solution which takes $O(n \log^* n)$ space and $O(\log n \cdot \log^* n + k)$ query time (see Theorem 3.5.1).

(b) To further reduce the query time to $O(\log n + k)$, our next idea is to increase the fanout of our base tree. This leads to breaking down each 4-sided rectangle into two *side rectangles* and one *middle rectangle*. As will become clear later, the decrease in height of the base tree implies that the side rectangles can now be reported in $O(\log n + k)$ time, instead of $O(\log n \cdot \log^* n + k)$ time. Handling the middle rectangles is the new challenge that arises. We build a structure so that the query on middle rectangles can be handled by asking $O(\log n)$ $2d$*-dominance reporting queries*. Another novel and new idea in this work is to build a structure which can efficiently identify the $O(\log n)$ data structures on which to pose these $2d$-dominance reporting queries. Also, we are able to answer each $2d$-dominance reporting query in $O(1)$ time (ignoring the output term). This finally leads to a data structure for answering $OPEQ$ on 4-sided rectangles in $O(\log n + k)$ query time and uses $O(n \log^* n)$ space (see Theorem 3.6.1).

The result obtained for 4-sided rectangles acts as a building block to answer the $OPEQ$ in $\mathbb{R}^3$ for 5-sided rectangles using $O(n \log^* n)$ space and $O(\log n \cdot \log \log n + k)$ query time. This allows us to finally answer $OPEQ$ in $\mathbb{R}^3$ for 6-sided rectangles. See Table 3.1 for a comparison of our results with the currently best known results. Note that we are only interested in structures which occupy linear or near-linear space.

3.4 Simple structure for OPEQ using linear space

In this section, we present a simple but sub-optimal structure to answer $OPEQ$ on 3-, 4-, 5-, 6-sided rectangles.

Query	Space	Query Time	Notes
4-sided	$O(n)$	$O(\log^2 n + k)$	[35] + Interval tree
4-sided	$O(n \log^* n)$	$O(\log n + k)$	New
4-sided	$O(n)$	$O(\log n \cdot \log^{(i)} n + k)$	New
5-sided	$O(n)$	$O(\log^3 n + k)$	[35] + Interval tree
5-sided	$O(n \log^* n)$	$O(\log n \cdot \log \log n + k)$	New
6-sided	nh	$\Omega(\log^2 n / \log h + k)$	[51, 34]
6-sided	$O(n)$	$O(\log^4 n + k)$	[35] + Interval tree
6-sided	$O(n \log^* n)$	$O(\log^2 n \cdot \log \log n + k)$	New

Table 3.1: Summary of our results for orthogonal point enclosure in $\mathbb{R}^3$. $\log^* n$ is the iterated logarithm of n. $\log^{(1)} n = \log n$ and $\log^{(i)} n = \log(\log^{(i-1)} n)$ when $i > 1$ is a constant integer. Existing solutions in the literature for 6-sided rectangles require $\Omega(n \log n)$ space.

3.4.1 Handling 3-sided rectangles

Handling $OPEQ$ on 3-sided rectangles is easy. Map each 3-sided rectangle $(-\infty, x] \times (-\infty, y] \times (-\infty, z]$ into a three-dimensional point (x, y, z) and map the query point $q(q_x, q_y, q_z)$ into a 3-sided query rectangle $q' = [q_x, \infty) \times [q_y, \infty) \times [q_z, \infty)$. Therefore, the problem maps to the *three-dimensional dominance reporting query*: Report all the points lying inside the 3-sided query rectangle q'. Initially, Afshani [35] and recently, Makris and Tsakalidis [62] presented an optimal solution for three-dimensional dominance query ($O(n)$ space and $O(\log n + k)$ query time).

3.4.2 Interval tree

We shall give a brief description of a classic structure called an *interval tree* [59] (see also [60]). It has traditionally been used to answer the orthogonal point enclosure query in $\mathbb{R}^1$. We will need the interval tree to handle $OPEQ$ for 4-,5-,6-sided rectangles.

Consider a set S of n intervals in $\mathbb{R}^1$ and let E be the set of endpoints of the intervals in S. Build a binary search tree IT in which the points of E are stored at the leaves from left to right in increasing order of their coordinate value. At each node $v \in IT$, we define $split(v)$ and $range(v)$. $split(v)$ is a value such that points of E in the left (resp. right) subtree of v have coordinate value less than or equal to (resp. greater than) $split(v)$. For the root node, $root$, $range(root) = (-\infty, +\infty)$. For a node v, if

the $range(v) = [x_l, x_r]$ then the range of its left (resp. right) child will be $[x_l, split(v)]$ (resp. $(split(v), x_r]$). Each interval is assigned to exactly one node v in IT such that the interval is contained inside $range(v)$ but is not contained inside $range(\cdot)$ of its children.

Let S_v be the set of intervals assigned at node v. We maintain additional structures at node v: A list IT_v^l (resp. IT_v^r) which stores the left (resp. right) endpoints of S_v in non-decreasing (resp. non-increasing) order of their coordinate value. The space occupied by the interval tree is $O(n)$.

Given a query point q to answer orthogonal point enclosure in $\mathbb{R}^1$, we visit a path from root to the leaf node of IT, s.t., at every node v on the path, $q \in range(v)$. At each node v on the search path, if the query point q lies to the left of $split(v)$ then we traverse the list IT_v^l from left to right till the entries in the list get exhausted or we find an endpoint whose coordinate value is greater than q. The case of q lying to the right of $split(v)$ is handled symmetrically. The time taken to answer the query is $O(\log n + k)$, where k is the number of intervals reported.

3.4.3 Handling 4-,5-,6-sided rectangles

Now we present a solution to handle $OPEQ$ on 4-,5-,6-sided rectangles. First, build an interval tree IT based on the x-projection of the rectangles of S. We make the following observation to build secondary structures at each node of the interval tree.

Observation 1. *Let S_v be the set of $(4+t)$-sided rectangles (where $t \in [0,2]$) whose corresponding x-projection gets stored at node v. Consider a rectangle $r = [x_1, x_2] \times [y_1, y_2] \times [z_1, z_2] \in S_v$.*

1. *Suppose the query point $q(q_x, q_y, q_z)$ lies to the left of $split(v)$, i.e., $q_x <= split(v)$. Then r contains q iff $q_x \in [x_1, \infty)$, $q_y \in [y_1, y_2]$ and $q_z \in [z_1, z_2]$.*

2. *Suppose the query point $q(q_x, q_y, q_z)$ lies to the right of $split(v)$, i.e., $q_x > split(v)$. Then r contains q iff $q_x \in (-\infty, x_2]$, $q_y \in [y_1, y_2]$ and $q_z \in [z_1, z_2]$.*

Consider a node $v \in IT$. To handle the case where the query point q lies to the right of $split(v)$, we build a structure IT_v^r: Each $(4+t)$-sided rectangle $r = [x_1, x_2] \times [y_1, y_2] \times [z_1, z_2] \in S_v$ is mapped into a $(4+t-1)$-sided rectangle $(-\infty, x_2] \times [y_1, y_2] \times [z_1, z_2]$. Using Observation 1, based on these newly mapped $(d+t-1)$-rectangles we build a

structure to handle $OPEQ$. A similar structure IT_v^l is built to handle the case where the query point q lies to the left of $split(v)$. Given a query point $q \in \mathbb{R}^3$, we visit a path from root to leaf node in IT containing q_x. At each node v in the path, depending on whether q is to the left or right of v we issue an $OPEQ$ on IT_v^l or IT_v^r, respectively, to report the rectangles in $S_v \cap q$.

Theorem 3.4.1. *$OPEQ$ on 4-,5-,6-sided rectangles can be answered using a structure of $O(n)$ size and in $O(\log^2 n+k)$, $O(\log^3 n+k)$ and $O(\log^4 n+k)$ query time, respectively.*

Proof. When $t = 0$ ($OPEQ$ on 4-sided rectangles), the secondary structures IT_v^l and IT_v^r will be the $OPEQ$ structure for 3-sided rectangles. Clearly, the space occupied by the entire structure will be $O(n)$. For a given query, at most $O(\log n)$ nodes in IT are visited and hence, the query time will be $O(\log^2 n + k)$. Now, it can be easily seen that when $t = 1$ and $t = 2$, the space remains $O(n)$ but the query time increases to $O(\log^3 n + k)$ and $O(\log^4 n + k)$, respectively. □

3.5 OPEQ for 4-sided rectangles: almost optimal query time

In this section we present a proof for the following result.

Theorem 3.5.1. *Orthogonal point enclosure query on 4-sided rectangles can be answered using a structure of $O(n \log^* n)$ size and in $O(\log n \cdot \log^* n + k)$ query time.*

3.5.1 Shallow cuttings

Given two points p and q in $\mathbb{R}^d$, we say p dominates q if p has a larger coordinate value than q in every dimension. Let P be a set of n three-dimensional points. A shallow cutting for the t-level of P gives a point set R with the following properties: (i) $|R| = O(n/t)$, (ii) Any 3-d point p that is dominated by at most t points of P dominates a point in R, (iii) Each point in R is dominated by $O(t)$ points of P. The existence of such shallow cuttings has been shown by Afshani [35]. Next we state a lemma which will help us use shallow cuttings efficiently in our data structure. This is a modification of a construction by Makris and Tsakalidis [63].

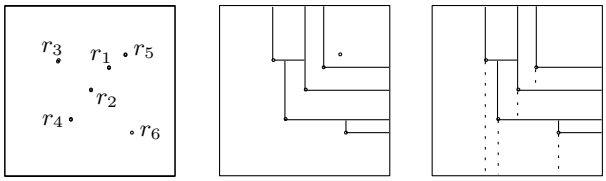


Figure 3.2: (a) Projection of points in R onto the xy-plane. (b) Region r'_i associated with each point. (c) Trapezoidal decomposition to obtain the subdivision $\mathcal{A}$.

Lemma 3.5.1. *Let R be a set of points in $\mathbb{R}^3$. Choose a strip $\mathcal{R}$ in the plane (i.e., the first two dimensions of $\mathbb{R}^3$) such that the projection of all the points of R onto the plane lie inside it. One can construct a subdivision $\mathcal{A}$ of the strip $\mathcal{R}$ into $O(|R|)$ smaller orthogonal rectangles such that for any given query point $q(q_x, q_y, q_z)$ in $\mathbb{R}^3$, if we find the rectangle in $\mathcal{A}$ that contains the projection $q(q_x, q_y)$, then it is possible to find a point of R that is dominated by q or conclude that none of the points in R are dominated by q.*

Proof. Let $r_1, r_2, \ldots, r_{|R|}$ be the sequence of points of R in non-decreasing order of their z-coordinate values. Each point $r_i(r_x, r_y, r_z)$ is projected onto the plane and a region r'_i is associated with it: $r'_i = ([r_x, \infty) \times [r_y, \infty)) \setminus \bigcup_{j=1}^{i-1} r'_j$. See Figure 3.2(a). Note that r'_i will be an empty set iff the point r_i dominates any other point in R. See Figure 3.2(b); r'_5 is an empty set. We shall discard all such points from the set R. Next we perform a trapezoidal decomposition of the strip $\mathcal{R}$ to obtain our subdivision $\mathcal{A}$, i.e., we shoot rays towards $y = -\infty$ from every remaining point in R till it hits an edge or the boundary of the strip. See Figure 3.2(c). It is easy to see that the number of rectangles in the subdivision will be $O(|R|)$.

Given a query point q, we perform a point location query on the subdivision $\mathcal{A}$. Two cases arise: (a) None of the r'_is contain q. It means none of the points in R are dominated by q. (b) Let r_i be the point associated with the rectangle which contains q. Note that among the points of R which are dominated by q in the plane, r_i has the smallest z-coordinate value. If the z-coordinate of r_i is smaller than q_z, then we have found a point in R that is dominated by q; else we can conclude that none of the points in R are dominated by q. □

3.5.2 Handling a special case

We start the presentation of our solution by first handling a special case of a set S of n 4-sided rectangles all of which cross the hyperplane $x = x^*$. We shall establish the following lemma.

Lemma 3.5.2. *Given a set S of n 4-sided rectangles all of which cross the plane $x = x^*$, we wish to answer the orthogonal point enclosure query. The space occupied is $O(n \log^* n)$ and excluding the time taken to query the point location data structure, the query time is $O(\log^* n + k)$.*

We discuss the case where q is to the right of $x = x^*$. From Observation 1, a rectangle $r = [x_1, x_2] \times (-\infty, y] \times (-\infty, z]$ is reported iff $x_2 \geq q_x$, $y \geq q_y$ and $z \geq q_z$, i.e., (x_2, y, z) dominates (q_x, q_y, q_z). Each rectangle in S is converted into a point (x_2, y, z). Call this new point set P. (The case where q is to the left of $x = x^*$ is handled symmetrically.)

The key idea here is to compute a shallow cutting for the $\log^{(i)} n$-level[1] of P to obtain a point set R_i, $\forall 0 \leq i \leq \log^* n$. For each point $p \in R_i$, based on the points of P which dominate it, build its *local structure* which is the optimal three-dimensional dominance reporting structure of Afshani [35]. Next, using Lemma 3.5.1 compute an arrangement $\mathcal{A}_i$ based on the point set R_i, $\forall 0 \leq i \leq \log^* n$. Finally, collect all the rectangles in the arrangements $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_{\log^* n}$ and construct the optimal structure of Chazelle [22] which can answer the orthogonal point enclosure query in $\mathbb{R}^2$. Call it a *global structure*.

For a given R_i, $|R_i| = O(n/\log^{(i)} n)$. *Local structure* of a point in R_i is built on $O(\log^{(i)} n)$ points of P and hence, occupies $O(\log^{(i)} n)$ space. Overall space occupied by the local structures of all the points in R_i will be $O(n)$. The total space occupied by all the local structures corresponding to $R_0, R_1, \ldots, R_{\log^* n}$ will be $O(n \log^* n)$. The number of rectangles in the arrangement $\mathcal{A}_i$ will be $O(n/\log^{(i)} n)$ and hence, the total number of rectangles in the arrangements $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_{\log^* n}$ will be $O(n)$. As the structure of Chazelle [22] uses space linear to the number of rectangles it is built on, the global structure will occupy only $O(n)$ space.

Given a query point q, if we succeed to find a point p from some point set R_i which is dominated by q in $\mathbb{R}^3$, then it is sufficient to query the local structure of p and be

[1] $\log^{(0)} n = n$ and $\log^{(i)} n = \log(\log^{(i-1)} n)$. $\log^* n$ is the smallest value of i s.t. $\log^{(i)} n \leq 1$.

done. Also, it is desirable that the size of the local structure of p is as small as possible. Therefore, our objective is to find the largest i s.t. there is a point p in R_i which is dominated by q. For this, we query the global structure with (q_x, q_y), which will report exactly one rectangle from each $\mathcal{A}_i, \forall 1 \leq i \leq \log^* n$. Scan the reported rectangles to find that rectangle whose corresponding point satisfies our objective. Once such a point $p \in R_i$ has been found, we ask a three-dimensional dominance reporting query on the local structure of p and finish the algorithm.

The time taken to perform the query on the global structure is $O(\log n + \log^* n) = O(\log n)$, where $\log^* n$ is the number of rectangles reported. Scanning the reported rectangles to find an appropriate point $p \in R_i$ takes $O(\log^* n)$ time. If $i < \log^* n$, then $k = \Omega(\log^{(i+1)} n)$, since there is no point in R_{i+1} which is dominated by q; then, querying the local structure of p takes $O(\log\log^{(i)} n + k) = O(\log^{(i+1)} n + k) = O(k)$ time. Else, if $i = \log^* n$, then querying the local structure of p takes $O(\log\log^{(\log^* n)} n + k) = O(1 + k) = O(1)$ time, since $k = O(1)$. Therefore, excluding the time taken to query the global structure, the query time is $O(\log^* n + k)$.

3.5.3 $O(\log n \cdot \log^* n + k)$-query time solution

Making use of the solution for the special case above, we present a solution for orthogonal point enclosure on 4-sided rectangles which uses $O(n \log^* n)$ space and $O(\log n \log^* n + k)$ query time. As done before, we first build an interval tree IT based on the projections of the rectangles of S on the x-axis. We shall focus on the case of reporting rectangles at those nodes v where q is to the right of $split(v)$. The symmetrical case can be similarly handled. At each node v, based on rectangles S_v, construct the *local structure* as described in section 3.5.2. The crucial technical aspect to take care while constructing the local structure is the following: The arrangements $\mathcal{A}_{(\cdot)}$ are constructed using Lemma 3.5.1, which requires as input a strip $\mathcal{R}$. For a node v with $range(v) = [x_l, x_r]$, the strip $\mathcal{R}$ will be $[split(v), x_r] \times (-\infty, +\infty)$.

Finally, we collect all the rectangles in arrangements $\mathcal{A}_0, \mathcal{A}_1, \ldots$ from all the nodes in IT and construct the optimal structure of Chazelle [22], which can answer the orthogonal point enclosure query in $\mathbb{R}^2$. This is our actual *global structure*.

Space Analysis: From section 3.5.2, the space occupied by the local structure at node v will be $O(|S_v| \log^* |S_v|)$ and the total space occupied by all the local structures

in IT will be $O(n \log^* n)$. The number of rectangles in the arrangements constructed at node v will be $O(|S_v|)$ and the total number of rectangles collected from all the nodes in IT will be $O(n)$. Therefore, the global structure will occupy $O(n)$ space. The total space occupied by our data structure will be $O(n \log^* n)$.

Given a query point q, let Π be the path from root to the leaf node containing q_x. We query the global structure to report all the rectangles containing (q_x, q_y). The crucial observation is that by our choice of strip $\mathcal{R}$ for each node in IT, if a node v doesn't lie on Π, then no rectangle corresponding to v will be reported. On the other hand, if a node v lies on Π, then exactly $\log^* |S_v|$ rectangles will be reported. To report the rectangles in $S_v \cap q$, we follow the query algorithm discussed in section 3.5.2. Repeating this procedure at every node on Π will report all the rectangles in $S \cap q$.

Query Analysis: Querying the global structure takes $O(\log n + \log n \cdot \log^* n)$ time, since $O(\log^* n)$ rectangles will be reported from each node on Π. From the analysis in section 3.5.2, the time taken to report rectangles in $S_v \cap q$, at each node v, will be $O(\log^* n + |S_v \cap q|)$. Therefore, the overall query time will be $O(\log n \cdot \log^* n + k)$. This finishes the proof of Theorem 3.5.1.

Remark 1: To answer $O(\log n)$ point location queries simultaneously, one would expect that an extra dimension on the rectangles is needed to capture their corresponding node in the interval tree. By suitably modifying the construction of Makris and Tsakalidis (with the introduction of the concept of "strip $\mathcal{R}$"), we are able to simultaneously solve multiple point location queries while staying with $OPEQ$ in $\mathbb{R}^2$. We believe this is a novel idea.

Remark 2: To obtain Theorem 3.5.1, we computed a shallow cutting for the $\log^{(i)} n$-level of P to obtain a point set R_i, $\forall 0 \leq i \leq \log^* n$. Instead, suppose we compute a shallow cutting for the $\log^{(i)} n$-level of P to obtain a point set R_i, $\forall 0 \leq i \leq c$, for any integer constant $c \geq 1$. Then, it can verified that one can obtain a data structure with space $O(n)$ and query time $O(\log n \cdot \log^{(c+1)} n + k)$.

3.6 OPEQ for 4-sided rectangles: optimal query time

In the above mentioned solution, we obtain $O(\log n \cdot \log^* n + k)$ query time, since $|\Pi| = O(\log n)$ and the time taken to report $S_v \cap q$ at each node $v \in \Pi$ is $O(\log^* n + |S_v \cap q|)$. One way to obtain a query time of $O(\log n + k)$ is by restricting $|\Pi| = O(\log n / \log^* n)$; indeed this can be achieved by decreasing the height of the interval tree to $O(\log n / \log f) = O(\log n / \log^* n)$ and increasing the fanout to $f = 2^{\log^* n}$. However, we now have to handle the "middle" structure for which we use some additional technical and novel ideas. We note that this idea of increasing the fanout of an interval tree has been used in the past [64, 65] for some aggregate problems involving rectangles; but our handling of the middle structure is completely different. The key observation to handle middle structure is that since the space is already $\log^* n$ factor away from linear, we can afford to "move" the $\log^* n$ factor from the query time to an additive term in the space complexity.

Skeleton structure: Construct an interval tree IT with fanout $f = 2^{\log^* n}$. The rectangles of S are projected onto the x-axis (each rectangle gets projected into an interval). Let E be the set of $2n$ endpoints of these projected intervals. Divide the x-axis into $2n$ vertical slabs such that each slab covers exactly 1 point of E. Create an f-ary tree IT on these slabs, each of which corresponds to a leaf node in IT. For each node $v \in IT$, we define its range on the x-axis, $range(v)$. If v is a leaf node, then $range(v)$ is the portion of the x-axis, occupied by the slab corresponding to the leaf; else if v is an internal node, then $range(v)$ is the union of the ranges of its children $v_1, v_2, \ldots, v_f$, i.e., $range(v) = \bigcup_{i=1}^{f} range(v_i) = [x_l, x_r]$. For each internal node $v \in IT$, we also define $f + 1$ boundary slabs $b_1(v), b_2(v), \ldots, b_{f+1}(v)$: $b_1(v) = x_l, b_{f+1} = x_r$ and $\forall 1 < i < f + 1$, b_i is the boundary separating $range(v_{i-1})$ and $range(v_i)$. Consider a rectangle $r = [x_1, x_2] \times (-\infty, y] \times (-\infty, z] \in S$. Rectangle r is assigned to an internal node $v \in IT$, if the interval $[x_1, x_2]$ crosses one of the slab boundaries of v but doesn't cross any of the slab boundaries of parent of v. See Figure 3 for an example of an internal node v in the interval tree. Let S_v be the set of rectangles of S associated with node v.

Each rectangle in S is broken into three disjoint rectangles as follows: Consider a rectangle $r = [x_1, x_2] \times (-\infty, y] \times (-\infty, z] \in S_v$. Let x_1 lie in $range(v_i)$ and x_2 lie in

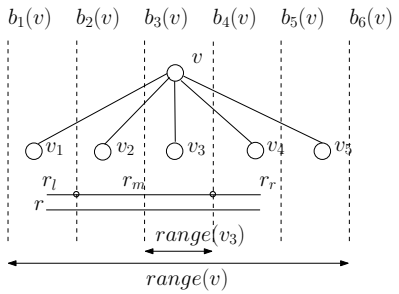


Figure 3.3: An internal node v in the interval tree IT.

$range(v_j)$. Then r is broken into a *left rectangle* $r_l = [x_1, b_{i+1}(v)) \times (-\infty, y] \times (-\infty, z]$, a *middle rectangle* $r_m = [b_{i+1}(v), b_j(v)] \times (-\infty, y] \times (-\infty, z]$ and a *right rectangle* $r_r = (b_j(v), x_2] \times (-\infty, y] \times (-\infty, z]$. Note that if $j = i + 1$, then we will only have a left and a right rectangle. Define S_v^l, S_v^m and S_v^r to be the set of left, middle and right rectangles obtained by breaking S_v. Furthermore, let $S^l = \bigcup_{v \in IT} S_v^l$, $S^m = \bigcup_{v \in IT} S_v^m$ and $S^r = \bigcup_{v \in IT} S_v^r$ be the sets of all left, middle and right rectangles, respectively. Note that it suffices to build separate data structures to handle S^l, S^m and S^r.

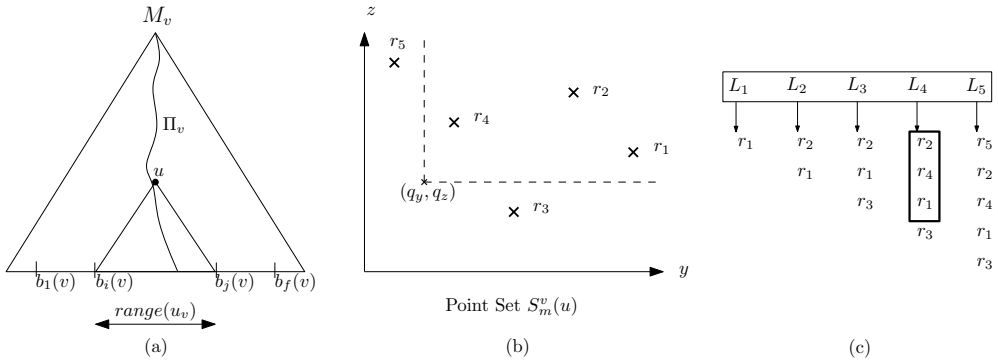


Figure 3.4: (a) M_v Structure. (b) Querying point set $S_v^m(u)$ with (q_y, q_z). (c) Lists L_i. For the example query in (b), we walk down the list L_4 to report r_2, r_4 and r_1.

The solution built in section 3.5 can easily be adapted to report $S^l \cap q$ and $S^r \cap q$ in $O(\log n + k)$ query time, since the height of the tree is now $O(\log n / \log^* n)$. The remaining part of this subsection will focus on building a suitable data structure(s) to report $S^m \cap q$ in $O(\log n + k)$ query time.

Local structure M_v: At each node $v \in IT$, we store a local structure M_v based on

the rectangles S_v^m. We first describe construction of M_v and how to query it to report $S_v^m \cap q$. Then, we shall describe the global structure and the global query algorithm to report $S^m \cap q$. We shall utilize the fact that the endpoints of the x-projections of S_v^m comes from a fixed universe $[1 : f] = [1 : 2^{\log^* n}]$. The primary structure of M_v is a segment tree [58] built on the x-projections of S_v^m. For each node $u \in M_v$, define $S_v^m(u)$ to be the rectangles whose x-projection was associated with node u. Also, associate $range(u_v)$ with each node $u \in M_v$, where $range(u_v) \subseteq range(v)$ is the span of boundary slabs covered by the leaf nodes in the subtree of u (see Figure 3.4(a)). The height of M_v will be $O(\log 2^{\log^* n}) = O(\log^* n)$ and, therefore, the space occupied by M_v will be $O(|S_v^m| \log^* n)$.

Given a query point $q(q_x, q_y, q_z)$, we trace a path Π_v in M_v from the root to the leaf node using q_x. At each node $u \in \Pi_v$, we need to report a rectangle $r \in S_v^m(u)$ iff yz-projection of r contains (q_y, q_z), i.e., $q_y \leq y$ and $q_z \leq z$ (a $2d$-dominance query). Unfortunately, using standard structures such as a priority search tree [58] will not help us to achieve our desired query time.

Instead, we shall build the following structure on the yz-projections of $S_v^m(u)$: Convert each rectangle $r \in S_v^m(u)$ into a new point (y, z) and the query q into a query rectangle $[q_y, \infty) \times [q_z, \infty)$. For the sake of convenience, we shall refer to the new point set as $S_v^m(u)$ itself. Sort the point set $S_v^m(u)$ in non-increasing order of their y-coordinate values. For simplicity, we will still refer to the sorted list as $S_v^m(u)$ itself. Add a dummy point at the end of the list with $y = -\infty$ and an arbitrary value of z. With the i^{th} element in $S_v^m(u)$, we store a list L_i which is the $1^{st}, 2^{nd}, \ldots, i^{th}$ element of $S_v^m(u)$ in non-increasing order of their z-coordinate values (see Figure 3.4 (b) & (c)). Then the total size of all the lists $L_i, \forall 1 \leq i \leq |S_v^m(u)|$ will be $O(|S_v^m(u)|^2)$. However, notice that given two consecutive elements i and $i+1$ in $S_v^m(u)$, L_{i+1} can be obtained from L_i by making $O(1)$ changes. Now treating the y-coordinate as time, we store all the lists $L_i, \forall 1 \leq i \leq |S_v^m(u)|$, in a partially persistent structure [20]. The total number of memory modifications will be $O(|S_v^m(u)|)$ and hence, the total size of all the lists reduces from $O(|S_v^m(u)|^2)$ to $O(|S_v^m(u)|)$. To answer the query at node u, we locate the element i in $S_v^m(u)$ which is the predecessor of q_y. Then, we walk down the list L_i to report the corresponding rectangles till either the list gets exhausted or we reach a

point whose z-coordinate is less than q_z. Ignoring the time taken to locate element i in $S_v^m(u)$, the time spent at node u will be $O(1 + |S_v^m(u) \cap q|)$. The performance of the local structure M_v is summarized next.

Lemma 3.6.1. *Given a set S_v^m of 4-sided rectangles, we wish to answer the orthogonal point enclosure query. The endpoints of the x-projections of S_v^m come from a fixed universe $[1 : f] = [1 : 2^{\log^* n}]$. Local structure M_v occupies $O(|S_v^m| \log^* n)$ space and excluding the time taken to locate element i in $S_v^m(u), \forall u \in \Pi_v$, the query time will be $O(\log^* n + |S_v^m \cap q|)$.*

Global structure: The only missing ingredient is an efficient technique to locate element i in $S_v^m(u)$'s which are visited during a query. Our technique will be based on the following simple yet powerful observation.

Observation 2. *For a given query $q(q_x, q_y, q_z)$, let the i^{th} element in point set $S_v^m(u)$ be the predecessor of q_y. Then we walk down the list L_i in $S_v^m(u)$ iff (i) $q_x \in range(u_v)$, and (ii) $q_y \in (y_{i+1}, y_i]$, where y_i and y_{i+1} are the y-coordinates of the i^{th} and $(i+1)^{th}$ entry in point set $S_v^m(u)$.*

Using the above observation, at every node $u \in M_v$, the i^{th} element in $S_v^m(u), \forall 1 \leq i \leq |S_v^m(u)|$, is mapped to a rectangle $range(u_v) \times (y_{i+1}, y_i]$ in $\mathbb{R}^2$. This process is repeated at every M_v structure in IT. Collect all the newly mapped rectangles and construct the optimal structure of Chazelle [22] which can answer $OPEQ$ in $\mathbb{R}^2$. This is our *global structure.*

Space Analysis: Since each local structure M_v occupies $O(|S_v^m| \log^* n)$ space, the overall space occupied by all the local structures in IT will be $O(n \log^* n)$. The number of rectangles mapped from all the nodes in M_v is $O(|S_v^m| \log^* n)$ and hence, the total number of rectangles collected to construct the global structure is $O(n \log^* n)$. Therefore, the global structure occupies $O(n \log^* n)$ space.

Given a query point q, we query the global structure with (q_x, q_y). From Observation 2 and our construction it is guaranteed that a rectangle $range(u_v) \times (y_{i+1}, y_i]$ is reported iff the i^{th} element in $S_v^m(u)$ is the predecessor of q_y. Then for each reported rectangle we go to its corresponding list L_i and report the rectangles in $S_v^m(u) \cap q$. This ensures that all the rectangles in $S^m \cap q$ get reported.

Query Analysis: Querying the global structure takes $O(\log n)$ time, since rectangles corresponding to $O(\log^* n)$ nodes from each of the $O(\log n/\log^* n)$ M_v structures are reported. Adding the time spent at each of the local M_v structures, we get $O(\sum_{v\in\Pi}(\log^* n + |S_v^m \cap q|)) = O(\log n + k)$. Overall query time to report $S^m \cap q$ will be $O(\log n + k)$. The final result is summarized below.

Theorem 3.6.1. *Orthogonal point enclosure query on 4-sided rectangles can be answered using a structure of $O(n \log^* n)$ size and in $O(\log n + k)$ query time.*

3.7 OPEQ on 5- and 6-sided rectangles

In this section, we use the result obtained for $OPEQ$ on 4-sided rectangles to answer $OPEQ$ on 5- and 6-sided rectangles. First, we present a solution for 5-sided rectangles. Alstrup, Brodal, and Rauhe introduced the grid-based technique [66] to index points for answering orthogonal range reporting queries. We also use their grid-based technique, but suitably adapt it for handling the indexing of 5-sided rectangles. At a high level, the query algorithm is based on the following approach: Theorem 3.4.1 handles $OPEQ$ on 5-sided rectangles in $O(\log^3 n + k)$ query time. When $k \geq \log^3 n$, Theorem 3.4.1 will have a query time $O(k)$, which is good. When $k < \log^3 n$, we can no longer use Theorem 3.4.1 but the low-output size allows us to pre-compute partial answers to each query.

Structure: Define a parameter $t = \log^4 n$. Consider the projection of the rectangles of S on to the xy-plane and impose an orthogonal $(2\sqrt{\frac{n}{t}}) \times (2\sqrt{\frac{n}{t}})$ grid such that each horizontal and vertical slab contains the projections of $\sqrt{nt}$ sides. Let $S_{root} \subseteq S$ be the set of rectangles stored at the root. A rectangle of S belongs to S_{root} iff it intersects at least one horizontal or vertical boundary of the grid. A couple of data structures are built on S_{root} which will be discussed below. Call this the root of the recursion tree. Finally, we recurse on the rectangles which lie completely inside a slab. At each node of the recursion tree, if we have m rectangles in the subproblem then the value then the value of t changes to $\log^4 m$ and the grid size changes to $(2\sqrt{\frac{m}{t}}) \times (2\sqrt{\frac{m}{t}})$. We stop the recursion when a subproblem has less than c rectangles, for a suitably large constant c.

A *grid structure*, a *slow structure* and *side structure*'s are built on S_{root}. The *slow structure* is Theorem 3.4.1 built on 5-sided rectangles S_{root}. The slow structure is queried

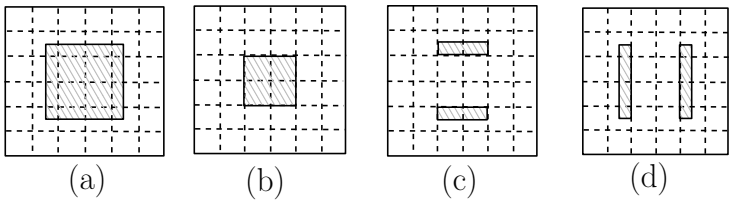


Figure 3.5: Breaking a rectangle in (a) into 2 horizontal side rectangles (shown in (c)) and 2 vertical side rectangles (shown in (d)).

only when $|S_{root} \cap q| = \Omega(\log^3 n)$. A rectangle r' is *higher* than rectangle r'' if r' has a larger span than r'' along z-direction. In the *grid structure*, for each cell c of the grid, among the rectangles which completely cover c, store the highest $\log^3 n$ rectangles in a linked list L_c in decreasing order of their z-coordinates. As shown in Figure 3.5, each rectangle in S_{root} is broken into at most 4 *side rectangles*. Observe that the side rectangles are 4-sided rectangles. For each row and column slab, we have a *side structure* which is Theorem 3.5.1 built on the side rectangles lying inside it.

Space Analysis: The space occupied by the slow structure and the side structures is $O(|S_{root}|)$ and $O(|S_{root}| \log^* |S_{root}|)$, respectively. Note that a rectangle in S is stored at exactly one node in the recursion tree. Therefore, the overall space occupied by the slow structures and the side structures in the recursion tree is $O(n)$ and $O(n \log^* n)$, respectively. The space occupied by the grid structure will be $O(n/t \cdot \log^3 n) = O(n/\log n)$. Thus the space occupied, $S(n)$, by all the grid structures in the recursion tree is given by the recurrence

$$S(n) = \sum_{i=1}^{4\sqrt{n/t}} S(n_i) + O\left(\frac{n}{\log n}\right), \forall i, n_i \leq \sqrt{nt}.$$

This solves to $S(n) = O(n)$. Therefore, the overall space occupied by the data structure will be $O(n \log^* n)$.

Query: Given a query point q, at the root we locate the cell c on the grid containing q. Scan the list L_c to report rectangles till we either (a) find a rectangle which doesn't contain q, or (b) the end of the list is reached. If case (b) happens, then we have reported $\log^3 n$ rectangles, so we query the slow structure to report $S_{root} \cap q$. If case (a) happens, then we also query the side structures of the horizontal and the vertical slab containing

q. Next, we recursively query the horizontal and the vertical slab containing q.

Query Analysis: First we analyze the query time at the root of the recursion tree. Cell c on the grid can be located in $O(\log \sqrt{n/t}) = O(\log n)$ time. If case (a) happens, then the time spent is $O(\log n + |S_{root} \cap q|)$. Else, if case (b) happens, then the time spent is $O(\log^3 n + |S_{root} \cap q|) = O(|S_{root} \cap q|)$, since $|S_{root} \cap q| \geq \log^3 n$. Therefore, the query time at the root is $O(\log n + |S_{root} \cap q|)$. Let $Q(n)$ denote the overall query time (excluding the output portion). Then

$$Q(n) = 2Q(\sqrt{nt}) + O(\log n), t = \log^4 n.$$

This solves to $Q(n) = O(\log n \cdot \log\log n)$. Therefore, the overall query time will be $O(\log n \log\log n + k)$.

Theorem 3.7.1. *Orthogonal point enclosure query on* 5*-sided rectangles can be answered using a structure of* $O(n \log^* n)$ *size and in* $O(\log n \cdot \log\log n + k)$ *query time.*

Now we look at $OPEQ$ for 6-sided rectangles. In Theorem 3.4.1, $OPEQ$ for 6-sided rectangles was handled by placing at each node of the interval tree a data structure which can handle $OPEQ$ for 5-sided rectangles. Now placing the data structure of Theorem 3.7.1 at each node of the interval tree leads to the following result.

Theorem 3.7.2. *Orthogonal point enclosure query on* 6*-sided rectangles can be answered using a structure of* $O(n \log^* n)$ *size and in* $O(\log^2 n \cdot \log\log n + k)$ *query time.*

By using segment trees, the above result extends to higher dimensions as well. (We omit the details.)

Theorem 3.7.3. *Orthogonal point enclosure query on* $2d$*-sided rectangles in* $\mathbb{R}^d (d \geq 3)$ *can be answered using a structure of* $O(n \log^{d-3} n \cdot \log^* n)$ *size and in* $O(\log^{d-1} n \cdot \log\log n + k)$ *query time.*

3.8 Open problems

We conclude with some open problems in the pointer machine model. As of now, an optimal solution in $\mathbb{R}^3$ is known only for 3-sided rectangles. Is it possible to answer

$OPEQ$ for 4-sided rectangles in $\mathbb{R}^3$ in $O(\log n + k)$ query time using an $O(n)$ space structure? More interestingly, what is the right bound to target for $OPEQ$ for 5-sided rectangles: Is there a linear-space structure which answers the query in $O(\log n + k)$ time? Or is there a lower bound of $\Omega(\log n \cdot \log\log n + k)$ on the query time for a linear-space structure?

Part II

Top-k Geometric Intersection Query

Chapter 4

Top-k Geometric Intersection Query (GIQ)

4.1 Problem Statement

Top-k GIQ: We are given a set $\mathcal{A} = \{a_1, \ldots, a_n\}$ of n geometric objects in $\mathbb{R}^d$ ($d \geq 1$), where a_i has a real-valued weight w_i, $1 \leq i \leq n$. We wish to organize $\mathcal{A}$ into a space-efficient data structure so that for any query pair (q, k), where q is a geometric object and $k > 0$ is an integer, we can report efficiently the k largest-weight objects of $\mathcal{A}$ that are intersected by q. (Two geometric objects in $\mathbb{R}^d$ *intersect* iff they have a point in common.)

More precisely, let $\mathcal{A}(q)$ be the set of objects in $\mathcal{A}$ that are intersected by q. Then we wish to find and report the objects in a set $\mathcal{A}_k(q) \subseteq \mathcal{A}(q)$ such that $|\mathcal{A}_k(q)| = k$ and for any $a_i \in \mathcal{A}_k(q)$ and any $a_j \in \mathcal{A}(q) \setminus \mathcal{A}_k(q)$, we have $w_i \geq w_j$. (Note that if q intersects k or fewer objects of $\mathcal{A}$ then we simply report all of them.)

We recollect some definitions. In a *reporting GIQ* we report $\mathcal{A}(q)$, in a *counting GIQ* we report $|\mathcal{A}(q)|$ and in a *max GIQ* we report the object in $\mathcal{A}(q)$ with the largest-weight.

4.2 Naïve solutions

As a warm-up, we discuss two naïve solutions for answering any top-k GIQ:

1. First, build a *reporting* structure based on the objects in $\mathcal{A}$ (disregarding their weights). Given the query pair (q, k), query the structure with q to report all the objects of $\mathcal{A}$ which intersect q. Next, run a standard selection algorithm [67] on $\mathcal{A}(q)$ to identify the object with the kth-largest weight. Finally, scan $\mathcal{A}(q)$ to output the k largest-weight objects. Clearly, this approach is not efficient if $|\mathcal{A}(q)| \gg k$, which is often the case.

2. Second, in the preprocessing phase, sort all the objects in $\mathcal{A}$ in non-increasing order of their weights and keeps them in an array. Given a query pair (q, k), the array is scanned from the beginning till either (i) k objects of $\mathcal{A}$ intersecting q are found, or (ii) the end of the array is reached. In this approach the query time can be as bad as $\Theta(n)$.

4.3 Key features of our techniques/reductions

In this report we present three different techniques (or reductions) to efficiently solve the top-k GIQ problems. The key features of our techniques are the following:

1. **Generic reductions.** Our reductions are aimed at solving *any* top-k GIQ problem efficiently. All the three reductions require efficient solutions for the corresponding reporting, or counting or max GIQ, which is often the case. Such general reductions did not exist in the literature before.

2. **Strong theoretical guarantees.** As will be shown later, our reductions also have attractive space, query time and update time bounds. Roughly speaking, there is either no deterioration or very little deterioration in the theoretical bounds for the top-k GIQ problem w.r.t. their corresponding reporting, counting and max GIQ.

3. **Output sensitivity.** A trivial observation is that the query time for any top-k GIQ is greater than or equal to k time-units (since one needs k time-units to report k objects). Therefore, an important feature for a top-k data structure is that they have query time that is sensitive to k, typically of the form $O(f(n)+k)$ or $O(f(n) + k \cdot g(n))$, where $f(n)$ and $g(n)$ are "small" (e.g., polylogarithmic).

This is a very desirable property, since it allows the query to be answered faster when k is small. Our data structures in this thesis achieve this property. The naïve solutions discussed above do not have this property; even for $k = 1$ they can have a query time of $\Theta(n)$.

4. **Ease of implementation.** There is very little overhead involved in implementing these techniques. Given data structures for the reporting and the counting GIQ problem (which typically exist for many GIQ problems), our first technique involves merely implementing a suitable binary search tree. We have demonstrated this in [4] for two problems: orthogonal range search and rectangle stabbing. In our second and third technique, one simply has to build a data structure for reporting and/or max GIQ problem on the entire dataset or on a collection of random samples from the dataset.

4.4 Three Generic Reductions

In this thesis we present three generic reductions to answer a top-k GIQ. Each generic reduction reduces the task of answering a top-k GIQ to a small number of queries on the companion problems of that GIQ, which typically have a small space and query time bounds.

4.4.1 Mathematical definitions

We (i) define $\log^*(n)$ as the number of times that we need to perform $\log_2(\cdot)$ on n to get a value no more than 1 (see also Chapter 3 for a precise definition), and (ii) use notation $\tilde{O}(.)$ to hide a factor polylogarithmic in n when such a factor is insignificant.

Finally, we say that a function $f(n)$ is *geometrically converging* if it satisfies two conditions:

- For any $n \geq B$: $\sum_{i=0}^{h} f\left(\frac{n}{c^i}\right) = O(f(n))$, for any value $c \geq 2$, where h is the largest integer i satisfying $n/c^i \geq B$.
- For any $n < B$, $f(n) = O(1)$.

4.4.2 First reduction: Using counting and reporting structure

Our first approach requires that efficient solutions to the reporting and the counting versions of the underlying GIQ problem be available. Roughly speaking, our technique incurs only a logarithmic increase in the space, the query time, and the update time over the corresponding bounds for the underlying GIQ problem. Specifically, we establish the following:

Theorem 4.4.1. *Suppose that there is*

- *A reporting structure of $\mathcal{S}_{rep}(n)$ space that answers a query in $\mathcal{Q}_{rep}(n)+O(t)$ time;*
- *A counting structure of $\mathcal{S}_{cnt}(n)$ space that answers a query in $\mathcal{Q}_{cnt}(n)$ time.*

Assume that $\mathcal{S}_{cnt}(n)/n$, $\mathcal{S}_{rep}(n)/n$, $\mathcal{Q}_{cnt}(n)$ and $\mathcal{Q}_{rep}(n)$ are non-decreasing functions for non-negative values of n.

Then there is a top-k structure of space $\mathcal{S}_{top}(n)$ and query time $\mathcal{Q}_{top}(n)+O(k)$ with

$$\mathcal{S}_{top}(n) = O((\mathcal{S}_{rep}(n) + \mathcal{S}_{cnt}(n)) \log_2 n) \tag{4.1}$$

$$\mathcal{Q}_{top}(n) = O((\mathcal{Q}_{rep}(n) + \mathcal{Q}_{cnt}(n)) \log_2 n) \tag{4.2}$$

Furthermore, if the reporting and the counting structure support updates in $\mathcal{U}_{rep}(n)$ and $\mathcal{U}_{cnt}(n)$ time, respectively, then the top-k structure supports updates in $\mathcal{U}_{top}(n) = O((\mathcal{U}_{rep}(n) + \mathcal{U}_{cnt}(n)) \log_2 n)$ amortized time.

Remark. The above reduction is presented in the RAM model. The following two reductions will be presented in the EM model. By setting M and B to appropriate constants, all the EM results also hold in the RAM model.

4.4.3 Second reduction: Using max and prioritized reporting structure

Our second approach requires that efficient solutions to the *max* and the *prioritized reporting* versions of the underlying GIQ problem be available. We start with the following definition.

Prioritized Reporting: Given a query q and a real value τ, this query reports all the objects in $\mathcal{A}(q)$ with weight greater than or equal to τ.

Before presenting the reduction, we present the following interesting fact: *prioritized reporting can be reduced to top-k reporting.* The formal statement is the following:

Theorem 4.4.2. *Suppose that there is a structure that consumes $\mathcal{S}_{top}(n)$ space on n elements, and answers a top-k query in $\mathcal{Q}_{top}(n) + O(k/B)$ I/Os. Then, there is a prioritized-reporting structure of $\mathcal{S}_{pri}(n)$ space that answers a query in $\mathcal{Q}_{pri}(n)+O(t/B)$ I/Os—where t is the number of reported elements—such that*

$$\begin{aligned} \mathcal{S}_{pri}(n) &= O(\mathcal{S}_{top}(n)) \\ \mathcal{Q}_{pri}(n) &= O(\mathcal{Q}_{top}(n)). \end{aligned}$$

The reduction does not depend on the underlying problem, i.e., prioritized reporting is no harder than top-k reporting, regardless of the type of input objects and query object. Therefore, if one does not even have a structure for the former, there is no hope for the latter.

An important special case of the top-k reporting is the *max reporting* where all queries have $k = 1$. Before solving queries of all k, one must at least be able to design an efficient structure for max reporting. In other words, just like the prioritized reporting, max reporting is also a *necessary* step towards settling the top-k reporting. Our second reduction shows that the two necessary structures are *sufficient* as well:

Theorem 4.4.3. *Suppose that there is*

- *A prioritized-reporting structure of $\mathcal{S}_{pri}(n)$ space that answers a query in $\mathcal{Q}_{pri}(n)+O(t/B)$ I/Os;*
- *A max-reporting structure of $\mathcal{S}_{max}(n)$ space that answers a query (i.e., $k = 1$) in $\mathcal{Q}_{max}(n)$ I/Os. It is required that $\mathcal{S}_{max}(n)$ is geometrically converging.*

Then, there is a top-k structure of expected space $\mathcal{S}_{top}(n)$ and expected query time $\mathcal{Q}_{top}(n) + O(k/B)$ with

$$\mathcal{S}_{top}(n) = O\left(\mathcal{S}_{pri}(n) + \mathcal{S}_{max}\left(\frac{6n}{B \cdot Q_{pri}(n)}\right)\right) \tag{4.3}$$

$$\mathcal{Q}_{top}(n) = O\left(\mathcal{Q}_{pri}(n) + \mathcal{Q}_{max}(n)\right). \tag{4.4}$$

Furthermore, if the prioritized and the max structures support an update in $\mathcal{U}_{pri}(n)$ and $\mathcal{U}_{max}(n)$ I/Os respectively, then the top-k structure supports an update in $O(\mathcal{U}_{pri}(n)+$

$\mathcal{U}_{max}(n)$) expected I/Os. If any of $\mathcal{U}_{pri}(n)$ and $\mathcal{U}_{max}(n)$ is amortized, the update cost of the top-k structure is amortized expected. In the above, every expectation is taken over the random choices made by our algorithms.

Remark 1. The above reduction is optimal in the sense that there is no performance degradation (in expectation): the space, the query time, and the update time of the top-k structure is determined by the worse between the prioritized and the max structure.

Remark 2. Our reduction constructs the max structure on at most $O\left(\frac{6n}{B \cdot Q_{pri}(n)}\right)$ objects, and, therefore, one need not try hard to minimize the space of the max structure. In fact, $\mathcal{S}_{max}(n)$ is allowed to be larger than $\mathcal{S}_{top}(n)$. For instance, consider a scenario where $\mathcal{S}_{pri}(n) = O(n/B)$, $\mathcal{Q}_{pri}(n) = \log_B n$, and $\mathcal{S}_{max}(n) = O((n/B)\log_B n)$. Plugging in these values into Theorem 4.4.3 leads to $\mathcal{S}_{top}(n) = O(n/B)$, since $\mathcal{S}_{max}\left(\frac{n}{B \log_B n}\right) = O\left(\frac{n}{B^2}\right) = O\left(\frac{n}{B}\right)$.

4.4.4 Third reduction: Using only the prioritized reporting structure

Our third reduction requires an efficient solution only for the *prioritized reporting* version of the underlying GIQ problem. We show that, under mild conditions, there only needs to be an $O(\log_B n)$ gap in the query cost between the top-k and the prioritized reporting:

Theorem 4.4.4. *Suppose that there is a prioritized structure of $\mathcal{S}_{pri}(n)$ space and query cost $\mathcal{Q}_{pri}(n) + O(t/B)$ such that $\mathcal{S}_{pri}(n)$ is geometrically converging, and*

$$\mathcal{Q}_{pri}(n) \quad \geq \quad \log_B n.$$

Furthermore, suppose that the problem is polynomially bounded*, namely, for any input D of n elements, there are only $n^{O(1)}$ distinct outcomes for $D(q)$ over all the possible queries on D.*

Then, there is a top-k structure of space $\mathcal{S}_{top}(n)$ and query time $\mathcal{Q}_{top}(n) + O(k/B)$ with

$$\mathcal{S}_{top}(n) = O(\mathcal{S}_{pri}(n)) \tag{4.5}$$

$$\mathcal{Q}_{top}(n) = O\left(\mathcal{Q}_{pri}(n) \cdot \frac{\log n}{\log B + \log \frac{\mathcal{Q}_{pri}(n)}{\log_B n}}\right) \tag{4.6}$$

Remark 1. It is worth mentioning that most of the known reporting problems are polynomially bounded. Consider, for example, D to be a set of n points in $\mathbb{R}^2$. In the *halfspace reporting*, given a halfspace q (the region on one side of a line), we want to report the set $D(q)$ of points in $D \cap q$. It is easy to see that $O(n^2)$ different subsets $D(q)$ exist, ranging over all possible q, because there are only $\binom{n}{2}$ different lines passing through two points in D.

Remark 2. Since the denominator in Equation (4.6) is at least $\log B$, we have $\mathcal{Q}_{top}(n) = O(\mathcal{Q}_{pri}(n) \cdot \log_B n)$.

Remark 3. If $\mathcal{Q}_{pri}(n) \geq (n/B)^\epsilon$ for an arbitrarily small constant $\epsilon > 0$, (4.6) becomes $\mathcal{Q}_{top}(n) = O(\mathcal{Q}_{pri}(n))$. In other words, top-$k$ reporting is asymptotically as difficult as prioritized reporting for "hard" queries.

4.5 New top-k GIQ structures

Using the three generic reductions discussed above we have been able to design several new top-k GIQ structures. We present the bounds obtained for each top-k GIQ problem as a theorem here and the proofs of these will be discussed in Chapter 6.

Orthogonal Range Reporting. In *top-k orthogonal range reporting*, $\mathcal{A}$ is a set of weighted points in $\mathbb{R}^d$ and the query is an axes-aligned hyper-rectangle in $\mathbb{R}^d$. This problem is very relevant in spatial databases. For example, in $\mathbb{R}^2$ the points could represent restaurants and the rating of each restaurant could be its weight. The query rectangle q can be an area in New York City and the user might want to know the top-5 rated restaurants in that area. Some of the key results we obtain for this problem are as follows:

Theorem 4.5.1. *For top-k orthogonal range reporting:*

- *When $d = 1$, there is a RAM structure of $O(n)$ space and $O(\log n + k)$ query time, both in expectation.*
- *When $d = 1$, there is an EM structure of $O(\frac{n}{B})$ space and $O(\log_B n + k/B)$ query time, both in expectation.*

Theorem 4.5.2. *For top-k orthogonal range reporting:*

- *When $d = 2$, there is a RAM structure of $O(n\frac{\log n}{\log\log n})$ space and $O(\log n + k)$ query time, both in expectation.*

- *When $d = 2$, there is an EM structure of $O(\frac{n}{B}\frac{\log n}{\log\log_B n}(\log\log B)^2)$ space and $O(\log_B n + k/B)$ query time, both in expectation.*

Note that our results above for $d = 1, 2$ hold in expectation. There are results in the literature with the same bounds as above and they hold true in the worst-case as well; naturally, those solutions are technically involved. On the other hand, the solutions presented in this report have the advantage of being conceptually simpler.

Finally we present a dynamic solution for this problem in $\mathbb{R}^d$.

Theorem 4.5.3. *For top-k orthogonal range reporting:*

- *When $d \geq 1$, there is a RAM structure of $O(n \log^d n)$ space, $O(\log^{d+1} n + k)$ query time and $O(\log^{d+1} n)$ amortized update time. All the bounds hold true in the worst-case.*

Unlike in the RAM model, in the EM model no efficient dynamic data structures are known for standard reporting and counting orthogonal range searching in higher dimensions. (Of course one can always take a RAM structure and use that as an EM structure, however no structure tailor-made for the EM model is known.) Therefore, we avoid discussing top-k orthogonal range searching in the EM model in higher dimensions.

Halfspace and Circular Range Reporting. In *halfspace reporting*, $\mathcal{A}$ is a set of points in $\mathbb{R}^d$, where d is a fixed integer. The query $q = (\boldsymbol{q}, c)$ is a halfspace, i.e., all the $\boldsymbol{x}$ satisfying $\boldsymbol{x} \cdot \boldsymbol{q} \geq c$, where $\boldsymbol{q}$ and c are the query parameters ($\boldsymbol{x}$ and $\boldsymbol{q}$ are d-dimensional vectors, and c a real value). The importance of halfspace reporting—in general, searching with linear constraints—has long been recognized in the database community; e.g., see [68]. A motivating example for the top-k halfspace range reporting in $\mathbb{R}^2$ is the following: Consider a financial database storing earnings (e) and volatility (v) information for a large number of stocks, as well as information on total return (r). Thus, each stock, s, is represented as a point (s_e, s_v) in $\mathbb{R}^2$, with weight s_r. Each investor, I, has a different preference for income and risk, specified as percentages I_e and I_v. A potential investor might wish to identify, say, the ten highest-return stocks, s,

for which the weighted score $I_e \cdot s_e + I_v \cdot s_v$ is at least an investor-specified threshold t_I. The equation $I_e \cdot s_e + I_v \cdot s_v \geq t_I$ defines a halfspace in $\mathbb{R}^2$ and we are, thus, interested in the ten highest-return stocks in this halfspace. This is an instance of *top-k halfspace range search* problem in $\mathbb{R}^2$, where $k = 10$ and the query, q, is the above halfspace.

We obtain the following results for this problem:

Theorem 4.5.4. *For top-k halfspace reporting:*

- *When $d = 2$, there is a RAM structure of $O(n \log n)$ space and $O(\log n + k)$ query time, both in expectation.*
- *When $d \geq 4$, there is a RAM structure of $O(n \log n)$ space and $\tilde{O}(n^{1-1/\lfloor d/2 \rfloor}) + O(k)$ query time, both in the worst case.*
- *When $d \geq 4$, there is an EM structure of $O(n/B)$ space and $O((n/B)^{1-1/\lfloor d/2 \rfloor + \epsilon} + k/B)$ query time, both in the worst case, where $\epsilon > 0$ is an arbitrarily small constant.*

Circular range reporting is a closely related problem. Again, $\mathcal{A}$ is a set of points in $\mathbb{R}^d$, but each $q = (\boldsymbol{q}, r)$ specifies a ball in $\mathbb{R}^d$, i.e., all the $\boldsymbol{x}$ satisfying $dist(\boldsymbol{x}, \boldsymbol{q}) \leq r$, where *dist* is the Euclidean distance between the vectors $\boldsymbol{x}$ and $\boldsymbol{q}$, and r is a positive real value (query parameters are $\boldsymbol{q}$ and r). Circular reporting is fundamental in spatial databases and similarity retrieval; e.g., see [69]. By the standard "lifting trick" [58], we obtain directly from Theorem 4.5.4:

Corollary 4.5.1. *For top-k circular reporting with $d \geq 3$, there is:*

- *A RAM structure of $O(n \log n)$ space and $\tilde{O}(n^{1-1/\lfloor (d+1)/2 \rfloor}) + O(k)$ query time, both in the worst case.*
- *An EM structure of $O(n/B)$ space and $O((n/B)^{1-1/\lfloor (d+1)/2 \rfloor + \epsilon} + k/B)$ query time, both in the worst case, where $\epsilon > 0$ is an arbitrarily small constant.*

Interval Stabbing. This problem is among the most classic problems in the database area; e.g., see [70]. Here, $\mathcal{A}$ is a set of intervals on the real-line and the query q is a point, such that an element $e = [x, y]$ in $\mathcal{A}$ satisfies the predicate if $q \in [x, y]$. We obtain the following result:

Theorem 4.5.5. *For top-k interval stabbing, there is an EM structure of*

- *$O(n/B)$ space and $O(\log_B n + k/B)$ query time, both in expectation. The structure can be updated in $O(\log_B n)$ I/Os amortized expected per insertion and deletion.*
- *$O(n/B)$ space and $O(\log_B^2 n + k/B)$ query time, both in the worst case.*

2D Point Enclosure. In this problem, $\mathcal{A}$ is a set of rectangles in $\mathbb{R}^2$. The query is a point $q \in \mathbb{R}^2$, such that an element $e \in \mathcal{A}$ satisfies the predicate if $q \in e$.

To emphasize the relevance of a top-k query of this type to databases, let us consider a dating website, where a person registers requirements on her/his ideal significant other: age in the range $[x_1, x_2]$, and height in the range $[y_1, y_2]$. Therefore, his/her requirement can be modeled as a rectangle $[x_1, x_2] \times [y_1, y_2]$. The weight assigned to the rectangle could be the salary of the person. A reasonable query from, say, a lady is:

> "Find the 10 gentlemen with the highest salaries such that my age and height fall into their preferred ranges."

This is an instance of a top-k point enclosure query with $k = 10$. We obtain:

Theorem 4.5.6. *For top-k point enclosure, there is a RAM structure of*

- *$O(n \log^*(n))$ space and $O(\log n \log\log n + k)$ query time, both in expectation.*
- *$O(n \log^*(n))$ space and $O(\log^2 n \frac{\log\log n}{\log\log\log n} + k)$ query time, both in the worst case.*

3D Dominance. In this problem, $\mathcal{A}$ is a set of points in $\mathbb{R}^3$. The query is a point $q = (x, y, z)$, such that an element $e = (e_x, e_y, e_z)$ in $\mathcal{A}$ satisfies the predicate if $e_x \leq x$, $e_y \leq y$, and $e_z \leq z$. A practical top-k query of this type is

> "Find the 10 best-rated hotels whose (i) prices are at most x dollars per night, (ii) distances from the town center are at most y km, and (iii) security rating is at least z."

We obtain:

Theorem 4.5.7. *For top-k 3D dominance, there is a RAM structure of $O(n \log n / \log\log n)$ space and $O(\log^{1.5} n + k)$ query time, both in expectation.*

4.6 Previous Results

The top-k problem has been well-studied in many domains, including, for example, web search, information retrieval, recommender systems, etc. We refer the reader to Ilyas *et al.* for an excellent discussion of top-k query processing in relational databases [71].

We will focus on the geometric version of the top-k problem here. The work of [72] appears to be the first attempt to incorporate top-k features into conventional reporting queries. Since then, work on the topic has grown into a sizable literature. The most extensively studied (and, hence, the best understood) problem is *top-k orthogonal range reporting*, whose 1D version was studied in [73, 74, 75, 76, 77], and 2D version in [4, 5]. See also [78] for a colored version of the problem in 1D. The work [79] investigated more sophisticated colored top-k versions of several computational geometry problems. Top-k queries on text retrieval problems have been considered in [80, 81, 78, 82]; see also a recent survey [83].

Closely related to the top-k problem is the problem of reporting only the point with the kth-heaviest weight in the query range. Gagie *et al.* [84] and Navarro *et al.* [85] answer this query in $\mathbb{R}^1$ and $\mathbb{R}^2$, respectively.

Chapter 5

First Generic Reduction: Using counting and reporting structures

In this chapter we will present our first reduction and prove Theorem 4.4.1. We first outline the key steps in our query algorithm and then discuss each step in detail.

5.1 Key steps

Given a query pair (q, k), we do the following:

1. **Perform initial check:** Let $\mathcal{A}(q)$ be the set of objects of $\mathcal{A}$ intersected by q. If $|\mathcal{A}(q)| \leq k$, then we simply report all the objects in $\mathcal{A}(q)$ and stop. Otherwise, if $\mathcal{A}(q) > k$, we proceed to step 2.

2. **Find a threshold object:** We determine an object a_t in $\mathcal{A}(q)$ that has the kth-largest weight and proceed to step 3. We call a_t a *threshold object*.

3. **Report top-k objects:** Given a_t, we report all objects in $\mathcal{A}(q)$ whose weights are greater than or equal to w_t.

As we will see, steps 1 and 3 are essentially instances of the underlying GIQ counting and reporting problems, respectively. Step 2 will employ a binary search-based approach to quickly identify a_t.

5.2 Implementation of step 1

Let $\mathcal{D}_C$ (resp. $\mathcal{D}_R$) denote a data structure for the counting (resp. reporting) version of the underlying GIQ problem on $\mathcal{A}$. That is, given a query object q, $\mathcal{D}_C$ (resp. $\mathcal{D}_R$) returns the count $|\mathcal{A}(q)|$ (resp. the set $\mathcal{A}(q)$). (For example, for the top-k orthogonal range search problem, $\mathcal{D}_C$ (resp. $\mathcal{D}_R$) is a data structure for the counting (resp. reporting) version of orthogonal range search.) For future use, we assume that $\mathcal{D}_C$ and $\mathcal{D}_R$ support updates.

We do step 1 by querying $\mathcal{D}_C$ with q. If $|\mathcal{A}(q)| \leq k$, then we also query $\mathcal{D}_R$ with q and output $\mathcal{A}(q)$.

5.3 Implementation of step 2

Intuition: W.l.o.g. let $a_1, \ldots, a_n$ be an ordering of the objects of $\mathcal{A}$ by non-increasing weight (ties broken arbitrarily). Our goal is to find the kth-leftmost object in this ordering that is intersected by q; this is the threshold object a_t. Consider object a_m, where $m = \lfloor n/2 \rfloor$, and let $\mathcal{A}'$ (resp. $\mathcal{A}''$) be the ordered subset of $\mathcal{A}$ consisting of objects at or to the left of a_m (resp. to the right of a_m). We count the number of objects in $\mathcal{A}'$ that are intersected by q, i.e., we compute $|\mathcal{A}'(q)|$. If $|\mathcal{A}'(q)| \geq k$, then a_t is in $\mathcal{A}'$ and is the kth-leftmost object in $\mathcal{A}'(q)$. Therefore, we search recursively in $\mathcal{A}'$ for the kth-leftmost object. However, if $|\mathcal{A}'(q)| < k$, then a_t is in $\mathcal{A}''$ and is the $(k - |\mathcal{A}'(q)|)$th-leftmost object in $\mathcal{A}''(q)$. Therefore, we search recursively in $\mathcal{A}''$ for the $(k - |\mathcal{A}'(q)|)$th-leftmost object.

We implement the above idea as follows: We sort the objects of $\mathcal{A}$ by non-increasing weight (breaking ties arbitrarily) and store them in left-to-right order at the leaves of a balanced binary search tree $\mathcal{T}$. At each node v of $\mathcal{T}$, we store an instance, $\mathcal{D}_C^v$, of the structure $\mathcal{D}_C$ which is built on the objects stored in v's subtree.

Let r be the root of $\mathcal{T}$. At the beginning of this step, our objective is to find the kth-leftmost leaf among the leaves of $\mathcal{T}$ that store objects intersected by q; this leaf contains a_t. However, as the algorithm progresses and reaches some subtree of $\mathcal{T}$, our objective will change in the sense that we will now be seeking the k'th-leftmost leaf among the leaves of this subtree that store objects intersected by q, for some $k' \leq k$.

Specifically, let v_{cur} denote the root of the subtree of $\mathcal{T}$ that the search is at currently.

Initially, we know (from step 1) that q intersects more than k objects among the ones stored in $\mathcal{T}$'s leaves, so we must search in the left subtree of r for the k-th leftmost object intersected by q. Thus, initially v_{cur} is set to the left child of r and k' is set to k. Let $C(v_{cur})$ be the count returned when $\mathcal{D}_C^{v_{cur}}$ is queried with q. If $C(v_{cur}) \geq k'$, then the leaf containing a_t is in the left subtree of v_{cur}, so the search proceeds to this subtree with k' unchanged. However, if $C(v_{cur}) < k'$, then the leaf containing a_t is in the subtree of the sibling of v_{cur}, so the search proceeds to the sibling's subtree with k' set to $k' - C(v_{cur})$. This process repeats iteratively until the leaf, u, containing a_t is reached.

5.4 Implementation of step 3

We store the objects of $\mathcal{A}$ at the leaves of a balanced binary search tree $\mathcal{T}'$, in the same order in which they appear at the leaves of $\mathcal{T}$. At each node v of $\mathcal{T}'$, we store an instance, $\mathcal{D}_R^v$, of the structure $\mathcal{D}_R$ which is built on the objects stored in v's subtree. Also, if object a_i appears at leaf u of $\mathcal{T}$ and at a leaf u' of $\mathcal{T}'$, then we store a pointer, ptr, at u that points to u'; i.e., $ptr(u) = u'$. (In fact, we could use $\mathcal{T}$ to store instances of both $\mathcal{D}_C$ and $\mathcal{D}_R$. We use a separate structure $\mathcal{T}'$ only for ease of exposition.)

To report all objects in $\mathcal{A}(q)$ whose weights are greater than or equal to w_t, we query $\mathcal{T}'$ with q, as follows:

Let u be the leaf of $\mathcal{T}$ that is found to contain the threshold object a_t in step 2. We follow $ptr(u)$ to find the leaf u' of $\mathcal{T}'$ that contains a_t. We then walk from u' up to the root of $\mathcal{T}'$ following parent pointers, thereby tracing a path, Π, in $\mathcal{T}'$. Let Z be the set of nodes, v, in $\mathcal{T}'$ such that v is the left child of a node on Π but is itself not on Π. We also include in Z the leaf u'. Z consists of both leaves and internal nodes and we call each such node a *canonical node*. Note that $|Z| = O(\log n)$ and, moreover, for each $v \in Z$, the range $[w_t, \infty)$ contains the weights of all the objects stored in v's subtree. For each $v \in Z$, we query $\mathcal{D}_R^v$ with q, which causes all objects in v's subtree that are intersected by q to be reported.

This concludes the description of the 3-step query algorithm. The algorithm is presented in pseudocode as Algorithm 1.

Algorithm 1: Query algorithm for top-k GIQ

Input: Data structures $\mathcal{T}$ and $\mathcal{T}'$ storing objects of $\mathcal{A}$ as described in Sections 5.2–5.4, query object q, and integer $k > 0$.

Output: The k largest-weight objects of $\mathcal{A}$ that are intersected by q.

begin

`// Step 1`

Query $\mathcal{D}_C^r$ with q to compute the number, $C(r)$, of objects in r's subtree that are intersected by q, where r is the root of $\mathcal{T}$.

if $C(r) \leq k$ **then**

Query $\mathcal{D}_R^{r'}$ with q to find all the objects in r''s subtree that are intersected by q, where r' is the root of $\mathcal{T}'$. Report these objects and exit.

`// Step 2`

$v_{cur} \longleftarrow$ left child of r

$k' \longleftarrow k$

while $v_{cur} \neq nil$ **do**

Query $\mathcal{D}_C^{v_{cur}}$ with q to compute the number of objects, $C(v_{cur})$, in v_{cur}'s subtree that are intersected by q.

$u \longleftarrow v_{cur}$

if $C(v_{cur}) < k'$ **then**

$k' \longleftarrow k' - C(v_{cur})$

$v_{cur} \longleftarrow$ sibling of v_{cur}

else

$v_{cur} \longleftarrow$ left child of v_{cur}

`// Step 3`

$u' \longleftarrow$ leaf of $\mathcal{T}'$ corresponding to u

Walk up $\mathcal{T}'$ from u' and identify the set, Z, of canonical nodes. For each $v \in Z$, query $\mathcal{D}_R^v$ with q and report all objects returned.

5.5 An example

We illustrate the query algorithm in Figure 5.1. Part (a) shows the input objects (points in $\mathbb{R}^2$) and the query object q (a rectangle), part (b) shows the structure $\mathcal{T}$, and part (c) shows the structure $\mathcal{T}'$. To avoid clutter, the structures $\mathcal{D}_C$ and $\mathcal{D}_R$ are not shown at the nodes. For simplicity, we refer to the points by their weights. For $k = 4$, the threshold point is 40 and the top-4 points are 80, 60, 50, and 40. Let v_1 be the root of $\mathcal{T}$. Step 1 finds $C(v_1)$ to be 5, corresponding to the points 80, 60, 50, 40, and 20 in v_1's subtree that lie inside q. Since $C(v_1) > k$ we proceed to Step 2.

In Step 2 our objective is to find a leaf node v such that among the points stored at v and the leaf nodes to the left of v, exactly 4 points lie inside q. Initially, $v_{cur} = v_2$ and $k' = k = 4$. Querying $\mathcal{D}_C^{v_2}$ gives $C(v_2) = 3$, corresponding to the points 80, 60, and 50 in v_2's subtree that lie inside q. Thus, the threshold point is not in the subtree of v_2 but instead is in the subtree of its sibling node v_3. Since $k = 4$ and $C(v_2) = 3$, k' is reset to $k - A(v_2) = 1$, and the search proceeds to v_3. Querying $\mathcal{D}_C^{v_3}$ gives $C(v_3) = 2$, corresponding to the points 40 and 20 in v_3's subtree that lie inside q. Since $C(v_3) > k'$, we proceed to v_3's left child v_4. Querying $\mathcal{D}_C^{v_4}$ we find that $C(v_4) = 1$, corresponding to point 40 lying inside q. Since $C(v_4) = k'$, we proceed to v_4's left child v_5. Querying $\mathcal{D}_C^{v_5}$ yields $C(v_5) = 1$, corresponding to point 40 lying inside q. Since $C(v_5) = k'$, we proceed to v_5's left child which happens to be *nil*. At this point we exit the **while**-loop with $u = v_5$ containing the threshold point 40.

Finally, in step 3, we follow $ptr(v_5)$ (not shown) to locate the leaf in $\mathcal{T}'$ storing threshold point 40, identify the path Π and the set Z of canonical nodes, and query $\mathcal{D}_R^v$ at each node $v \in Z$ with q to report the top-4 points in q.

5.6 Proof of Theorem 4.4.1

The correctness of the query algorithm follows from the discussion in Sections 5.2–5.4.

We now analyze the space bound. Let $v_1, v_2, \ldots, v_t$ be the nodes of $\mathcal{T}$ at a given level (i.e., distance from the root) and let $n_1, n_2, \ldots, n_t$ be, respectively, the number of objects stored at the leaves of their subtrees. The space used by all the secondary structures, $\mathcal{D}_C^{v_i}$, at these nodes is $\sum_{i=1}^{t} S_c(n_i, d) = \sum_{i=1}^{t} (S_c(n_i, d)/n_i) \times n_i \leq (S_c(n, d)/n) \sum_{i=1}^{t} n_i = O(S_c(n, d))$, since $S_c(n_i, d)/n_i$ is non-decreasing, $n_i \leq n$, and $\sum_{i=1}^{t} n_i \leq n$. Since $\mathcal{T}$ has

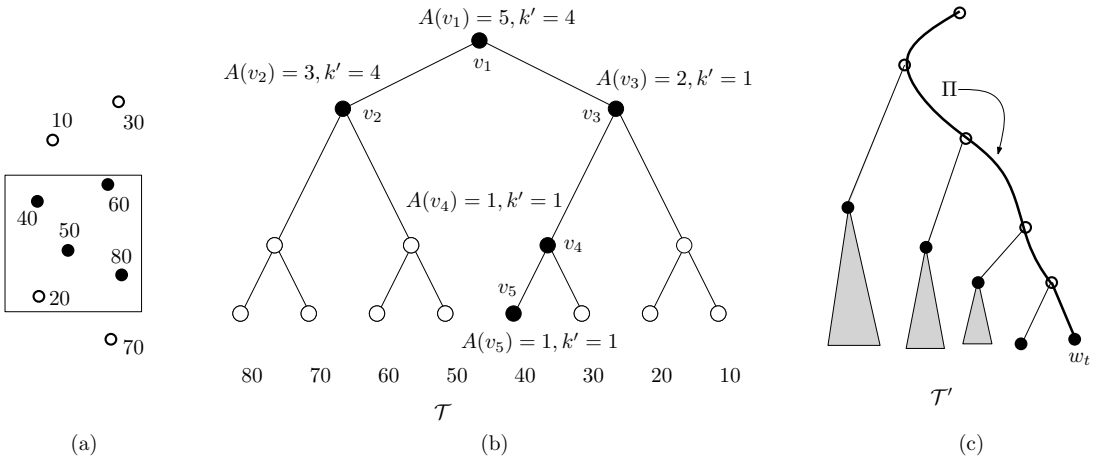


Figure 5.1: The general technique illustrated for top-k orthogonal range search in $\mathbb{R}^2$, with $k = 4$. (a) Set $\mathcal{A}$ consisting of 8 weighted points and query rectangle q. Points shown filled are the k largest-weight objects intersected by q. (b) Finding the threshold point by querying $\mathcal{T}$. The nodes visited by the query algorithm are shown filled. (c) Search path, Π, in $\mathcal{T}'$ (shown in heavy lines) and canonical nodes (shown filled).

height $O(\log n)$, the space used by $\mathcal{T}$ is $O(S_c(n,d)\log n)$. Similarly, the space used by $\mathcal{T}'$ is $O(S_r(n,d)\log n)$. Thus, the overall space is $O(max\{S_c(n,d), S_r(n,d)\}\log n) = O(S(n,d)\log n)$.

Next, we analyze the query time. The time for step 1 is $O(Q_c(n,d) + Q_r(n,d) + k)$. For step 2, consider the path in $\mathcal{T}$ from the root to the leaf node containing a_t. At each node v on the path, the secondary structure $\mathcal{D}_C^v$ is queried with q. Also, if v is a right child of its parent, then the secondary structure at the left child of v's parent is also queried. So, at each level of $\mathcal{T}$, the secondary structures of at most two nodes, v_1 and v_2, at that level are queried. Let n_i, $i = 1, 2$, be the number of objects stored at the leaves of v_i's subtree. Thus step 2 takes $\sum_{i=1}^{2} Q_c(n_i, d)$ time. Since $Q_c(n_i, d)$ is non-decreasing and $n_i \leq n$, $i = 1, 2$, the query time per level is $\sum_{i=1}^{2} Q_c(n_i, d) = O(Q_c(n, d))$. Summing over the $O(\log n)$ levels of $\mathcal{T}$ gives an overall query time of $O(Q_c(n,d)\log n)$ time for step 2.

In Step 3, it takes $O(\log n)$ time to identify the set, Z, of canonical nodes. For each $v_i \in Z$, let n_i be the number of objects stored at the leaves of v_i's subtree and let k_i be the number of these objects intersected by q. Querying $\mathcal{D}_R^{v_i}$ with q at each $v_i \in Z$ takes $O(Q_r(n_i, d) + k_i)$ time. Thus the total query time in step 3 is $O(\sum_{i=1}^{|Z|}(Q_r(n_i, d) + k_i))$.

Since $Q_r(n_i, d)$ is non-decreasing, and since $\sum_{i=1}^{|Z|} k_i = k$, the query time for step 3 is $O(Q_r(n, d) \log n + k)$.

Therefore, the time for steps 1–3 is $O(max\{Q_c(n, d)+Q_r(n, d)\} \log n+k) = O(Q(n, d) \log n+ k)$.

Finally, we consider the update time. If $\mathcal{T}$ and $\mathcal{T}'$ are implemented as $BB(\alpha)$ trees, then the technique of Willard *et al.* [86] can be used to keep the trees balanced as updates are performed. As shown in [86], the amortized update time for $\mathcal{T}$ and $\mathcal{T}'$ will be $O(U_c(n, d) \log n)$ and $O(U_r(n, d) \log n)$, respectively. Thus, the overall update time will be $O(max\{U_c(n, d) + U_r(n, d)\} \log n) = O(U(n, d) \log n)$ (amortized).

5.7 Remarks

1. In [4] we presented experimental results by applying this general reduction to two problems: orthogonal range searching and rectangle stabbing. The experiments showed that our data structures were quite efficient in practice, in terms of storage and query time.

2. In our current solution, we are trying to find the object a_t which has the kth-largest weight in $\mathcal{A}(q)$. However, if we observe closely, it suffices to find any object a which has k'th-largest weight in $\mathcal{A}(q)$, where $k' \in [k, ck]$ for some constant c. In the implementation of step 3 we will end up reporting $k' = \Theta(k)$ objects from which we can filter the top-k objects in $O(k')$ time by a standard selection algorithm. Lets call a an *approximate threshold object*.

   How can we make use of the idea of approximate threshold object? It turns out that instead of a counting structure $\mathcal{D}_C^v$ at each node of the tree $\mathcal{T}$, one can store an *approximate counting* structure which reports a c-approximate value (if the exact counting structure reports t then the approximate counting structure reports any value in the range $[t, ct]$). Now repeating the same query algorithm as before, it can be seen that one can obtain an approximate threshold object.

   For many GIQ problems (such as halfspace range searching), approximate counting structures have far better bounds than the exact counting structures and hence, this observation of approximate threshold object is useful for such problems. For

example, consider the halfplane range searching in $\mathbb{R}^2$ problem. There is linear-sized data structure which answers the exact counting query for this problem in roughly $O(\sqrt{n})$ time [87], whereas there is a linear-sized data structure which answers a c-approximate counting query in merely $O(\log n)$ time [88].

Chapter 6

Second Generic Reduction: Using top-1 and prioritized reporting structures

In this chapter we will prove Theorem 4.4.2 and Theorem 4.4.3. Recall that in Theorem 4.4.2 we prove that prioritized reporting is no harder than top-k reporting for any GIQ. Next, in proving Theorem 4.4.3 we present our second reduction which uses the max reporting and the prioritized reporting structure without any performance loss in the space, the query time and the update time (in the expected sense).

6.1 Prioritized reporting is no harder than top-k reporting

In this section we prove Theorem 4.4.2. Assume there is a data structure, $\mathcal{D}$, which can answer the top-k GIQ. Suppose we have a set, $\mathcal{A}$, of n objects and we want to answer a prioritized reporting query on $\mathcal{A}$. We will show that this can be done using a data structure that uses $O(\mathcal{S}_{top}(n))$ space and has a query time of $O(\mathcal{Q}_{top}(n) + t/B)$ I/Os, where t is the number of points reported.

Based on the objects of $\mathcal{A}$ we build an instance of the data structure $\mathcal{D}$. Clearly the space occupied by $\mathcal{D}$ will be $O(\mathcal{S}_{top}(n))$.

Given a query q and a real value τ, the query algorithm is executed in multiple

rounds. In round j (starting with $j = 1$), we query $\mathcal{D}$ with $(q, k = 2^{j-1} \cdot B \cdot \mathcal{Q}_{top}(n))$. Two cases arise:

1. If exactly $2^{j-1} \cdot B \cdot \mathcal{Q}_{top}(n)$ objects are reported, then we scan the reported objects to find the object with the smallest weight (say w_i). If $w_i < \tau$, then we do not go to the next round. Otherwise, we go to round $j + 1$.

2. If less than $2^{j-1} \cdot B \cdot \mathcal{Q}_{top}(n)$ objects are reported, then we do not go to the next round.

If the query does not go beyond round j, then among the objects reported in round j, we remove each object whose weight is less than τ. The remaining objects are the output of the prioritized reporting query on $\mathcal{A}$ with q and τ.

Now we analyze the time taken to answer the prioritized reporting query. There are two cases:

1. Only one round is executed: Then the query time will be $O(\mathcal{Q}_{top}(n))$ since the value of $k = B \cdot \mathcal{Q}_{top}(n)$.

2. There are $i > 1$ rounds of execution: Then the query time will be

$$O\left(\sum_{j=1}^{i}\left(\mathcal{Q}_{top}(n) + \frac{2^{j-1} \cdot B \cdot \mathcal{Q}_{top}(n)}{B}\right)\right) = O(2^i \cdot \mathcal{Q}_{top}(n)).$$

   The crucial observation is that since the $(i-1)$-th round was executed and we then entered i-th round, we have $t \geq 2^{i-2} \cdot B \cdot \mathcal{Q}_{top}(n)$, which implies that $2^i \cdot B \cdot \mathcal{Q}_{top}(n) \leq 4t$. Therefore, the query time will be $O(2^i \cdot \mathcal{Q}_{top}(n)) = O(t/B)$.

Therefore, the overall query time will be $O(\mathcal{Q}_{top}(n) + t/B)$.

6.2 Reduction

In this section, we present a reduction to establish the correctness of Theorem 4.4.3. Our discussion focuses on $n \geq B \cdot \mathcal{Q}_{max}(n)$; otherwise, a top-$k$ query can be trivially answered by performing k-selection on the whole D in $O(n/B) = O(\mathcal{Q}_{max}(n))$ I/Os.

Key Idea and Challenges. We start with some definitions. Let S be a set of elements. By independently sampling each element of S with probability p, we obtain a *p-sample set R*. Furthermore, let us assume that the elements of S are distinct and are drawn from an ordered domain. We say that an element $e \in S$ has *rank i*, if e is the i-th greatest in S. Intuitively, given an integer k that is reasonably large and $p = 1/k$, the element e_{max} with rank 1 in R has some positive probability of having rank in the range $[k, ck]$ in S, for some constant c. If this happens, then we can use e_{max} to retrieve the top-k objects in S (report all objects in S with weight greater than e_{max}). The key technical challenges are the following: (i) the value of k will be known only during the query, so our random samples should be able to handle all values of k, and (ii) how to efficiently handle the case where e_{max}'s rank in S is not in the range $[k, ck]$?

Rank Sampling. The following lemma formalizes the intuition presented above.

Lemma 6.2.1. *Let S be a set of n elements, and $K \geq 2$ a real value satisfying $n \geq 4K$. For a $(1/K)$-sample set R of S, the following hold simultaneously with probability at least 0.09:*

- $|R| \geq 1$
- *The largest element in R has rank in S greater than K but at most $4K$.*

Proof. The first bullet fails only if none of the elements in D were sampled, which occurs with a probability

$$(1 - 1/K)^n \leq (1 - 1/K)^{4K} \leq 1/e^4$$

where the last inequality used the fact that $(1 - x)^{1/x} < 1/e$ for all $x \geq 0$.

Let e be the largest element in R (note: this should be distinguished from the base of natural logarithm; the semantics of each occurrence of "e" should be clear from the context throughout the thesis). Denote by $\hat{K}$ the rank of e in D. Next, we bound the probability of the event $\hat{K} > 4K$, which occurs only if none of the $4K$ largest elements in D were sampled. Hence:

$$\mathbf{Pr}[\hat{K} > 4K] = (1 - 1/K)^{4K} \leq 1/e^4.$$

Finally, we bound the probability of the event $\hat{K} \leq K$, which occurs only if at least one of the K largest elements in D was sampled. Hence:

$$\mathbf{Pr}[\hat{K} \leq K] = 1 - (1 - 1/K)^K.$$

Applying the fact that $(1 - 1/x)^x \geq 1/e^2$ for all $x \geq 2$, we know:

$$\mathbf{Pr}[\hat{K} \leq K] \leq 1 - 1/e^2.$$

The union bound now shows that the probability of violating at least one bullet of Lemma 6.2.1 is at most

$$2/e^4 + (1 - 1/e^2) < 0.91.$$

We thus complete the proof. □

Structure. We now describe how to design a top-k structure from (i) a prioritized structure, which uses $\mathcal{S}_{pri}(n)$ space on n elements and answers a prioritized query in $\mathcal{Q}_{pri}(n) + O(t/B)$ I/Os, and (ii) a max structure, which uses $\mathcal{S}_{max}(n)$ space and answers a max query in $\mathcal{Q}_{max}(n)$ I/Os.

Fix a constant $\sigma = 1/20$. For each integer $i \geq 1$, define:

$$K_i \quad = \quad B \cdot \mathcal{Q}_{max}(n) \cdot (1 + \sigma)^{i-1}.$$

Let h be the largest i such that $K_i \leq n/4$; clearly, $h = O(\log(n/B))$. We create a prioritized structure on $\mathcal{A}$. Also, for each $i \in [1, h]$, we take a $(1/K_i)$-sample set R_i of $\mathcal{A}$, and create a max structure on R_i.

Query. Let us first eliminate queries with $k < B \cdot \mathcal{Q}_{max}(n)$. Given such a query q, we first treat it as a top-$(B \cdot \mathcal{Q}_{max}(n))$ query, i.e., extracting the $B \cdot \mathcal{Q}_{max}(n)$ elements with the greatest weights in $\mathcal{A}(q)$. Then, the final result of the original query can be obtained by performing k-selection on those elements. The total cost is $O(\mathcal{Q}_{max}(n))$ plus the time of the top-$(B \cdot \mathcal{Q}_{max}(n))$ query.

Let us now focus on a top-k query q with $k \geq B \cdot \mathcal{Q}_{max}(n)$. If $k > K_h$, the query is answered naïvely by reading the whole $\mathcal{A}$ in $O(n/B)$ I/Os, which is $O(k/B)$ because $k > K_h \geq n/(4(1 + \sigma)) = \Omega(n)$.

If $k \leq K_h$, identify the smallest i such that $K_i \geq k$; note that $K_i = \Theta(k)$. Setting $j = i$, we carry out a *round* with the steps below:

1. If $|\mathcal{A}(q)| \leq 4K_j$, solve the query with the prioritized structure on $\mathcal{A}$ in the cost-monitoring manner (see Section 7.3), which costs $\mathcal{Q}_{pri}(n) + O(K_j/B)$ I/Os. The algorithm declares the round *succeeded* and terminates.

2. Otherwise, identify the element e in $R_j(q)$ with the maximum weight from the max structure on R_j in $\mathcal{Q}_{max}(n)$ I/Os. In the special case where $R_j(q)$ is empty, treat e as a dummy element with $w(e) = -\infty$.

3. Perform a prioritized query on $\mathcal{A}$ with q and threshold $\tau = w(e)$ in a cost-monitoring manner:

   (a) Either the query terminates by itself—let S be the set of elements retrieved,

   (b) Or we terminate it as soon as $4K_j + 1$ elements have been reported.

   In both cases, the cost is $\mathcal{Q}_{pri}(n) + O(K_j/B)$.

4. Declare this round *failed* if either of the following is true:

   - Case 3(a) occurred, but $|S| \leq K_j$.
   - Case 3(b) occurred.

   Otherwise, declare this round *succeeded*.

5. If succeeded, perform k-selection on S to produce the k elements in $\mathcal{A}(q)$ with the largest weights, and terminate the algorithm by returning them as the final answer.

6. Otherwise (i.e., failed), increase j by 1.

   (a) If $j \leq h$, start the next round from Step 1.

   (b) Else (i.e., $j = h + 1$), answer the top-k query naïvely by reading the whole $\mathcal{A}$ in $O(n/B) = O(K_h/B)$ I/Os; the algorithm then terminates. This is the only scenario where termination can happen in a failed round.

To analyze the cost of the algorithm, notice that a round fails only if (i) $|\mathcal{A}(q)| > 4K_j$ (otherwise, Line 1 terminates the algorithm), and (ii) one of the two bullets in Step 4 is true. Thus, Lemma 6.2.1 tells us that failure happens with probability at most

$1 - 0.09 = 0.91$, noticing that $R_i(q)$ is a $(1/K_j)$-sample set of $\mathcal{A}(q)$. This implies that round j—for a specific $j \geq i$—is executed only with probability 0.91^{j-i}, namely, only when all the preceding rounds have failed. Also observe that round j, regardless of whether it fails, performs at most

$$\mathcal{Q}_{pri}(n) + \mathcal{Q}_{max}(n) + O(K_j/B)$$

I/Os. Thus, the expected cost of the algorithm is bounded by

$$\begin{aligned} & \sum_{j=i}^{h} O\Big(\Big(\mathcal{Q}_{pri}(n) + \mathcal{Q}_{max}(n) + \frac{K_j}{B}\Big) \cdot 0.91^{j-i}\Big) \\ = \quad & O\Big(\mathcal{Q}_{pri}(n) + \mathcal{Q}_{max}(n) + \sum_{j=i}^{h} \frac{K_j}{B} 0.91^{j-i}\Big) \end{aligned} \tag{6.1}$$

Notice that $K_j = K_i \cdot (1+\sigma)^{j-i} = O(k) \cdot (1+\sigma)^{j-i}$. Plugging these into (6.1) shows that the expected cost is

$$O\left(\mathcal{Q}_{pri}(n) + \mathcal{Q}_{max}(n) + \frac{k}{B} \sum_{j=i}^{h} ((1+\sigma) \cdot 0.91)^{j-i}\right)$$

which is $O(\mathcal{Q}_{pri}(n) + \mathcal{Q}_{max}(n) + k/B)$ because $(1+\sigma) \cdot 0.91 < 1$.

Handling updates. It remains to discuss how to support insertions and deletions on the input set $\mathcal{A}$. This is in fact fairly easy, if one observes that each element $e \in \mathcal{A}$ has in expectation only $O(1)$ copies in the entire structure—recall that the sampling rate of R_i equals $1/K_i$, which geometrically decreases as i increases. Hence, the insertion of e triggers one insertion into the prioritized structure, and one insertion into $O(1)$ max structures in expectation. The total cost is thus $O(\mathcal{U}_{pri} + \mathcal{U}_{max})$ expected. Also, we can record in $O(1)$ expected words which max structures include e. By hashing, this "bookkeeping" itself can be maintained in $O(1)$ expected I/Os as e is inserted/deleted, without increasing the overall space complexity. In this way, a deletion of e can also be supported in $O(\mathcal{U}_{pri} + \mathcal{U}_{max})$ expected I/Os.

The above argument still works even if one or both of $\mathcal{U}_{pri}$ and $\mathcal{U}_{max}$ are amortized. This completes the proof of Theorem 4.4.3.

6.3 Remarks

1. At a conceptual level, our reduction may be reminiscent of a method by Aronov and Har-Peled [36] that reduces approximate counting to *emptiness queries*. However, our approach differs substantially in both algorithmic and technical details, a quick proof of which is the following fact: the counting structure of [36] suffers from performance degradation by a logarithmic factor compared to the emptiness structure, while our reduction incurs no performance degradation.

2. **Open Problem:** Is there a reduction which obtains the same bounds as our reduction but also holds in the worst-case. This looks like a challenging problem and will require new ideas.

Chapter 7

Third generic reduction: Using only the prioritized reporting structure

This section serves as a proof of Theorem 4.4.4. We will need the Chernoff bounds given in the appendix (at the end of the chapter).

7.1 Key Ideas

We use two new ideas to build a top-k strucuture using only the prioritized reporting structure. Firstly, when k is large we prove the existence of a *core-set* that allows us to reduce the problem to a top-$\Theta(\log n)$ query. Secondly, when k is small we construct *nested core-sets* $R_1, R_2, \ldots, R_h$ and then to answer a query on any R_i, we use a recursive mechanism that "fine-tunes" the results of queries on $R_{i+1}, \ldots, R_h$.

7.2 Top-k Core-Set

Rank Sampling. Let S be a set of elements. By independently sampling each element of S with probability p, we obtain a *p-sample set* R. Furthermore, let us further assume that the elements of S are distinct, and drawn from an ordered domain. We say that an element $e \in S$ has *rank* i, if e is the i-th greatest in S. Intuitively, given an integer

k that is reasonably large, the element with rank kp in R ought to have rank roughly k in S. The next lemma formalizes this intuition.

Lemma 7.2.1. *Let S be a set of n elements, and R be a p-sample set of S. Suppose that integer $k \geq 1$ and real value $\delta \in (0,1)$ satisfy $kp \geq 3\ln(3/\delta)$ and $n \geq 4k$. Then, the following hold simultaneously with probability at least $1-\delta$:*

- $|R| > 2kp$
- *The element with rank $\lceil 2kp \rceil$ in R has rank between k and $4k$ in S.*

Proof. The first bullet fails with probability

$$\begin{aligned}
\mathbf{Pr}[|R| \leq 2kp] &= \mathbf{Pr}[|R| \leq (2k/n) \cdot np] \\
&\leq \mathbf{Pr}[|R| \leq (1/2)np] \\
\text{(Chernoff bound (7.9))} &\leq \exp(-np/12) \\
&\leq \exp(-kp/3) \\
&\leq \delta/3.
\end{aligned}$$

Let e be the element with rank $\lceil 2kp \rceil$ in R, and $\hat{k}$ be the rank of e in S. Next, we bound the probability of the event $\hat{k} > 4k$. For $i \in [1, 4k]$, define x_i to be 1 if the i-th greatest element in S is sampled, or 0 otherwise. Let $X = \sum_{i=1}^{4k} x_i$, and thus, $\mathbf{E}[X] = 4kp \geq 12\ln(3/\delta)$. Event $\hat{k} > 4k$ implies $X \leq \lceil 2kp \rceil - 1$. We have:

$$\begin{aligned}
\mathbf{Pr}[\hat{k} > 4k] &\leq \mathbf{Pr}[X \leq \lceil 2kp \rceil - 1] \\
&= \mathbf{Pr}[X < 2kp] \\
&= \mathbf{Pr}[X < (1/2) \cdot \mathbf{E}[X]] \\
\text{(Chernoff bound (7.9))} &\leq \exp(-\mathbf{E}[X]/12). \\
&\leq \delta/3.
\end{aligned}$$

Finally, we bound the probability of the event $\hat{k} < k$. Define $Y = \sum_{i=1}^{k} x_i$, and thus, $\mathbf{E}[Y] = kp \geq 3\ln(3/\delta)$. Event $\hat{k} < k$ implies that $Y \geq \lceil 2kp \rceil$. We have:

$$\begin{aligned}
\mathbf{Pr}[\hat{k} < k] &\leq \mathbf{Pr}[Y \geq 2kp] \\
&= \mathbf{Pr}[Y \geq 2\mathbf{E}[Y]] \\
\text{(Chernoff bound (7.10))} &\leq \exp(-\mathbf{E}[Y]/3) \\
&\leq \delta/3.
\end{aligned}$$

By the union bound, the two bullets in the lemma hold simultaneously with probability at least $1 - \delta$. □

Core-Set. As stated in Theorem 4.4.4, suppose that the underlying problem is polynomially bounded. More specifically, we say that the problem is λ-*polynomially bounded* if for any input $\mathcal{A}$ of n elements, there are at most n^λ distinct outcomes for $\mathcal{A}(q)$ over all possible q, where λ is a constant.

Given a subset R of $\mathcal{A}$, we say that an element $e \in R$ has *weight rank* i in R if it has the i-th greatest weight in R. The next lemma proves the existence of a small-size core-set that approximately captures a specific rank for all the "large" queries whose predicates are satisfied by many elements.

Lemma 7.2.2 (Top-k Core-Set Lemma)**.** *For any integer $K \geq 4\lambda \ln n$, there is a subset R of $\mathcal{A}$ such that*

- $|R| \leq 12\lambda \cdot (n/K) \ln n$.
- *For any q satisfying $|\mathcal{A}(q)| \geq 4K$, the following hold:*
  - $|R(q)| > 8\lambda \ln n$
  - *The element with weight rank $\lceil 8\lambda \ln n \rceil$ in $R(q)$ has weight rank between K and $4K$ in $\mathcal{A}(q)$.*

Proof. Set $p = 4(\lambda/K) \ln n$, and $\delta = 1/(2n^\lambda)$. These values ensure:

$$Kp = 4\lambda \ln n \geq 3 \ln(3/\delta). \tag{7.1}$$

Let R be a p-sample set of $\mathcal{A}(q)$. We will prove that R satisfies all the conditions in the lemma with a non-zero probability.

Fix a q satisfying $|\mathcal{A}(q)| \geq 4K$. Clearly, $R(q)$ is a p-sample set of $\mathcal{A}(q)$. Applying Lemma 7.2.1 on $S = \mathcal{A}(q)$ (the application is enabled by (7.1)), we know that with probability at least $1 - \delta$, the following hold simultaneously:

- $|R(q)| > 2Kp = 8\lambda \ln n$.
- The element with rank $\lceil 2Kp \rceil$ has rank between K and $4K$ in $\mathcal{A}(q)$.

By λ-polynomially boundedness and the union bound, the above holds for all queries with probability at least $1 - \delta n^{\lambda} = 1/2$.

Finally, as $|R|$ equals np in expectation, by Markov's inequality, $|R| \leq 3np = 12\lambda(n/K)\ln n$ with probability at least $2/3$. It thus follows that all the conditions of the lemma hold with probability at least $1 - (1/2 + 1/3) > 0$. □

7.3 Structure

We proceed to explain how to use a prioritized reporting structure to design a top-k reporting structure on a problem that is λ-polynomially bounded for a constant λ. Recall that the former structure consumes space $\mathcal{S}_{pri}(n)$ space on n elements, and answers a prioritized query in $\mathcal{Q}_{pri}(n) + O(t/B)$ I/Os.

Define:

$$g = \frac{\mathcal{Q}_{pri}(n)}{\log_B n} \tag{7.2}$$

$$f = 12\lambda B \cdot \mathcal{Q}_{pri}(n). \tag{7.3}$$

Note that $g \geq 1$ (as required by Theorem 4.4.4), and for $B \geq 64$, both the following are fulfilled:

$$\frac{12\lambda}{f} \cdot \ln n \leq \frac{1}{g\sqrt{B}} \tag{7.4}$$

$$f \geq \lceil 8\lambda \ln n \rceil. \tag{7.5}$$

To prove Theorem 4.4.4, next we first describe our solution to top-k queries with $k \leq f$, and then, queries with larger k.

Queries with $k \leq f$. It suffices, in fact, to consider $k = f$. Given a query q with $k < f$, we first treat it as a top-f query, i.e., retrieving the set of f elements with the greatest weights in $\mathcal{A}(q)$. Then, the final result of q can be easily obtained by performing *k-selection* [10] on these elements in $O(f/B) = O(\mathcal{Q}_{pri}(n))$ I/Os. Apart from this, the cost depends only on the top-f query.

Given a top-f query q, we answer it directly using a prioritized structure on $\mathcal{A}$, if $|\mathcal{A}(q)| \leq 4f$. We do *not* need any counting structure for estimating $\mathcal{A}(q)$. Instead, we

can achieve the purpose by issuing a *prioritized query* with predicate q and threshold $\tau = -\infty$ in a *cost monitoring* manner:

- Either the query terminates by itself
- Or we terminate it manually as soon as $4f + 1$ elements have been reported.

In both cases, the number of I/Os performed by the query is at most $\mathcal{Q}_{pri}(n)+O(f/B) = O(\mathcal{Q}_{pri}(n))$. In the former case, we obtain the final result of the top-f query by performing k-selection on the elements fetched by the prioritized query. In the latter case, it must hold that $|\mathcal{A}(q)| > 4f$; we answer such queries with a structure constructed as follows.

Take a core-set R_1 of $\mathcal{A}$ using Lemma 7.2.2 with $K = f$, and build a prioritized structure on R_1. This process is then carried out recursively: for every $i \geq 2$, we take a core-set R_{i+1} of R_i with the same $K = f$, and build a prioritized structure on R_{i+1}. The recursion ends at some $i = h$ where $|R_h| \leq 4f$.

For convenience, let us treat $\mathcal{A}$ as R_0. For each R_i $(0 \leq i \leq h-1)$, it holds that $f \geq 4\lambda \ln n \geq 4\lambda \ln |R_i|$; hence, by Lemma 7.2.2:

$$\begin{aligned} |R_{i+1}| \leq \frac{12\lambda \cdot |R_i|}{f} \ln |R_i| &\leq \frac{12\lambda \cdot |R_i|}{f} \ln n \\ \text{(by (7.4))} &\leq \frac{|R_i|}{g\sqrt{B}}. \end{aligned} \tag{7.6}$$

The total space of all the prioritized structures is therefore $O(\mathcal{S}_{pri}(n))$ by the fact that $\mathcal{S}_{pri}(n)$ is geometrically converging. Furthermore, (7.6) indicates that

$$h = O(\log_{g\sqrt{B}} n).$$

We now explain inductively how to answer a top-f query on any R_i for $i \in [0, h]$ in no more than

$$c \cdot (h - i + 1) \cdot \mathcal{Q}_{pri}(n) \tag{7.7}$$

I/Os, for some constant $c \geq 1$. At the bottom level $i = h$, the purpose can be easily achieved by scanning the entire R_h in $O(f/B)$ I/Os, which is at most $c_1 \cdot \mathcal{Q}_{pri}(n)$ for some constant $c_1 \geq 1$. Assuming that this can be done for all $i \geq j + 1$, consider $i = j$, at which level we distinguish two scenarios:

- $|R_j(q)| \leq 4f$: Answer the query in the cost monitoring manner as explained earlier using the prioritized structure on R_j. The cost is $\mathcal{Q}_{pri}(|R_j|)+O(f/B) \leq c_2 \cdot \mathcal{Q}_{pri}(n)$ for some constant $c_2 \geq 1$.

- $|R_j(q)| > 4f$: According to Lemma 7.2.2, $R_{j+1}(q)$ must have size at least $\lceil 8\lambda \ln |R_j(q)| \rceil$. By (7.5), it must hold:

  $$f \geq \lceil 8\lambda \ln |R_j(q)| \rceil.$$

  Therefore, we can retrieve the element e with weight rank $\lceil 8\lambda \ln |R_j(q)| \rceil$ in $R_{j+1}(q)$ by issuing a top-f query on R_{j+1} in at most

  $$c \cdot (h-i) \cdot \mathcal{Q}_{pri}(n)$$

  I/Os. Lemma 7.2.2 indicates that the weight rank of e in $R_j(q)$ is between f and $4f$. We deploy the prioritized structure on R_j to fetch all the elements of $R_j(q)$ with weights at least $w(e)$ in $\mathcal{Q}_{pri}(|R_j|) + O(f/B) \leq c_2 \cdot \mathcal{Q}_{pri}(n)$ I/Os. The result of the top-f query can be obtained from these objects with "k-selection" in no more than $c_3 \cdot \mathcal{Q}_{pri}(n)$ I/Os for some constant $c_3 \geq 1$.

We choose c to be $\max\{c_1, c_2 + c_3\}$, which ensures:

$$\begin{aligned} & c \cdot (h-i) \cdot \mathcal{Q}_{pri}(n) + (c_2 + c_3) \cdot \mathcal{Q}_{pri}(n) \\ & \leq c \cdot (h-i+1) \cdot \mathcal{Q}_{pri}(n) \end{aligned}$$

and hence, completing our claim in (7.7). It thus becomes clear that the cost of answering a query on $\mathcal{A}$ is

$$O(h \cdot \mathcal{Q}_{pri}(n)) = O(\mathcal{Q}_{pri}(n) \cdot \log_{g\sqrt{B}} n).$$

Plugging in the definition equation (7.2) of g gives the claimed complexity in Theorem 4.4.4.

Queries with $k > f$. We apply Lemma 7.2.2 to take a core-set $R[i]$ of $\mathcal{A}$ with $K = 2^{i-1}f$, for $i = 1, 2, ..., h$, where h is the largest integer i satisfying $2^{i-1}f \leq n$. It is easy to verify from (7.3) that

$$h \quad = \quad O(\log(n/B)). \tag{7.8}$$

Our structure has two components:

- A prioritized structure on $\mathcal{A}$.
- On each $R[i]$ where $1 \leq i \leq h$, build a *top-f structure*, namely, the structure we just explained for answering queries with $k \leq f$. Since $|R[i]| \leq 12\lambda(n/(2^{i-1}f)) \ln n$, all these top-$f$ structures use in total

$$\begin{aligned} & O\left(\sum_{i=1}^{h} \mathcal{S}_{pri}\left(\frac{12\lambda \cdot n \ln n}{2^{i-1}f}\right)\right) \\ = \quad & O(\mathcal{S}_{pri}(n) + h) = O(\mathcal{S}_{pri}(n)) \end{aligned}$$

  space, where the derivation used the facts that (i) $\mathcal{S}_{pri}(n)$ is geometrically converging, (ii) $\mathcal{S}_{pri}(n)$ obviously needs to be $\Omega(n/B)$, and hence, by Equation (7.8), $h = O(\mathcal{S}_{pri}(n))$.

The total space occupied by our structure is therefore $O(\mathcal{S}_{pri}(n))$.

Now consider a top-k query q with $f < k \leq n$. First, if $k \geq n/2$, we answer it by simply scanning the entire $\mathcal{A}$ in $O(n/B) = O(k/B)$ I/Os. Next, we consider $k < n/2$.

Identify the smallest $i \in [1, h]$ such that $2^{i-1}f \geq k$. Fix the value of K to $2^{i-1}f$ in the rest of the section. Note that $k \leq K < 2k$. We then proceed as follows:

- If $|D(q)| \leq 4K$, we answer the query with cost monitoring through the prioritized structure on $\mathcal{A}$ in $\mathcal{Q}_{pri}(n) + O(K/B)$ I/Os.
- If $|D(q)| > 4K$, Lemma 7.2.2 indicates that $R[i](q)$ has size at least $\lceil 8\lambda \ln n \rceil$, and that the element e with weight rank $\lceil 8\lambda \ln n \rceil$ in $R[i](q)$ has weight rank between K and $4K$ in $\mathcal{A}(q)$.

  Retrieve e by issuing a top-f query on $R[i]$. By searching the top-f structure on $R[i]$, the query finishes in $O(\mathcal{Q}_{pri}(n) \log_{g\sqrt{B}} n)$ I/Os, as proved earlier. Extract from the prioritized structure on $\mathcal{A}$ the elements in $\mathcal{A}(q)$ whose weights are at least $w(e)$; this entails $\mathcal{Q}_{pri}(n) + O(K/B)$ I/Os. Finally, the query result can be produced with k-selection in $O(K/B)$ I/Os.

Overall, the query performs $O(\mathcal{Q}_{pri}(n) \log_{g\sqrt{B}} n + K/B)$ I/Os. This completes the proof of Theorem 4.4.4.

7.4 Remark

Open Problem: Is it possible to obtain a top-k structure with $\mathcal{S}_{top}(n) = O(\mathcal{S}_{pri}(n))$ and $\mathcal{Q}_{top}(n) = O(\mathcal{Q}_{pri}(n))$? This would have a strong implication, namely that the top-k reporting and the prioritized reporting are equivalent problems for any GIQ.

Appendix: Chernoff Bounds

Let $X_1, ..., X_n$ be independent Bernoulli variables such that $\mathbf{Pr}[X_i = 1] = p_i$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathbf{E}[X] = \sum_{i=1}^{n} p_i$. Then for any $\alpha \in (0, 1)$,

$$\mathbf{Pr}[X \leq (1 - \alpha)\mu] \quad \leq \quad e^{-\alpha^2 \mu/3}. \tag{7.9}$$

For any $\alpha \geq 2$,

$$\mathbf{Pr}[X \geq \alpha\mu] \quad \leq \quad e^{-\alpha\mu/6}. \tag{7.10}$$

The above inequalities can be found in many papers and textbooks, e.g., [89, 90]

Chapter 8

New Top-k GIQ structures

In this chapter we present new top-k structures for various GIQ problems. In the process, we prove Theorems 4.5.1–4.5.7. While those theorems were presented essentially in descending order of their importance to database query-retrieval, here we will prove them in a different order: from the least sophisticated to the most.

8.1 Top-k Interval Stabbing (Theorem 4.5.5)

We make the following observations, which, as we will see, lead to Theorem 4.5.5. The prioritized-reporting version of the problem has been studied by Tao [91] (where the version is called *ray stabbing*), who gave an $O(n/B)$-size structure that answers a query in $O(\log_B n + t/B)$ I/Os, and supports an update in $O(\log_B n)$ amortized I/Os. The max-reporting version has been studied by Agarwal et al. [64], who gave an $O(n/B)$-size structure that answers a query in $O(\log_B n)$ I/Os, and supports an update in $O(\log_B n)$ amortized I/Os.

First bullet of Theorem 4.5.5. To prove this, we will use our second reduction (Theorem 4.4.3), which requires a priortized-reporting structure and a max-reporting structure. Plugging in the bounds of these structures into Theorem 4.4.3, we obtain an EM structure of $O(n/B)$ space and $O(\log_B n + k/B)$ query time, both in expectation. The update time will be $O(\log_B n)$ amortized I/Os.

Second bullet of Theorem 4.5.5. To prove this, we will use our third reduction (Theorem 4.4.4), which requires only a priortized-reporting structure. Plugging in the bounds of the prioritized structure into Theorem 4.4.4, we obtain an EM structure of $O(n/B)$ space and $O(\log_B^2 n + k/B)$ query time, both in the worst-case.

8.2 Top-k Orthogonal Range Reporting (Thm. 4.5.1–4.5.3)

Proof of Theorem 4.5.1. We will use the second reduction (Theorem 4.4.3), which requires a priortized-reporting structure and a max-reporting structure. The prioritized-reporting structure is the external-memory priority search tree [92] which uses $O(\frac{n}{B})$ space and answers a query in $O(\log_B n + t/B)$ I/Os. The max-reporting structure can be obtained by augmenting the standard B-tree which uses $O(\frac{n}{B})$ space and answers a query in $O(\log_B n)$ I/Os. Plugging in these bounds into Theorem 4.4.3 leads to an EM structure which uses $O(\frac{n}{B})$ space and answers a top-k query in $O(\log_B n + k/B)$ I/Os (both in expectation). By setting B to be a constant, we obtain the bounds for the RAM structure.

Proof of Theorem 4.5.2. We will use the second reduction (Theorem 4.4.3), which requires a priortized-reporting structure and a max-reporting structure. The prioritized-reporting version of the problem has been studied by Afshani, Arge and Larsen [51] in the RAM model and by Rahul and Tao [5] in the EM model. The RAM structure uses $O(n\frac{\log n}{\log\log n})$ space and answers a query in $O(\log n + t)$ time. The EM structure uses $O(\frac{n}{B}\frac{\log n}{\log\log_B n}(\log\log B)^2)$ space and answers a query in $O(\log_B n + t/B)$ I/Os. The max-reporting version has been studied by Rahul and Tao [5]. The RAM structure uses $O(n\frac{\log n}{\log\log n})$ space and answers a query in $O(\log n)$ time. The EM structure uses $O(\frac{n}{B}\frac{\log n}{\log\log_B n})$ space and answers a query in $O(\log_B n)$ I/Os.

Now we plug these bounds into Theorem 4.4.3 to obtain a RAM structure of $O(n\frac{\log n}{\log\log n})$ space and $O(\log n + k)$ query time (both in expectation), and an EM structure of $O(\frac{n}{B}\frac{\log n}{\log\log_B n}(\log\log B)^2)$ space and $O(\log_B n + k/B)$ query time (both in expectation).

Proof of Theorem 4.5.3. We will use the first reduction (Theorem 4.4.1). For that we use the following structure: the classical d-dimensional range tree [58]. In $\mathbb{R}^d$ it

occupies $O(n \log^{d-1} n)$ space, handles an update in $O(\log^d n)$ amortized time, answers a counting query in $O(\log^d n)$ time and a reporting query in $O(\log^d n + t)$ time. This leads to a top-k structure of $O(n \log^d n)$ space, $O(\log^{d+1} n + k)$ query time and $O(\log^{d+1} n)$ amortized update time.

8.3 Top-k Point Enclosure (Theorem 4.5.6)

The prioritized-reporting version of the problem has been studied by Rahul [1], who gave a structure of $O(n \log^* n)$ size and $O(\log n \log \log n + t)$ query time.

Next, we explain how to solve the max-reporting version with a structure that uses $O(n \log n)$ space and answers a query in $O(\log n)$ time. Based on the above, the first and second bullets of Theorem 4.5.6 follow from Theorems 4.4.3 and Theorem 4.4.4, respectively.

1D Stabbing Max. Section 8.1 already mentioned a dynamic structure for solving the max-reporting version of interval stabbing. In fact, if the goal is to design a *static* structure, that problem can be settled with a very simple structure using $O(n)$ space and $O(\log n)$ time. Although this should be folklore, we give the details nonetheless because it will be helpful later.

Let D be a set of n weighted intervals in $\mathbb{R}$. The $2n$ endpoints of the intervals divide $\mathbb{R}$ into at most $2n+1$ disjoint subintervals. With each subinterval I, we associate the maximum weight of all the intervals in D that span I. Given a value $q \in \mathbb{R}$, a query returns the maximum weight of the intervals of D containing q. This is precisely the weight associated with the subinterval containing q. Finding the subinterval is essentially predecessor search, which can be carried out in $O(\log n)$ time by performing binary search on the endpoints.

2D Stabbing Max (Point Enclosure Max). Now we return to the max-reporting version of point enclosure. The input is a set D of n weighted axes-aligned rectangles. Create a segment tree T on the x-projections of those rectangles. For each node u of T, define D_u to be the set of segments assigned to u. Build a 1D stabbing max structure on D_u. The overall space is clearly $O(n \log n)$.

Given a point $q = (x, y)$, a query returns the maximum weight of the rectangles

of D containing q. To process the query, we descend along a root-to-leaf path Π of T according to x, and then, on each node $u \in \Pi$, issue a 1D stabbing max query on D_u with y. The final answer is the maximum of the results of these 1D queries. The algorithm takes $O(\log^2 n)$ time, which can be improved to $O(\log n)$ with fractional cascading [21] because, as mentioned earlier, each 1D query performs nothing but predecessor search on a sorted list.

8.4 Top-k 3D Dominance (Theorem 4.5.7)

The prioritized-reporting version of the problem has been studied by Afshani et al. [34] (where the version is called *4D dominance reporting*), who gave a structure with size $O(n \log n / \log \log n)$ and query time $O(\log^{1.5} n + t)$.

Next, we explain how to solve the max-reporting version with a structure that uses $O(n)$ space and answers a query in $O(\log^{1.5} n)$ time. Plugging in these bounds into Theorem 4.4.3 proves Theorem 4.5.7.

In this setting, D is a set of n weighted points in $\mathbb{R}^3$. Let $e_1, e_2, ..., e_n$ be the sequence of points in descending order of weight. With each point e_i, we associate a region ρ_i in $\mathbb{R}^3$ satisfying the following constraint: any point $q = (x, y, z)$ belongs to ρ_i if and only if e_i is the point with the maximum weight in $(-\infty, x] \times (-\infty, y] \times (-\infty, z]$. The region assignment below ensures the constraint for all points $e_i = (e_{ix}, e_{iy}, e_{iz})$:

- $\rho_1 = [e_{1x}, \infty) \times [e_{1y}, \infty) \times [e_{1z}, \infty)$.
- For $i \in [2, n]$:

$$\rho_i = [e_{ix}, \infty) \times [e_{iy}, \infty) \times [e_{iz}, \infty) \setminus \bigcup_{j=1}^{i-1} \rho_j.$$

Each non-empty region ρ_i is decomposed into axes-aligned disjoint cuboids by performing a *vertical decomposition*. If ρ_i has n_i vertices, then the number of cuboids in the decomposition of ρ_i will be $O(n_i)$. It can be verified [35] that $\sum_{i=1}^{n} n_i = O(n)$.

Therefore, the max reporting problem can be transformed to a point location problem: Given a query point q, find the cuboid (if any) containing q from a set of $O(n)$ disjoint axes-aligned cuboids. Rahul [1] presented a structure of size $O(n)$ to answer such a query in $O(\log^{1.5} n)$ time.

8.5 Top-k Halfspace Reporting: $d = 2$ (Theorem 4.5.4: 1st Bullet)

We will show:

- The prioritized-reporting version of the problem can be settled by an $O(n \log n)$-size structure that answers a query in $O(\log n + t)$ time.
- The max-reporting version can be settled by an $O(n)$-size structure that answers a query in $O(\log n)$ time.

Plugging in these results into Theorem 4.4.3 proves the first bullet of Theorem 4.5.4.

Prioritized Reporting. Chazelle et al. [93] settled the *original* 2D halfspace reporting problem. Specifically, they showed that n points in $\mathbb{R}^2$ can be stored in an $O(n)$-size structure such that, given a halfspace q, all the t input points falling in q can be reported in $O(\log n + t)$ time. Their query algorithm, in fact, starts with finding the predecessor of some query value (that depends on q) in a pre-computed list of real values. This accounts for the $O(\log n)$ term. Once the predecessor is found, the rest of the algorithm finishes in $O(1 + t)$ time.

Next, we leverage the above structure to tackle the prioritized-reporting version. In this setting, the input is a set D of n weighted points in $\mathbb{R}^2$. Create a balanced binary search tree T on their weights, with each weight stored in a leaf, which is associated with the corresponding point in D. For each node u of T, denote by D_u the set of points stored in the subtree of u. Create a halfspace reporting structure of [93] on D_u. The total space is $O(n \log n)$.

Given a halfspace q and a threshold τ, a query returns all the points $e \in D$ such that $e \in q$ and weight $w(e) \geq \tau$. We answer it as follows. First, collect the *canonical* set $U(\tau)$ of nodes $u_1, u_2, ..., u_m$ with the smallest m such that D_{u_1}, D_{u_2}, ..., D_{u_m} are disjoint, and their union equals $\{e \in D \mid w(e) \geq \tau\}$. It is rudimentary to find these $m = O(\log n)$ nodes in $O(\log n)$ time. Then, perform a halfspace reporting query using q on D_{u_i}, for each $i \in [1, m]$. The final answer is the union of the outputs of all these m queries.

As explained earlier, each halfspace reporting query spends $O(\log n)$ time on a predecessor search, which makes the total query time $O(\log^2 n + t)$. A standard application of fractional cascading reduces the time to $O(\log n + t)$.

Max Reporting. The input is again a set D of n weighted points. Given a halfspace q, a query returns the maximum weight of the points of D covered by q. By standard duality, we consider instead the following equivalent *stabbing max* problem. The input is a set D' of n weighted halfspaces in $\mathbb{R}^2$. The goal is to store D' in a data structure such that, given a point q' in $\mathbb{R}^2$, we can report efficiently the maximum weight of the halfspaces of D' containing q'. Below we describe an $O(n)$-size structure with $O(\log n)$ query time.

Using the idea in Section 8.4, we can transform the problem into a point location problem on a planar subdivision of complexity $O(n)$. Let $e'_1, e'_2, ..., e'_n$ be the halfspaces of D', in descending order of weight. For each e'_i, define a region ρ_i in $\mathbb{R}^2$ satisfying the following constraint: any point q belongs to ρ_i if and only if e'_i is the halfspace with the maximum weight among all the halfspaces containing q. The region assignment below ensures the constraint:

- $\rho_1 = e'_1$.
- For $i \in [2, n]$, $\rho_i = e'_i \setminus \bigcup_{j=1}^{i-1} \rho_j$.

Here ρ_1, ρ_2, ..., ρ_n are disjoint polygons such that the planar subdivision they induce has at most n vertices. To see this, imagine generating ρ_i in ascending order of i as follows. If e'_i falls entirely in $\bigcup_{j=1}^{i-1} \rho_j$, then e'_i introduces no vertex on the subdivision. Otherwise, at least one point p on the boundary line of e'_i must be outside $\bigcup_{j=1}^{i-1} \rho_j$. Walk from p along the boundary line towards one direction, and stop as soon as hitting the boundary line of any of the halfspaces already considered. The stopping point is a new vertex on the subdivision. Similarly, walking from p towards the other direction will determine another new vertex.

Given a query point q', it suffices to find the polygon of the subdivision containing q'. This can be done in $O(\log n)$ time with an $O(n)$-size structure [19].

8.6 Top-k Halfspace Reporting: $d \geq 4$ (Theorem 4.5.4: 2nd and 3rd Bullets)

The subsequent discussion demonstrates the power of Theorem 4.4.4 in showing the asymptotic equivalence between top-k reporting and prioritized reporting, when the query time is large.

RAM. We will show that the prioritized-reporting version of the problem can be settled by an $O(n \log n)$-size structure that answers a query in $\tilde{O}(n^{1-1/\lfloor d/2 \rfloor}) + O(t)$ time. Plugging this into Theorem 4.4.4 yields the second bullet of Theorem 4.5.4.

In the *original* halfspace reporting problem, we want to store n points in $\mathbb{R}^d$ in a structure such that, given any halfspace q in $\mathbb{R}^d$, all the input points falling in q can be reported efficiently. Afshani and Chan [94] gave a structure of $O(n)$ space and query time $\tilde{O}(n^{1-1/\lfloor d/2 \rfloor}) + O(t)$.

Recall that Section 8.5 presented a prioritized reporting structure in 2D space, where there is a 2D halfspace reporting structure on each D_u. To obtain a prioritized reporting structure for $\mathbb{R}^d$, we simply replace that 2D structure with the d-dimensional halfspace reporting structure of [94]. A prioritized query is answered in the way as described in Section 8.5, excluding the part about fractional cascading. The claimed space and query bounds follow from the same analysis as in Section 8.5.

EM. We will show that the prioritized-reporting version of the problem can be settled by an $O(n/B)$-size structure that answers a query in $O((n/B)^{1-1/\lfloor d/2 \rfloor + \epsilon} + t/B)$ time. Plugging this into Theorem 4.4.4 yields the third bullet of Theorem 4.5.4.

For the *original* halfspace reporting problem, Agarwal et al. [68] gave a structure of $O(n/B)$ space and query time $O((n/B)^{1-1/\lfloor d/2 \rfloor + \epsilon'} + t/B)$ for any arbitrarily small constant $\epsilon' > 0$, which we utilize below to design the required structure for prioritized reporting.

The input is a set D of n weighted points in $\mathbb{R}^d$, which we denote as $e_1, e_2, ..., e_n$ in descending order of weight. Set $f = (n/B)^{\epsilon/2}$. Build a B-tree T on the weights of the n points with leaf capacity B and internal fanout f. Store each point together with its weight in the corresponding leaf. For each node u of T, denote by D_u the set of points

stored in the subtree of u; we create a structure of [68] on D_u by setting $\epsilon' = \epsilon/2$. The overall space consumption is $O(n/B)$—noticing that T has $O(1)$ levels.

To answer a query with halfspace q and threshold τ, we collect the canonical set $U(\tau)$ of nodes $u_1, u_2, ..., u_m$ with the smallest m such that $D_{u_1}, ..., D_{u_m}$ are disjoint, and their union equals $\{e \in D \mid w(e) \geq \tau\}$. It is rudimentary to find these $m = O(f)$ nodes in $O(1 + f/B)$ I/Os. We then perform a halfspace reporting query using q on D_{u_i}, for all $i \in [1, m]$. The final answer is the union of the outputs of all these m queries. The query cost is

$$O\left(m \cdot (n/B)^{1-1/\lfloor d/2 \rfloor + \epsilon'} + t/B\right) = O\left((n/B)^{1-1/\lfloor d/2 \rfloor + \epsilon} + t/B\right) \text{ I/Os.}$$

Part III

Approximate Range Counting

Chapter 9

Approximate Range Counting

9.1 Problem statement

Let S be a set of n geometric objects in $\mathbb{R}^d$ which are segregated into disjoint groups (i.e., *colors*). Given a query $q \subseteq \mathbb{R}^d$, a color c *intersects (or, is present in)* q if any object in S of color c intersects q, and let k be the number of colors of S present in q. In the *approximate colored range-counting problem*, the task is to preprocess S into a data structure, so that for a query q, one can efficiently report the *approximate* number of colors present in q. Specifically, return any value in the range $[(1-\varepsilon)k, (1+\varepsilon)k]$, where $\varepsilon \in (0,1)$ is a pre-specified parameter.

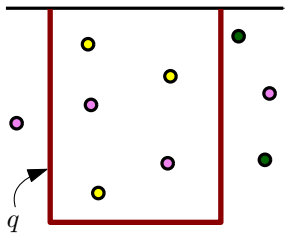


Figure 9.1: An instance of a colored setting.

Colored range searching and its related problems have been studied before [95, 96, 97, 81, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110]. They are known as GROUP-BY queries in the database literature. A popular variant is the *colored orthogonal range searching* problem: S is a set of n colored points in $\mathbb{R}^d$, and q is

an axes-aligned rectangle. As a motivating example for this problem, consider the following query: "How many countries have employees aged between X_1 and X_2 while earning annually more than Y dollars?". An employee is represented as a colored point $(age, salary)$, where the color encodes the country, and the query is the axes-aligned rectangle $[X_1, X_2] \times [Y, \infty)$.

9.2 Previous work and background

In the *standard* approximate range counting problem there are no colors. One is interested in the approximate number of objects intersecting the query. Specifically, if k is the number of objects of S intersecting q, then return a value in the range $[(1-\varepsilon)k, (1+\varepsilon)k]$.

ε-approximations. In the *additive-error ε-approximation*, a set $Z \subseteq S$ is picked such that, given a query q, we *only* inspect Z and return a value which lies in the range $[k - \varepsilon n, k + \varepsilon n]$. Vapnik and Chervonenkis [23] proved that a random sample Z of size $O(\frac{\delta}{\varepsilon^2} \log \frac{\delta}{\varepsilon})$ provides an ε-approximation with good probability, where δ is the VC-dimension (δ is usually a constant).

Relative (p, ε)-approximation. Har-Peled and Sharir [111] introduced the notion of *relative (p, ε)-approximation* for geometric settings. The goal is to pick a *small* set $Z \subset S$ which can be used to compute a relative approximation for queries with large value of k. Formally, given a parameter $p \in (0, 1)$, a set $Z \subset S$ is a relative (p, ε)-approximation if:

$$|Z \cap q| \cdot \frac{n}{|Z|} \in \begin{cases} [(1-\varepsilon)k, (1+\varepsilon)k] & \text{if } k \geq pn \\ [k - \varepsilon pn, k + \varepsilon pn] & \text{otherwise.} \end{cases}$$

Har-Peled and Sharir prove that a sample Z from S of size $O\left(\frac{1}{\varepsilon^2 p}\left(\delta \log \frac{1}{p} + \log \frac{1}{q}\right)\right)$ will succeed with probability at least $1 - q$.

Har-Peled and Sharir construct relative (p, ε)-approximations for point sets and half-spaces in $\mathbb{R}^d$, for $d \geq 2$, and use them to answer approximate counting for any query which contains more than pn points. A nice feature of these results is that they are *sensitive* to the value of k. Specifically, the larger the value of k is, the faster the query

is answered. The intuition is that the larger the value of k is, the larger is the error the query is allowed to make and hence, a smaller sample suffices. Even though relative (p, ε)-approximations give a relative approximation only for queries with large values of k, Aronov and Sharir [112], and Sharir and Shaul [113] incorporated them into data structures which give an approximate count for all values of k.

General reduction to companion problems. Aronov and Har-Peled [36], and Kaplan, Ramos and Sharir [37] presented general techniques to answer approximate range counting queries. In both instances, the authors reduce the task of answering an approximate counting query, into answering a few queries in data structures solving an easier *(companion)* problem. Aronov and Har-Peled's companion problem is the emptiness query, where the goal is to report whether $|S \cap q| = 0$. Specifically, assume that there is a data structure of size $S(n)$ which answers the emptiness query in $O(Q(n))$ time. Aronov and Har-Peled show that there is a data structure of size $O(S(n) \log n)$ which answers the approximate counting query in $O(Q(n) \log n)$ time (for simplicity we ignore the dependency on ε). Kaplan et al.'s companion problem is the range-minimum query, where each object of S has a weight associated with it and the goal is to report the object in $S \cap q$ with the minimum weight.

Even though the reductions of [36] and [37] seem different, there is an interesting discussion in Section 6 of [36] about the underlying "sameness" of both techniques.

Levels. Informally, for a set S of n objects, a *t-level* of S is a surface such that if a point q lies above (resp., on/below) the surface, then the number of objects of S containing q is $> t$ (resp., $\leq t$). Range counting can be reduced in some cases to deciding the level of a query point. Unfortunately, the complexity of a single level is not well understood. For example, for hyperplanes in the plane, the t-level has super-linear complexity $\Omega(n2^{\sqrt{\log t}})$ [114] in the worst-case (the known upper bound is $O(nt^{1/3})$ [115] and closing the gap is a major open problem). In particular, the prohibitive complexity of such levels makes them inapplicable for the approximate range counting problem, where one strives for linear (or near-linear) space data structures.

Shallow cuttings. A *t-level shallow cutting* is a set of simple cells, that lies strictly below the $2t$-level, and their union covers all the points below (and on) the t-level.

For many geometric objects in two and three dimensions, such t-shallow cuttings have $O(n/t)$ cells [116]. Using such cuttings leads to efficient data structures for approximate range counting. Specifically, one uses binary search on a "ladder" of approximate levels (realized via shallow cuttings) to find the approximation.

For halfspaces in $\mathbb{R}^3$, Afshani and Chan [88] avoid doing the binary search and find the two consecutive levels in optimal $O(\log \frac{n}{k})$ expected time. Later, Afshani, Hamilton and Zeh [117] obtained a worst-case optimal solution for many geometric settings. Interestingly, their results hold in the pointer machine model, the I/O-model and the cache-oblivious model. However, in the word-RAM model their solution is not optimal and the query time is $\Omega(\log\log U + (\log\log n)^2)$.

Specific problems. Approximate counting for orthogonal range searching in $\mathbb{R}^2$ was studied by Nekrich [109], and Chan and Wilkinson [118] in the word-RAM model. In this setting, the input set is points in $\mathbb{R}^2$ and the query is a rectangle in $\mathbb{R}^2$. A hyper-rectangle in $\mathbb{R}^d$ is $(d+k)$-*sided* if it is bounded on both sides in k out of the d dimensions and unbounded on one side in the remaining $d-k$ dimensions. Nekrich [109] presented a data structure for approximate colored 3-sided range searching in $\mathbb{R}^2$, where the input is points and the query is a 3-sided rectangle in $\mathbb{R}^2$. However, it has an approximation factor of $(4+\varepsilon)$, whereas we are interested in obtaining a tighter approximation factor of $(1+\varepsilon)$. To the best of our knowledge, this is the only work directly addressing an approximate colored counting query.

9.3 Motivation

Avoiding expensive counting structures. A search problem is $O(1)$-decomposable if given two disjoint sets of objects S_1 and S_2, the answer to $F(S_1 \cup S_2)$ can be computed in constant time, given the answers to $F(S_1)$ and $F(S_2)$ separately. This property is widely used in the literature [12] for counting in standard problems (going back to the work of Bentley and Saxe [119] in the late 1970s). For colored counting problems, however, $F(\cdot)$ is not $O(1)$-decomposable. If $F(S_1)$ (resp. $F(S_2)$) has n_1 (resp. n_2) colors, then this information is insufficient to compute $F(S_1 \cup S_2)$, as they might have common colors.

As a result, for *many exact* colored counting queries the known space and query time

bounds are expensive. For example, for colored orthogonal range searching problem in $\mathbb{R}^d$, existing structures use $O(n^d)$ space to achieve polylogarithmic query time [106]. Any substantial improvement in the preprocessing time *and* the query time would lead to a substantial improvement in the best exponent of matrix multiplication [106] (which is a major open problem). Similarly, counting structures for colored halfspace counting in $\mathbb{R}^2$ and $\mathbb{R}^3$ [100] are expensive.

Instead of an exact count, if one is willing to settle for an approximate count, then this work presents a data structure with $O(n \text{ polylog } n)$ space and $O(\text{polylog } n)$ query time.

Approximate counting at the speed of an emptiness query. In an emptiness query, the goal is to decide if $S \cap q$ is empty. The approximate counting query is at least as hard as the emptiness query: When $k = 0$ and $k = 1$, no error is tolerated. Therefore, a natural goal while answering approximate range counting queries is to match the bounds of its corresponding *emptiness query*.

9.4 Our results and techniques

9.4.1 Specific problems

The focus of this work is in building data structures for approximate colored counting queries, which exactly match or *almost* match the bounds of their corresponding emptiness problem.

3-sided rectangle stabbing in 2-d and related problems. In the colored interval stabbing problem, the input is n colored intervals with endpoints in $[U] = \{1, \ldots, U\}$, and the query is a point in $[U]$. We present a linear-space data structure which answers the approximate counting query in $O(\log \log U)$ time. The new data structure can be used to handle some geometric settings in 2-d: the *colored dominance search* (the input is a set of n points, and the query is a 2-sided rectangle) and the *colored* 3*-sided rectangle stabbing* (the input is a set of n 3-sided rectangles, and the query is a point). The results are summarized in Table 9.1.

Range searching in $\mathbb{R}^2$. The input is a set of n colored points in the plane. For 3-sided query rectangles, an *optimal* solution (in terms of n) for approximate counting is obtained. For 4-sided query rectangles, an *almost-optimal* solution for approximate counting is obtained. The size of our data structure is off by a factor of $\log\log n$ w.r.t. its corresponding emptiness structure which occupies $O(n\frac{\log n}{\log\log n})$ space and answers the emptiness query in $O(\log n)$ time [22]. The results are summarized in Table 9.1.

Dominance search in $\mathbb{R}^3$. The input is a set of n colored points in $\mathbb{R}^3$ and the query is a 3-sided rectangle in $\mathbb{R}^3$ (i.e., an octant). An almost-optimal solution is obtained requiring $O(n\log\log n)$ space and $O(\log n)$ time to answer the approximate counting query.

9.4.2 General reductions

We obtain two general reductions for solving approximate colored counting queries by reducing them to "easy" companion queries. However, in the interest of space, we included only Reduction-I in the thesis.

Reduction-I (Reporting + C-approximation). In the first reduction a colored approximate counting query is answered using two companion structures: (a) *reporting structure* (its objective is to report the k colors), and (b) *C-approximation structure* (its objective is to report any value z s.t. $k \in [z, Cz]$, where C is a constant). Significantly, unlike previous reductions [36, 37], there is *no asymptotic loss* of efficiency in space and query time bounds w.r.t. to the two companion problems.

Reduction-II (Only Reporting). The second reduction is a modification of the Aronov and Har-Peled [36] reduction. We present the reduction for the following reasons:

- Unlike reduction-I, this reduction is "easier" to use since it uses only the reporting structure and avoids the C-approximation structure.
- The analysis of Aronov and Har-Peled is slightly complicated because of their insistence on querying emptiness structures. We show that by using reporting structures the analysis becomes simpler. This reduction is useful when the reporting query is not significantly costlier than the emptiness query.

Dime--nsion	Input, Query	New Results	Previous Approx. Counting Results	Exact Counting Results	Model
1	intervals, point	S: n, Q: $\log\log U$	S: n, Q: $\log\log U+$ $(\log\log n)^2$	S: n, Q: $\log\log U+$ $\log_w n$	WR
2	points, 2-sided rectangle				
2	3-sided, rect., point	Theorem 10.0.1	Remark 1	Remark 2	
2	points, 3-sided rectangle	S: n, Q: $\log n$ Theorem 12.0.1(A)	S: $n\log^2 n$, Q: $\log^2 n$ Remark 3	not studied	PM
2	points, 4-sided rectangle	S: $n\log n$, Q: $\log n$ Theorem 12.0.1(B)	S: $n\log^3 n$, Q: $\log^2 n$ Remark 3	S: $n^2\log^6 n$, Q: $\log^7 n$ Kaplan et al. [106]	PM
3	points, 3-sided rectangle	S: $n\log^* n$, Q: $\log n\cdot\log\log n$	S: $n\log^2 n$, Q: $\log^2 n$	not studied	PM

Table 9.1: A summary of the results obtained for several approximate colored counting queries. To avoid clutter, the $O(\cdot)$ symbol and the dependency on ε is not shown in the space and the query time bounds. For the second column in the table, the first entry is the input and the second entry is the query. For each results column in the table, the first entry is the space occupied by the data structure and the second entry is the time taken to answer the query. WR denotes the word-RAM model and PM denotes the pointer machine model.

9.4.3 Our techniques

The results are obtained via a non-trivial combination of several techniques. For example, (a) new reductions from colored problems to standard problems, (b) obtaining a linear-space data structure by performing random sampling on a super-linear-size data structure, (c) refinement of path-range trees of Nekrich [109] to obtain an optimal data structure for C-approximation of colored 3-sided range search in $\mathbb{R}^2$, and (d) *random sampling on colors* to obtain the two general reductions.

In addition, we introduce *nested shallow cuttings* for 3-sided rectangles in 2-d. The

idea of using a hierarchy of cuttings (or samples) is, of course, not new. However, for this specific setting, we get a hierarchy where there is no penalty for the different levels being compatible with each other. Usually, cells in the lower levels have to be clipped to cells in the higher levels of the hierarchy, leading to a degradation in performance. In our case, however, cells at lower levels are fully contained in the cells at level above it.

Organization. To keep the thesis concise we have only included a few solutions. In Chapter 10, we present a solution to the colored 3-sided rectangle stabbing in 2-d problem. In Chapter 11, reduction-I is described. Finally, in Chapter 12, the application of reduction-I to colored orthogonal range search in 2-d is shown.

Chapter 10

Nested Shallow Cuttings

The goal of this chapter is to prove the following theorem.

Theorem 10.0.1. *Consider the following three colored geometric settings:*

1. **Colored interval stabbing in 1-d**, *where the input is a set S of n colored intervals in one-dimension and the query q is a point. The endpoints of the intervals and the query point lie on a grid $[U]$.*

2. **Colored dominance search in 2-d**, *where the input is a set S of n colored points in 2-d and the query q is a quadrant of the form $[q_x, \infty) \times [q_y, \infty)$. The input points and the point (q_x, q_y) lie on a grid $[U] \times [U]$.*

3. **Colored 3-sided rectangle stabbing in 2-d**, *where the input is a set S of n colored 3-sided rectangles in 2-d and the query q is a point. The endpoints of the rectangles and the query point lie on a grid $[U] \times [U]$.*

Then there exists an $O_\varepsilon(n)$ size word-RAM data structure which can answer an approximate counting query for these three settings in $O_\varepsilon(\log \log U)$ time. The notation $O_\varepsilon(\cdot)$ hides the dependency on ε.

Our strategy for proving this theorem is the following: In Section 10.1, we present a transformation of these three colored problems to the *standard* 3-sided rectangle stabbing in 2-d problem. Then in Section 10.2, we construct nested shallow cuttings and use them to solve the standard 3-sided rectangle stabbing in 2-d problem.

10.1 Transformation to a standard problem

From now on the focus will be on colored 3-sided rectangle stabbing in 2-d problem, since the geometric setting of (1) and (2) in Theorem 10.0.1 are its special cases. We present a transformation of the colored 3-sided rectangle stabbing in 2-d problem to the *standard* 3-sided rectangle stabbing in 2-d problem.

Let $S_c \subseteq S$ be the set of 3-sided rectangles of a color c. In the preprocessing phase, we perform the following steps: (1) Construct a union of the rectangles of S_c. Call it $\mathcal{U}(S_c)$. (2) The vertices of $\mathcal{U}(S_c)$ include original vertices of S_c and some new vertices. Perform a *vertical decomposition* of $\mathcal{U}(S_c)$ by shooting a vertical ray upwards from every *new* vertex of $\mathcal{U}(S_c)$ till it hits $+\infty$. This leads to a decomposition of $\mathcal{U}(S_c)$ into $\Theta(|S_c|)$ pairwise-disjoint 3-sided rectangles. Call these new set of rectangles $\mathcal{N}(S_c)$.

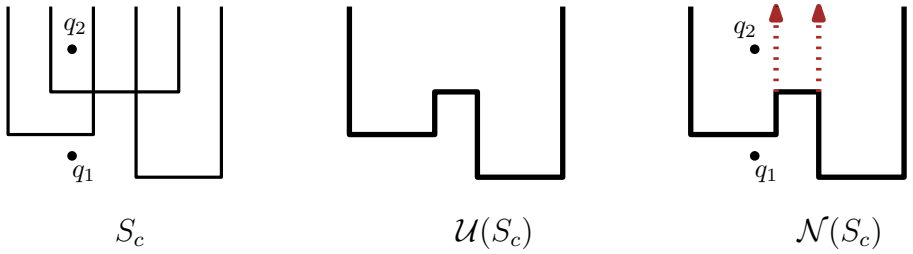


Figure 10.1: Transformation to a standard problem.

Given a query point q, we can make the following two observations:

- If $S_c \cap q = \emptyset$, then $\mathcal{N}(S_c) \cap q = \emptyset$. See query point q_1 in the above figure.
- If $S_c \cap q \neq \emptyset$, then exactly one rectangle in $\mathcal{N}(S_c)$ is stabbed by q. See query point q_2 in the above figure.

Let $\mathcal{N}(S) = \bigcup_{\forall c} \mathcal{N}(S_c)$, and clearly, $|\mathcal{N}(S)| = O(n)$. Therefore, the colored 3-sided rectangle stabbing in 2-d problem on S has been reduced to the *standard* 3-sided rectangle stabbing in 2-d problem on $\mathcal{N}(S)$.

10.2 Standard 3-sided rectangle stabbing in 2-d

In this section we will prove the following lemma.

Lemma 10.2.1. (Standard 3-sided rectangle stabbing in 2-d.) *Given a set S of n uncolored 3-sided rectangles of the form $[x_1, x_2] \times [y, \infty)$ whose endpoints lie on a grid $[U] \times [U]$, and a query q which is a point, there exists a data structure of size $O_\varepsilon(n)$ which can answer an approximate counting query for this geometric setting in $O_\varepsilon(\log \log U)$ time.*

By a standard rank-space reduction, the rectangles of S can be projected to a $[2n] \times [n]$ grid: Let S_x (resp., S_y) be the list of the $2n$ vertical (resp., n horizontal) sides of S in increasing order of their x- (resp., y-) coordinate value. Then each rectangle $r = [x_1, x_2] \times [y, \infty) \in S$ is projected to a rectangle $[rank(x_1), rank(x_2)] \times [rank(y), \infty)$, where $rank(x_i)$ (resp., $rank(y)$) is the index of x_i (resp., y) in the list S_x (resp., S_y). Given a query point $q \in [U] \times [U]$, we can use the van Emde Boas structure [120] to perform a predecessor search on S_x and S_y in $O(\log \log U)$ time to find the position of q on the $[2n] \times [n]$ grid. Now we will focus on the new setting and prove the following result.

Lemma 10.2.2. *For the standard 3-sided rectangle stabbing in 2-d problem, consider a setting where the rectangles have endpoints lying on a grid $[2n] \times [n]$. Then there exists a data structure of size $O_\varepsilon(n)$ which can answer the approximate counting query in $O_\varepsilon(1)$ time.*

10.2.1 Nested shallow cuttings

To prove Lemma 10.2.2, we will first construct shallow cuttings for 3-sided rectangles in 2-d. Unlike the general class of shallow cuttings, the shallow cuttings that we construct for 3-sided rectangles will have a stronger property of cells in the lower level lying completely inside the cells of a higher level.

Lemma 10.2.3. *Let S be a set of 3-sided rectangles (of the form $[x_1, x_2] \times [y, \infty)$) whose endpoints lie on a $[2n] \times [n]$ grid. A t-*level shallow cutting *of S produces a set $\mathcal{C}$ of interior-disjoint 3-sided rectangles/cells of the form $[x_1, x_2] \times (-\infty, y]$. There exists a set $\mathcal{C}$ with the following three properties:*

1. *$|\mathcal{C}| = 2n/t$.*
2. *If q does not lie inside any of the cell in $\mathcal{C}$, then $|S \cap q| \geq t$.*

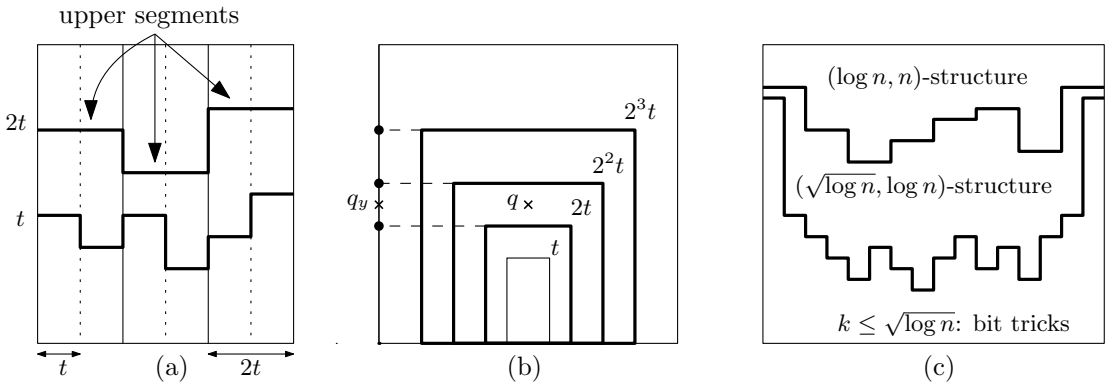


Figure 10.2: (a) A portion of the t-level and $2t$-level is shown. Notice that by our construction, each cell in the t-level is contained inside a cell in the $2t$-level. (b) A cell in the t-level and the set $\mathcal{C}_r$ associated with it. (c) A high-level summary of our data structure.

3. Each cell in $\mathcal{C}$ intersects at most $2t$ rectangles of S.

Proof. Partition the plane into $\frac{2n}{t}$ vertical slabs, such that t vertical lines of S lie in each slab, i.e., each slab has a width of t. See Figure 10.2(a). Consider a slab $s = [x_1, x_2] \times (-\infty, +\infty)$. Among all the rectangles of S which completely span the slab s, let y_t be the y-coordinate of the rectangle with the t-th smallest y-coordinate. If less than t segments of S span slab s, then set $y_t := +\infty$. Let the *upper segment* of the slab s be the horizontal segment $[x_1, x_2] \times [y_t]$. Each slab contributes a cell $[x_1, x_2] \times (-\infty, y_t]$ to set $\mathcal{C}$. See Figure 10.2(a).

Property 1 is easy to verify, since $\frac{2n}{t}$ slabs are constructed. To prove Property 2, consider a point q which lies in slab s but does not lie in the cell $[x_1, x_2] \times (-\infty, y_t]$. This implies that there are at least t rectangles of S which contain q, and hence, $|S \cap q| \geq t$. To prove Property 3, consider a cell r and its corresponding slab s. The rectangles of S which intersect r either span the slab s or partially span the slab s. By our construction, there can be at most t rectangles of S of each type. □

Observation 3. (Nested Property) *Let t and i be integers. Consider a t-level and a 2^it-level shallow cutting. By our construction, each cell in 2^it-level contains exactly 2^i cells of the t-level. More importantly, each cell in the t-level is contained inside a single cell of 2^it-level (see Figure 10.2(a)).*

10.2.2 Data structure

Now we will use nested shallow cuttings to find a constant-factor approximation for the 3-sided rectangle stabbing in 2-d problem. In [117], the authors show how to convert a constant-factor approximation into a $(1+\varepsilon)$-approximation for this geometric setting. The solution is based on (t, t')-*level-structure* and $(\leq \sqrt{\log n})$-*level shared table.*

(t, t')-level structure. Let i, t and t' be integers s.t. $t' = 2^i t$. If $q(q_x, q_y)$ lies between the t-level and the t'-level cutting of S, then a (t, t')-level-structure will answer the approximate counting query in $O(1)$ time and occupy $O\left(n + \frac{n}{t}\log t'\right)$ space.

Structure. Construct a shallow cutting of S for levels $2^j t, \forall j \in [0, i]$. For each cell, say r, in the t-level we do the following: Let $\mathcal{C}_r$ be the set of cells from the $2^1 t, 2^2 t, 2^3 t, \ldots, 2^i t$-level, which contain r (Observation 3 guarantees this property). Now project the upper segment of each cell of $\mathcal{C}_r$ onto the y-axis (each segment projects to a point). Based on the y-coordinates of these $|\mathcal{C}_r|$ projected points build a fusion-tree [9]. Since there are $O(n/t)$ cells in the t-level and $|\mathcal{C}_r| = O(\log t')$, the total space occupied is $O(\frac{n}{t}\log t')$. See Figure 10.2(b).

Query algorithm. Since $q_x \in [2n]$, it takes $O(1)$ time to find the cell r of the t-level whose x-range contains q_x. If the predecessor of q_y in $\mathcal{C}_r$ belongs to the $2^j t$-level, then $2^j t$ is a constant-factor approximation of k. The predecessor query also takes $O(1)$ time.

$(\leq \sqrt{\log n})$-level shared table. Suppose q lies in a cell in the $\sqrt{\log n}$-level shallow cutting of S. Then constructing the $(\leq \sqrt{\log n})$-level shared table will answer the exact counting query in $O(1)$ time. We will need the following lemma.

Lemma 10.2.4. *For a cell c in the $\sqrt{\log n}$-level shallow cutting of S, its* conflict list S_c *is the set of rectangles of S intersecting c. Although the number of cells in the $\sqrt{\log n}$-level is $O\left(\frac{n}{\sqrt{\log n}}\right)$, the number of combinatorially "different" conflict lists is merely $O(\sqrt{n})$.*

Proof. Consider any set S_c from the shallow cutting. By a standard rank-space reduction the endpoints of S_c will lie on a $[2|S_c|] \times [|S_c|]$ grid. Any set S_c on the $[2|S_c|] \times [|S_c|]$ grid can be *uniquely* represented using $O(|S_c| \log |S_c|) = O(\sqrt{\log n} \log\log n)$ bits as follows: (a) assign a *label* to each rectangle, and (b) write down the label of each rectangle

in increasing order of their y-coordinates. The label for a rectangle $[x_1, x_2] \times [y, \infty)$ will be "x_1x_2" which requires $O(\log \log n)$ bits. The number of combinatorially different conflict lists which can be represented using $O(\sqrt{\log n} \log \log n)$ bits is bounded by $2^{O(\sqrt{\log n} \log \log n)} = O(n^{\delta})$, for an arbitrarily small $\delta < 1$. We set $\delta = 1/2$. □

Shared table. Construct a $\sqrt{\log n}$-level shallow cutting of S. For each cell c, perform a rank-space reduction of its conflict list S_c. Collect the combinatorially different conflict lists. On each conflict list, the number of combinatorially different queries will be only $O(|S_c|^2) = O(\log n)$. In a lookup table, for each pair of (S_c, q) we store the exact value of $|S_c \cap q|$. The total number of entries in the lookup table is $O(n^{1/2} \log n)$.

Query algorithm. Given a query $q(q_x, q_y)$, the following three $O(1)$ time operations are performed: (a) Find the cell c in the $\sqrt{\log n}$-level which contains q. If no such cell is found, then stop the query and conclude that $k \geq \sqrt{\log n}$. (b) Otherwise, perform a rank-space reduction on q_x and q_y to map it to the $[2|S_c|] \times [|S_c|]$ grid. Since, $|S_c| = O(\sqrt{\log n})$, we can build fusion trees [9] on S_c to perform the rank-space reduction in $O(1)$ time. (c) Finally, search for (S_c, q) in the lookup table and report the exact count.

Final structure. At first thought, one might be tempted to construct a $(0, n)$-level-structure. However, that would occupy $O(n \log n)$ space. The issue is that the (t, t')-level structure requires super-linear space for small values of t. Luckily, the $(\leq \sqrt{\log n})$-level shared table will efficiently handle the small values of t.

Therefore, the strategy is to construct the following: (a) a $(\leq \sqrt{\log n})$-level shared table, (b) a $(\sqrt{\log n}, \log n)$-level-structure, and (c) a $(\log n, n)$-level-structure. Now, the space occupied by all the three structures will be $O(n)$. See Figure 10.2(c) for a summary of our data structure.

Remark 1. For the standard 3-sided rectangle stabbing in 2-d problem, a simple binary search on the levels leads to a linear-space data structure with a query time of $O_{\varepsilon}(\log \log U + (\log \log n)^2)$. The technique of Afshani *et al.* [117] can be used to answer this approximate counting query. However, their analysis works well for structures with query time of the form $\log n$ or $\log_B n$, but breaks down for structures with query time of the form $\log \log n$.

Remark 2. If we want an exact count for the standard 3-sided rectangle stabbing in 2-d problem, then the problem can be reduced to exact counting for standard dominance search in 2-d [52]. Jaja *et al.* [55] present a linear-space structure which can answer the exact counting for dominance search in 2-d in $O_\varepsilon(\log\log U + \log_w n)$ time.

Chapter 11

A General Reduction

Given a colored reporting structure and a colored C-approximation structure, we present a general reduction to obtain a colored $(1+\varepsilon)$-approximation structure with no additional loss of efficiency. We need a few definitions before stating the theorem. A geometric setting is *polynomially bounded* if there are only $n^{O(1)}$ possible outcomes of $S \cap q$, over all possible values of q. For example, in $1d$ orthogonal range search on n points, there are only $\Theta(n^2)$ possible outcomes of $S \cap q$. A function $f(n)$ is *converging* if $\sum_{i=0}^{t} n_i = n$, then $\sum_{i=0}^{t} f(n_i) = O(f(n))$. For example, it is easy to verify that $f(n) = n \log n$ is converging.

Theorem 11.0.1. *For a colored geometric setting, assume that we are given the following two structures:*

- *a colored reporting structure of $\mathcal{S}_{rep}(n)$ size which can solve a query in $O(\mathcal{Q}_{rep}(n)+\kappa)$ time, where κ is the output-size, and*
- *a colored C-approximation structure of $\mathcal{S}_{capp}(n)$ size which can solve a query in $O(\mathcal{Q}_{capp}(n))$ time.*

We also assume that: (a) $\mathcal{S}_{rep}(n)$ and $\mathcal{S}_{capp}(n)$ are converging, and (b) the geometric setting is polynomially bounded. Then we can obtain a $(1+\varepsilon)$-approximation using a structure that requires $\mathcal{S}_{\varepsilon app}(n)$ space and $\mathcal{Q}_{\varepsilon app}(n)$ query time, such that

$$\mathcal{S}_{\varepsilon app}(n) = O(\mathcal{S}_{rep}(n) + \mathcal{S}_{capp}(n)) \tag{11.1}$$

$$\mathcal{Q}_{\varepsilon app}(n) = O\left(\mathcal{Q}_{rep}(n) + \mathcal{Q}_{capp}(n) + \varepsilon^{-2} \cdot \log n\right). \tag{11.2}$$

11.1 Refinement Structure

The goal of a refinement structure is to convert a constant-factor approximation of k into a $(1+\varepsilon)$-approximation of k.

Lemma 11.1.1. (Refinement structure) *Let $\mathcal{C}$ be the set of colors in set S, and $\mathcal{C} \cap q$ be the set of colors in $\mathcal{C}$ present in q. For a query q, assume we know that:*

- $k = |\mathcal{C} \cap q| = \Omega(\varepsilon^{-2} \log n)$, *and*
- $k \in [z, Cz]$, *where z is an integer.*

Then there is a refinement structure of size $O\left(\mathcal{S}_{rep}\left(\frac{\varepsilon^{-2} n \log n}{z}\right)\right)$ which can report a value $\tau \in [(1-\varepsilon)k, (1+\varepsilon)k]$ in $O(\mathcal{Q}_{rep}(n) + \varepsilon^{-2} \log n)$ time.

The following lemma states that sampling colors (instead of input objects) is a useful approach to build the refinement structure.

Lemma 11.1.2. *Consider a query q which satisfies the two conditions stated in Lemma 11.1.1. Let c_1 be a sufficiently large constant and c be another constant s.t. $c = \Theta(c_1 \log e)$. Choose a random sample R where each color in $\mathcal{C}$ is picked independently with probability $M = \frac{c_1 \varepsilon^{-2} \log n}{z}$. Then with probability $1 - n^{-c}$ we have $\left|k - \frac{|R \cap q|}{M}\right| \leq \varepsilon k$.*

Proof. For each of the k colors which are present in q, define an indicator variable X_i. Set $X_i = 1$, if the corresponding color is in the random sample R. Otherwise, set $X_i = 0$. Then $|R \cap q| = \sum_{i=1}^{k} X_i$ and $E[|R \cap q|] = k \cdot M$. By Chernoff bound (see Appendix of Chapter 7),

$$\begin{aligned}\mathbf{Pr}\left[\left||R \cap q| - E[|R \cap q|]\right| > \varepsilon \cdot E[|R \cap q|]\right] &< \exp\Big(-\varepsilon^2 E[|R \cap q|]\Big) \\ &< \exp\left(-\varepsilon^2 \cdot kM\right) < \exp\left(-\varepsilon^2 zM\right) < \exp\left(-c_1 \log n\right) \leq \frac{1}{n^c}\end{aligned}$$

Therefore, with high probability $\left||R \cap q| - kM\right| \leq \varepsilon \cdot kM$. □

Lemma 11.1.3. (Finding a suitable R) *Pick a random sample R as defined in Lemma 11.1.2. Let n_R be the number of objects of S whose color belongs to R. We say R is* suitable *if it satisfies the following two conditions:*

- $\left|k - \frac{|R \cap q|}{M}\right| \leq \varepsilon k$ *for all queries which have* $k = \Omega(\varepsilon^{-2} \log n)$.
- $n_R \leq 10nM$. *This condition is needed to bound the size of the data structure.*

A suitable R *always exists.*

Proof. Let n^α be the number of combinatorially different queries q on the set S. From Lemma 11.1.2, by setting $c = \alpha + 1$, we can conclude that $\tau \longleftarrow \frac{|R \cap q|}{M}$ will lie in the range $[(1-\varepsilon)k, (1+\varepsilon)k]$ with probability at least $1 - 1/n^{\alpha+1}$. By the standard union bound, it implies that the probability of the random sample R failing for any query is at most $1/n^{\alpha+1} \times n^\alpha = 1/n$.

Next, it is easy to observe that the *expected* value of n_R is nM: Let n_c be the number of objects of S having color c. Then $\mathbf{E}[n_R] = \sum_{\forall c} n_c \cdot M = nM$. By Markov's inequality, the probability of n_R being larger than $10nM$ is less than or equal to $1/10$. By union bound, R will be not be suitable with probability $\leq 1/n + 1/10$. Therefore, with probability $\geq 9/10 - 1/n$, R will be suitable and hence, we are done. □

Refinement structure and query algorithm. In the preprocessing stage pick a random sample $R \subseteq \mathcal{C}$ as stated in Lemma 11.1.2. If the sample R is *not suitable*, then discard R and re-sample, till we get a suitable sample. Based on all the objects of S whose color belongs to R, build a colored reporting structure. Given a query q, the colored reporting structure is queried to compute $|R \cap q|$. We report $\tau \longleftarrow (|R \cap q|/M)$ as the final answer. The query time is bounded by $O(\mathcal{Q}_{rep}(n) + \varepsilon^{-2} \log n)$, since by Lemma 11.1.2, $|R \cap q| \leq (1+\varepsilon) \cdot kM = O(\varepsilon^{-2} \log n)$. This finishes the description of the refinement structure.

11.2 Overall solution

Data structure. The data structure consists of the following three components:

1. *Reporting structure.* Based on the set S we build a colored reporting structure. This occupies $O(\mathcal{S}_{rep}(n))$ space.
2. $\sqrt{C}$*-approximation structure.* Based on the set S we build a $\sqrt{C}$-approximation structure. The choice of $\sqrt{C}$ will become clear in the analysis. This occupies $O(\mathcal{S}_{capp}(n))$ space.

3. *Refinement structures.* Build the refinement structure of Lemma 11.1.1 for the values $z = (\sqrt{C})^i \cdot \varepsilon^{-2} \log n, \forall i \in \left[0, \log_{\sqrt{C}} \left(\lceil \varepsilon^2 n \rceil\right)\right]$. The total size of all the refinement structures will be $\sum O\left(\mathcal{S}_{rep}(nM)\right) = O(\mathcal{S}_{rep}(n))$, since $\mathcal{S}_{rep}(\cdot)$ is converging and $\sum nM = O(n)$. Note that our choice of z ensures that the size of the data structure is independent of ε.

Query algorithm. The query algorithm performs the following steps:

1. Given a query object q, the colored reporting structure reports the colors present in $S \cap q$ till all the colors have been reported or $\varepsilon^{-2} \log n + 1$ colors have been reported. If the first event happens, then the exact value of k is reported. Otherwise, we conclude that $k = \Omega(\varepsilon^{-2} \log n)$. This takes $O(\mathcal{Q}_{rep}(n) + \varepsilon^{-2} \log n)$ time.

2. If $k > \varepsilon^{-2} \log n$, then

   (a) First, query the $\sqrt{C}$-approximation structure. Let k_a be the $\sqrt{C}$-approximate value returned s.t. $k \in [k_a, \sqrt{C} k_a]$. This takes $O(\mathcal{Q}_{capp}(n))$ time.

   (b) Then query the refinement structure with the largest value of z s.t. $z \leq k_a \leq \sqrt{C} z$. It is trivial to verify that $k \in [z, Cz]$. This takes $O(\mathcal{Q}_{rep}(n) + \varepsilon^{-2} \log n)$ time.

11.3 Open problem

We do not discuss the preprocessing time in this chapter. It is not known how to verify efficiently if a sample R is "suitable". It would be interesting to find a solution which performs better than the naïve approach of *manually* verifying if R is suitable for every possible query.

Chapter 12

Application of the General Reduction

We illustrate an application of Reduction-I by studying the approximate colored counting query for orthogonal range search in $\mathbb{R}^2$.

Theorem 12.0.1. *Consider the following two problems:*

A) **Colored 3-sided range search in $\mathbb{R}^2$.** *In this setting, the input set S is n colored points in $\mathbb{R}^2$ and the query q is a 3-sided rectangle in $\mathbb{R}^2$. There is a data structure of $O(n)$ size which can answer the approximate colored counting query in $O(\varepsilon^{-2}\log n)$ time. This pointer machine structure is optimal in terms of n.*

B) **Colored 4-sided range search in $\mathbb{R}^2$.** *In this setting, the input set S is n colored points in $\mathbb{R}^2$ and the query q is a 4-sided rectangle in $\mathbb{R}^2$. There is a data structure of $O(n\log n)$ size which can answer the approximate colored counting query in $O(\varepsilon^{-2}\log n)$ time.*

12.1 Colored 3-sided range search in $\mathbb{R}^2$

We use the framework of Theorem 11.0.1 to prove the result of Theorem 12.0.1(A). For this geometric setting, a colored reporting structure with $\mathcal{S}_{rep} = n$ and $\mathcal{Q}_{rep} = \log n$ is already known [110]. The path-range tree of Nekrich [109] gives a $(4+\varepsilon)$-approximation,

but it requires super-linear space. The C-approximation structure presented in this section is a refinement of the path-range tree for the pointer machine model.

Lemma 12.1.1. *For the colored 3-sided range search in $\mathbb{R}^2$ problem, there is a C-approximation structure which requires $O(n)$ space and answers a query in $O(\log n)$ time.*

We prove Lemma 12.1.1 in the rest of this section.

12.1.1 Reduction to 5-sided rectangle stabbing in $\mathbb{R}^3$

In this subsection we present a reduction of colored 3-sided range search in $\mathbb{R}^2$ problem to the 5-sided rectangle stabbing problem in $\mathbb{R}^3$. Let S be a set of n colored points lying in $\mathbb{R}^2$. Let $S_c \subseteq S$ be the set of points of color c. For each color c which has at least one point inside $q = [x_1, x_2] \times [y_1, \infty)$, the objective is to identify the topmost point (in terms of y-coordinate) among $S_c \cap q$. Consider a point $p(p_x, p_y) \in S_c$. Starting from the x-coordinate value p_x, we walk to the left (resp. right) along the x-axis till we find the first point $p^l(p^l_x, p^l_y) \in S_c$ (resp. $p^r(p^r_x, p^r_y) \in S_c$) which has a higher y-coordinate value than p. (Conceptually imagine two dummy points at $(+\infty, +\infty)$ and $(-\infty, +\infty)$ to ensure that p^l and p^r always exist). Now we make the following important observation.

Observation 4. *A point $p \in S_c$ will be the topmost point in $S_c \cap q$ iff (1) p lies inside q, and (2) p_r and p_l do not lie inside q. In other words, $p \in S_c$ will be the topmost point in $S_c \cap q$ iff $(x_1, x_2, y_1) \in [p^l_x, p_x] \times [p_x, p^r_x] \times (-\infty, p_y]$.*

Figure 12.1 is an illustration of the above observation. Based on the above observation, we perform the following transformation: Each point $p \in S$ is transformed into a 5-sided rectangle $[p^l_x, p_x] \times [p_x, p^r_x] \times (-\infty, p_y]$. The query rectangle $q = [x_1, x_2] \times [y_1, \infty)$ is transformed into a point $q'(x_1, x_2, y_1) \in \mathbb{R}^3$. Now we can observe that (i) If a color c has at least one point inside q, then exactly one of its transformed rectangle will contain q', and (ii) If a color c has no point inside q, then none of its transformed rectangles will contain q'.

12.1.2 Interval tree

Our solution is based on an interval tree and we will need the following fact about it.

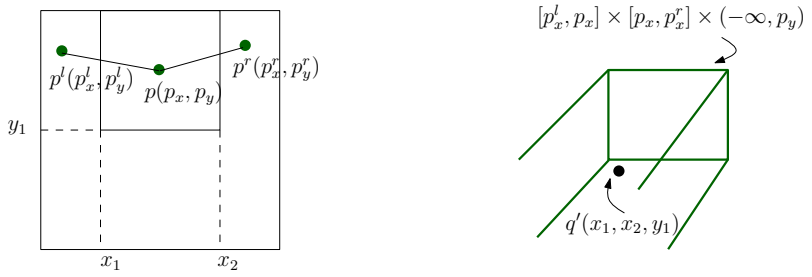


Figure 12.1: Reduction from colored 3-sided range search in $\mathbb{R}^2$ problem to the 5-sided rectangle stabbing problem in $\mathbb{R}^3$.

Lemma 12.1.2. *Using interval trees, a query on $(3+t)$-sided rectangles in $\mathbb{R}^3$ can be broken down into $O(\log n)$ queries on $(2+t)$-sided rectangles in $\mathbb{R}^3$. Here $t \in [1, 3]$.*

Proof. Let R be a set of n $(3+t)$-sided rectangles. We build an interval tree IT as follows: W.l.o.g., assume that the rectangles are bounded along the x-axis. Let h be a plane perpendicular to the x-axis such that there are equal number of endpoints of R on each side of the plane. The splitting halfplane h is stored at the root of IT and the two subtrees are built recursively. In general, $h(v)$ is the splitting halfplane stored at a node $v \in IT$. A rectangle $r \in R$ is stored at the highest node v s.t. r intersects $h(v)$. Let R_v be the set of rectangles stored at a node v. Each rectangle in $r \in R_v$ is split by $h(v)$ into two rectangles r^- and r^+. Define $R_v^- := \bigcup_{r \in R_v} r^-$ and $R_v^+ := \bigcup_{r \in R_v} r^+$.

Given a query point q, trace a path Π of length $O(\log n)$ from the root to a leaf node corresponding to q. For a node $v \in \Pi$, if q lies to the left (resp., right) of $h(v)$, then answering a query on $R_v \cap q$ is equivalent to answering it on $R_v^- \cap q$ (resp., $R_v^+ \cap q$), and we can treat R_v^- (resp., R_v^+) as $(2+t)$-sided rectangles in $\mathbb{R}^3$, since $h(v)$ is effectively $+\infty$ (resp., $-\infty$). □

12.1.3 Initial structure

Lemma 12.1.3. *For the colored 3-sided range search in $\mathbb{R}^2$ problem, there is a 2-approximation structure which requires $O(n)$ space and answers a query in $O(\log^3 n)$ time.*

Proof. We will use the reduction shown in subsection 12.1.1. For brevity, we will refer to 5-sided rectangle stabbing problem as 5-sided RSP. There is a simple linear-size data

structure which reports in $O(\log^3 n)$ time a 2-approximation for the 5-sided RSP: By inductively applying Lemma 12.1.2 twice, we can decompose 5-sided RSP to $O(\log^2 n)$ 3-sided RSPs. For 3-sided RSP, there is a linear-size structure of which reports a 2-approximation in $O(\log n)$ time [117]. By using this structure the 5-sided RSP can be solved in $O(\log^3 n)$ time. □

12.1.4 Final structure

Now we will present the optimal C-approximation structure of Lemma 12.1.1.

Structure. Sort the points of S based on their x-coordinate value and divide them into buckets containing $\log^2 n$ consecutive points. Based on the points in each bucket, build a D-structure which is an instance of Lemma 12.1.3. Next, build a height-balanced binary search tree $\mathcal{T}$, where the buckets are placed at the leaves from left to right based on their ordering along the x-axis. Let v be a proper ancestor of a leaf node u and let $\Pi(u, v)$ be the path from u to v (excluding u and v). Let $S_l(u, v)$ be the set of points in the subtrees rooted at nodes that are left children of nodes on the path $\Pi(u, v)$ but not themselves on the path. Similarly, let $S_r(u, v)$ be the set of points in the subtrees rooted at nodes that are right children of nodes on the path $\Pi(u, v)$ but not themselves on the path. See Figure 12.2, which illustrates these sets for two leaves $u = u_l$ and $u = u_r$. For each pair (u, v), let $S'_l(u, v)$ (resp., $S'_r(u, v)$) be the set of points that each have the highest y-coordinate value among the points of the same color in $S_l(u, v)$ (resp., $S_r(u, v)$).

Finally, for each pair (u, v), construct a *sketch*, $S''_l(u, v)$, by selecting the $2^0, 2^1, 2^2, \ldots$-th highest y-coordinate point in $S'_l(u, v)$. A symmetric construction is performed to obtain $S''_r(u, v)$. The number of (u, v) pairs is bounded by $O((n/\log^2 n) \times (\log n)) = O(n/\log n)$ and hence, the space occupied by all the $S''_l(u, v)$ and $S''_r(u, v)$ sets is $O(n)$.

Query algorithm. To answer a query $q = [x_1, x_2] \times [y, \infty)$, we first determine the leaf nodes u_l and u_r of $\mathcal{T}$ containing x_1 and x_2, respectively. If $u_l = u_r$, then we query the D-structure corresponding to the leaf node and we are done. If $u_l \neq u_r$, then we find the node v which is the least common ancestor of u_l and u_r. The query is now broken into four sub-queries: First, report the approximate count in the leaves u_l and u_r by querying the D-structure of u_l with $[x_1, \infty) \times [y, \infty)$ and the D-structure of u_r with $(-\infty, x_2] \times [y, \infty)$. Next, scan the list $S''_r(u_l, v)$ (resp., $S''_l(u_r, v)$) to find a

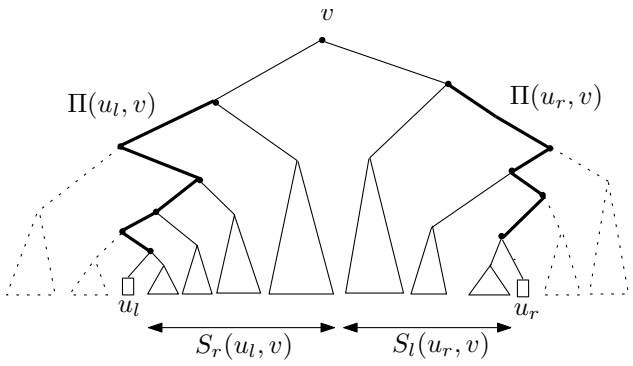


Figure 12.2: Answering a colored 3-sided range search in $\mathbb{R}^2$ query.

2-approximation of the number of colors of $S_r(u_l, v)$ (resp., $S_l(u_r, v)$) present in q.

The final answer is the sum of the counts returned by the four sub-queries. The time taken to find u_l, u_r and v is $O(\log n)$. Querying the leaf structures takes $O((\log(\log^2 n))^3) = O(\log n)$ time. The time taken for scanning the lists $S''_r(u_l, v)$ and $S''_l(u_r, v)$ is $O(\log n)$. Therefore, the overall query time is bounded by $O(\log n)$. Since each of the four sub-queries give a 2-approximation, overall we get an 8-approximation.

12.2 C-approximation for 4-sided range search

Now we will prove Theorem 12.0.1(B). Again we will use the framework of Theorem 11.0.1. It is straightforward to obtain a data structure with $\mathcal{S}_{capp} = O(n \log n)$, $\mathcal{Q}_{capp} = O(\log n)$ and $C = 16$. Simply build a binary range tree on the y-coordinates of S and at each node build an instance of Lemma 12.1.1 based on the points in its subtree. Given a 4-sided query rectangle q, it can be broken down into two 3-sided query rectangles. Shi and Jaja [110] presented a reporting structure with $\mathcal{S}_{rep} = O(n \log n)$ and $\mathcal{Q}_{rep} = O(\log n)$. Plugging in these values into Theorem 11.0.1 proves Theorem 12.0.1(B).

Remark 3. The technique of [36] can be adapted to answer a colored approximate counting query. For colored 3-sided range search in $\mathbb{R}^2$, plugging in $S(n) = O(n)$ and $Q(n) = O(\log n)$ [17] leads to a data structure of size $O(n \log^2 n)$ and query time $O(\log^2 n)$. For colored 4-sided range search in $\mathbb{R}^2$, plugging in $S(n) = O(n \log n)$ and $Q(n) = O(\log n)$ [58] leads to a data structure of size $O(n \log^3 n)$ and query time

$O(\log^2 n)$ (the structure of Chazelle [22] can be used to obtain slightly better space).

12.3 Open problem

The focus of our solutions for approximate counting has been on optimizing the space and the query time bounds in terms of n, and not on ε. (Currently, our space and query time bounds have a factor of ε^{-2}.) New ideas might be needed to improve the dependency on ε.

Chapter 13

Final Remarks

This thesis has presented new structures for several GIQ problems. There are several interesting open problems. To preserve context, these open problems have been mentioned at the conclusion of the relevant chapters. We wrap-up by reviewing the main techniques which were used to obtain the solutions presented in the thesis.

Random sampling. For approximate counting, random sampling on colors was used to reduce the problem to simpler companion problems of C-approximation and reporting query (with small output size). For top-k queries, random sampling on objects was used to reduce the problem to the companion problems of max-reporting and/or prioritized-reporting query. Interestingly, for top-k queries, although the query requests an exact answer, the intermediate steps of the query algorithm involve approximation via random sampling.

Shallow cuttings. For any geometric set, $\mathcal{A}$, whose lower-envelope has linear-complexity, we can efficiently construct a "ladder" of approximate levels via shallow cuttings. In Chapter 3 the set $\mathcal{A}$ was octants in $\mathbb{R}^3$, q was a point, and the aggregation function was reporting. To obtain an optimal query time, we used shallow cuttings to quickly retrieve a superset of $\mathcal{A} \cap q$ whose size was at most $O(|\mathcal{A} \cap q|)$. In the context of approximate counting, we used shallow cuttings to approximate the quantity $|\mathcal{A} \cap q|$. In fact, we constructed *nested* shallow cuttings which led to an optimal solution for 3-sided rectangle stabbing in 2-d.

Filtering-search type arguments. Suppose we want to solve a GIQ problem in $O(f(n)+k)$ time, where k is the number of objects to be reported. In this thesis, we made use of the following observations: (a) When $k \leq f(n)$, then we are allowed to answer the query in $O(f(n))$ time, and, hence, we can afford to report a superset of $O(f(n))$ objects before performing a pruning step. (b) When $k \geq f(n)$, then $O(f(n)+k) = O(k)$ and the amount of time the query algorithm can spend "searching" for the answer is $O(k)$, which can be significantly more than $f(n)$. These two observations provide a little freedom to the query algorithm, and helped us in designing data structures and general reductions which occupied less space (for e.g., Section 3.5, 3.6, 6.1, and 6.2).

Miscellaneous. We also used space partitioning techniques (for e.g., range tree, interval tree, segment tree, van Emde Boas tree, fusion tree, $\sqrt{n} \times \sqrt{n}$-grid tree), persistence, word-RAM tricks, and a reduction of a colored problem to an uncolored problem.

References

[1] Saladi Rahul. Improved bounds for orthogonal point enclosure query and point location in orthogonal subdivisions in $\mathbb{R}^3$. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 200–211, 2015.

[2] Timothy Chan, Yakov Nekrich, Saladi Rahul, and Konstantinos Tsakalidis. New results for rectangle stabbing and orthogonal point location in 3-d. Manuscript.

[3] Saladi Rahul and Ravi Janardan. Algorithms for range-skyline queries. In *Proceedings of ACM Symposium on Advances in Geographic Information Systems (GIS)*, pages 526–529, 2012.

[4] Saladi Rahul and Ravi Janardan. A general technique for top-k geometric intersection query problems. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 26(12):2859–2871, 2014.

[5] Saladi Rahul and Yufei Tao. On top-k range reporting in 2d space. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 265–275, 2015.

[6] Saladi Rahul and Yufei Tao. Efficient top-k indexing via general reductions. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, 2016.

[7] Saladi Rahul. Approximate range counting revisited. In *Proceedings of Symposium on Computational Geometry (SoCG)*, 2017.

[8] Saladi Rahul and Yufei Tao. Optimal top-k halfplane searching in 2-d. Manuscript.

[9] Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences (JCSS)*, 47(3):424–436, 1993.

[10] Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM (CACM)*, 31(9):1116–1127, 1988.

[11] Robert Endre Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *Journal of Computer and System Sciences (JCSS)*, 18(2):110–127, 1979.

[12] Pankaj K. Agarwal and Jeff Erickson. Geometric range searching and its relatives. *Advances in Discrete and Computational Geometry*, pages 1–56.

[13] Jon Louis Bentley. Decomposable searching problems. *Information Processing Letters (IPL)*, 8(5):244–251, 1979.

[14] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM (CACM)*, 18(9):509–517, 1975.

[15] Raphael A. Finkel and Jon Louis Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9, 1974.

[16] Rudolf Bayer and Edward M. McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 1:173–189, 1972.

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

[18] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 47–57, 1984.

[19] Neil Sarnak and Robert Endre Tarjan. Planar point location using persistent search trees. *Communications of the ACM (CACM)*, 29(7):669–679, 1986.

[20] James R. Driscoll, Neil Sarnak, Daniel Dominic Sleator, and Robert Endre Tarjan. Making data structures persistent. *Journal of Computer and System Sciences (JCSS)*, 38(1):86–124, 1989.

[21] Bernard Chazelle and Leonidas J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1(2):133–162, 1986.

[22] Bernard Chazelle. Filtering search: A new approach to query-answering. *SIAM Journal of Computing*, 15(3):703–724, 1986.

[23] V.N. Vapnik and A.Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280, 1971.

[24] David Haussler and Emo Welzl. Epsilon-nets and simplex range queries. *Discrete & Computational Geometry*, 2:127–151, 1987.

[25] Kenneth L. Clarkson and Peter W. Shor. Application of random sampling in computational geometry, ii. *Discrete & Computational Geometry*, 4:387–421, 1989.

[26] Peter van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10:99–127, 1977.

[27] Dan E. Willard. Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. *Information Processing Letters (IPL)*, 17(2):81–84, 1983.

[28] Timothy Chan. Orthogonal range searching in moderate dimensions: k-d trees and range trees strike back. In *Proceedings of Symposium on Computational Geometry (SoCG)*, 2017.

[29] Dieter Pfoser, Christian S. Jensen, and Yannis Theodoridis. Novel approaches to the indexing of moving object trajectories. In *Proceedings of Very Large Data Bases (VLDB)*, pages 395–406, 2000.

[30] Pankaj K. Agarwal, Lars Arge, and Jeff Erickson. Indexing moving points. *Journal of Computer and System Sciences (JCSS)*, 66(1):207–243, 2003.

[31] Pankaj K. Agarwal, Siu-Wing Cheng, Yufei Tao, and Ke Yi. Indexing uncertain data. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 137–146, 2009.

[32] Pankaj K. Agarwal, Nirman Kumar, Stavros Sintos, and Subhash Suri. Range-max queries on uncertain data. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 465–476, 2016.

[33] Jian Li and Haitao Wang. Range queries on uncertain data. *Theoretical Computer Science*, 609:32–48, 2016.

[34] Peyman Afshani, Lars Arge, and Kasper Green Larsen. Higher-dimensional orthogonal range reporting and rectangle stabbing in the pointer machine model. In *Proceedings of Symposium on Computational Geometry (SoCG)*, pages 323–332, 2012.

[35] Peyman Afshani. On dominance reporting in 3D. In *Proceedings of European Symposium on Algorithms (ESA)*, pages 41–51, 2008.

[36] Boris Aronov and Sariel Har-Peled. On approximating the depth and related problems. *SIAM Journal of Computing*, 38(3):899–921, 2008.

[37] Haim Kaplan, Edgar Ramos, and Micha Sharir. Range minima queries with respect to a random permutation, and approximate range counting. *Discrete & Computational Geometry*, 45(1):3–33, 2011.

[38] Pankaj Agarwal. Range searching. In *CRC Handbook of Discrete and Computational Geometry (J. Goodman, J. ORourke, and C. Toth eds.), CRC Press, New York*, 2016.

[39] Pankaj Agarwal. Simplex range searching and its variants: A review. In *Journey Through Discrete Mathematics. (M. Loebl, J. Neetril, and R. Thomas eds.), Springer, Heidelberg, to appear.*

[40] Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM Journal of Computing*, 9(3):615–627, 1980.

[41] David G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal of Computing*, 12(1):28–35, 1983.

[42] Herbert Edelsbrunner, Leonidas J. Guibas, and Jorge Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal of Computing*, 15(2):317–340, 1986.

[43] Jack Snoeyink. Point location. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 767–787. CRC Press, 2nd edition, 2004.

[44] Michael T. Goodrich, Jyh-Jong Tsay, Darren Erik Vengroff, and Jeffrey Scott Vitter. External-memory computational geometry. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 714–723, 1993.

[45] Lars Arge, Andrew Danner, and Sha-Mayn Teh. I/O-efficient point location using persistent B-trees. *ACM Journal of Experimental Algorithmics*, 8, 2003.

[46] Timothy M. Chan. Persistent predecessor search and orthogonal point location on the word RAM. *ACM Transactions on Algorithms*, 9(3):22, 2013.

[47] Mark de Berg, Marc J. van Kreveld, and Jack Snoeyink. Two- and three-dimensional point location in rectangular subdivisions. *Journal of Algorithms*, 18(2):256–277, 1995.

[48] John Iacono and Stefan Langerman. Dynamic point location in fat hyperrectangles with integer coordinates. In *Proceedings of the Canadian Conference on Computational Geometry (CCCG)*, 2000.

[49] Yakov Nekrich. I/O-efficient point location in a set of rectangles. In *Latin American Symposium on Theoretical Informatics (LATIN)*, pages 687–698, 2008.

[50] Herbert Edelsbrunner, G. Haring, and D. Hilbert. Rectangular point location in d dimensions with applications. *Comput. J.*, 29(1):76–82, 1986.

[51] Peyman Afshani, Lars Arge, and Kasper Dalgaard Larsen. Orthogonal range reporting: query lower bounds, optimal structures in 3-d, and higher-dimensional improvements. In *Proceedings of Symposium on Computational Geometry (SoCG)*, pages 240–246, 2010.

[52] Herbert Edelsbrunner and Mark H. Overmars. On the equivalence of some rectangle problems. *Information Processing Letters (IPL)*, 14(3):124–127, 1982.

[53] Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal of Computing*, 17(3):427–462, 1988.

[54] Sathish Govindarajan, Pankaj K. Agarwal, and Lars Arge. CRB-tree: An efficient indexing scheme for range-aggregate queries. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 143–157, 2003.

[55] Joseph JáJá, Christian Worm Mortensen, and Qingmin Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *Algorithms and Computation, 15th International Symposium, ISAAC 2004, Hong Kong, China, December 20-22, 2004, Proceedings*, pages 558–568, 2004.

[56] Michael T. Goodrich, Mark W. Orletsky, and Kumar Ramaiyer. Methods for achieving fast query times in point location data structures. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1997.

[57] Timothy M. Chan and Gelin Zhou. Multidimensional range selection. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 83–92, 2015.

[58] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd edition, 2008.

[59] Herbert Edelsbrunner. A new approach to rectangle intersections, part I. *International Journal of Computer Mathematics*, 13:209–219, 1983.

[60] Kurt Mehlhorn. *Data Structures and Algorithms 3*, volume 3 of *Monographs in Theoretical Computer Science. An EATCS Series*. Springer, 1984.

[61] A. Boroujerdi and Bernard M. E. Moret. Persistency in computational geometry. In *Proceedings of the Canadian Conference on Computational Geometry (CCCG)*, pages 241–246, 1995.

[62] Christos Makris and Konstantinos Tsakalidis. An improved algorithm for static 3d dominance reporting in the pointer machine. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 568–577, 2012.

[63] Christos Makris and Athanasios K. Tsakalidis. Algorithms for three-dimensional dominance searching in linear space. *Information Processing Letters (IPL)*, 66(6):277–283, 1998.

[64] Pankaj K. Agarwal, Lars Arge, Haim Kaplan, Eyal Molad, Robert Endre Tarjan, and Ke Yi. An optimal dynamic data structure for stabbing-semigroup queries. *SIAM Journal of Computing*, 41(1):104–127, 2012.

[65] Lars Arge and Jeffrey Scott Vitter. Optimal external memory interval management. *SIAM Journal of Computing*, 32(6):1488–1508, 2003.

[66] Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. New data structures for orthogonal range searching. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 198–207, 2000.

[67] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition.* The MIT Press, 2001.

[68] Pankaj K. Agarwal, Lars Arge, Jeff Erickson, Paolo Giulio Franciosa, and Jeffrey Scott Vitter. Efficient searching with linear constraints. *Journal of Computer and System Sciences (JCSS)*, 61(2):194–216, 2000.

[69] David A. White and Ramesh Jain. Similarity indexing with the SS-tree. In *Proceedings of International Conference on Data Engineering (ICDE)*, pages 516–523, 1996.

[70] Hans-Peter Kriegel, Marco Potke, and Thomas Seidl. Managing intervals efficiently in object-relational databases. In *Proceedings of Very Large Data Bases (VLDB)*, pages 407–418, 2000.

[71] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys*, 40(4), 2008.

[72] Iwona Bialynicka-Birula and Roberto Grossi. Rank-sensitive data structures. In *String Processing and Information Retrieval (SPIRE)*, pages 79–90, 2005.

[73] Peyman Afshani, Gerth Stolting Brodal, and Norbert Zeh. Ordered and unordered top-K range reporting in large data sets. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 390–400, 2011.

[74] Gerth Stolting Brodal. External memory three-sided range reporting and top-k queries with sublogarithmic updates. In *Proceedings of Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 47, pages 23:1–23:14. 2016.

[75] Gerth Stolting Brodal, Rolf Fagerberg, Mark Greve, and Alejandro Lopez-Ortiz. Online sorted range reporting. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 173–182, 2009.

[76] Cheng Sheng and Yufei Tao. Dynamic top-K range reporting in external memory. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, 2012.

[77] Yufei Tao. A dynamic I/O-efficient structure for one-dimensional top-k range reporting. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 256–265, 2014.

[78] Gonzalo Navarro and Yakov Nekrich. Top-k document retrieval in optimal time and linear space. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1066–1077, 2012.

[79] Biswajit Sanyal, Prosenjit Gupta, and Subhashis Majumder. Colored top-K range-aggregate queries. *Information Processing Letters (IPL)*, 113(19-21):777–784, 2013.

[80] Wing-Kai Hon, Rahul Shah, Sharma V. Thankachan, and Jeffrey Scott Vitter. Space-efficient frameworks for top-k string retrieval. *Journal of the ACM (JACM)*, 61(2):9:1–9:36, 2014.

[81] S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 657–666, 2002.

[82] Rahul Shah, Cheng Sheng, Sharma V. Thankachan, and Jeffrey Scott Vitter. Top-k document retrieval in external memory. In *Proceedings of European Symposium on Algorithms (ESA)*, pages 803–814, 2013.

[83] Moshe Lewenstein. Orthogonal range searching for text indexing. In *Space-Efficient Data Structures, Streams, and Algorithms*, pages 267–302, 2013.

[84] Travis Gagie, Simon J. Puglisi, and Andrew Turpin. Range quantile queries: Another virtue of wavelet trees. In *String Processing and Information Retrieval (SPIRE)*, pages 1–6, 2009.

[85] Gonzalo Navarro and Luís M. S. Russo. Space-efficient data-analysis queries on grids. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 323–332, 2011.

[86] Dan E. Willard and George S. Lueker. Adding range restriction capability to dynamic data structures. *Journal of the ACM (JACM)*, 32(3):597–617, 1982.

[87] Emo Welzl. Partition trees for triangle counting and other range searching problems. In *Proceedings of Symposium on Computational Geometry (SoCG)*, pages 23–33, 1988.

[88] Peyman Afshani and Timothy M. Chan. On approximate range counting and depth. *Discrete & Computational Geometry*, 42(1):3–21, 2009.

[89] Torben Hagerup and Christine Rub. A guided tour of Chernoff bounds. *Information Processing Letters (IPL)*, 33(6):305–308, 1990.

[90] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[91] Yufei Tao. Stabbing horizontal segments with rays. In *Proceedings of Symposium on Computational Geometry (SoCG)*, 2012.

[92] Lars Arge, Vasilis Samoladas, and Jeffrey Scott Vitter. On two-dimensional indexability and optimal range search indexing. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 346–357, 1999.

[93] Bernard Chazelle, Leonidas J. Guibas, and D. T. Lee. The power of geometric duality. *BIT Numerical Mathematics*, 25(1):76–90, 1985.

[94] Peyman Afshani and Timothy M. Chan. Optimal halfspace range reporting in three dimensions. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 180–186, 2009.

[95] Marek Karpinski and Yakov Nekrich. Top-k color queries for document retrieval. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 401–411, 2011.

[96] Yakov Nekrich and Jeffrey Scott Vitter. Optimal color range reporting in one dimension. In *Proceedings of European Symposium on Algorithms (ESA)*, pages 743–754, 2013.

[97] Manish Patil, Sharma V. Thankachan, Rahul Shah, Yakov Nekrich, and Jeffrey Scott Vitter. Categorical range maxima queries. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 266–277, 2014.

[98] Ying Kit Lai, Chung Keung Poon, and Benyun Shi. Approximate colored range and point enclosure queries. *Journal of Discrete Algorithms*, 6(3):420–432, 2008.

[99] Haim Kaplan, Micha Sharir, and Elad Verbin. Colored intersection searching via sparse rectangular matrix multiplication. In *Proceedings of Symposium on Computational Geometry (SoCG)*, pages 52–60, 2006.

[100] Prosenjit Gupta, Ravi Janardan, and Michiel H. M. Smid. Computational geometry: Generalized intersection searching. In *Handbook of Data Structures and Applications.* 2004.

[101] Prosenjit Gupta, Ravi Janardan, Saladi Rahul, and Michiel H. M. Smid. Computational geometry: Generalized (or colored) intersection searching. In *Handbook of Data Structures and Applications, to appear.*

[102] Panayiotis Bozanis, Nectarios Kitsios, Christos Makris, and Athanasios K. Tsakalidis. New upper bounds for generalized intersection searching problems. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, pages 464–474, 1995.

[103] Pankaj K. Agarwal, Sathish Govindarajan, and S. Muthukrishnan. Range searching in categorical data: Colored range searching on grid. In *Proceedings of European Symposium on Algorithms (ESA)*, pages 17–28, 2002.

[104] Prosenjit Gupta, Ravi Janardan, and Michiel H. M. Smid. Further results on generalized intersection searching problems: Counting, reporting, and dynamization. *Journal of Algorithms*, 19(2):282–317, 1995.

[105] Ravi Janardan and Mario A. Lopez. Generalized intersection searching problems. *International Journal of Computational Geometry and Applications*, 3(1):39–69, 1993.

[106] Haim Kaplan, Natan Rubin, Micha Sharir, and Elad Verbin. Counting colors in boxes. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 785–794, 2007.

[107] Kasper Green Larsen and Rasmus Pagh. I/O-efficient data structures for colored range and prefix reporting. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 583–592, 2012.

[108] Kasper Green Larsen and Freek van Walderveen. Near-optimal range reporting structures for categorical data. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 265–276, 2013.

[109] Yakov Nekrich. Efficient range searching for categorical and plain data. *ACM Transactions on Database Systems (TODS)*, 39(1):9, 2014.

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

[111] Marios Hadjieleftheriou and Divesh Srivastava. Approximate string processing. *Foundations and Trends in Databases*, 2(4):267–402, 2011.

[112] Boris Aronov and Micha Sharir. Approximate halfspace range counting. *SIAM Journal of Computing*, 39(7):2704–2725, 2010.

[113] Micha Sharir and Hayim Shaul. Semialgebraic range reporting and emptiness searching with applications. *SIAM Journal of Computing*, 40(4):1045–1074, 2011.

[114] Géza Tóth. Point sets with many k-sets. In *Proceedings of Symposium on Computational Geometry (SoCG)*, pages 37–42, 2000.

[115] Tamal K. Dey. Improved bounds for planar k-sets and related problems. *Discrete & Computational Geometry*, 19(3):373–382, 1998.

[116] Pankaj K. Agarwal, Alon Efrat, and Micha Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM J. Comput.*, 29(3):912–953, 1999.

[117] Peyman Afshani, Chris H. Hamilton, and Norbert Zeh. A general approach for cache-oblivious range reporting and approximate range counting. *Computational Geometry*, 43(8):700–712, 2010.

[118] Timothy M. Chan and Bryan T. Wilkinson. Adaptive and approximate orthogonal range counting. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 241–251, 2013.

[119] Jon Louis Bentley and James B. Saxe. Decomposable searching problems I: static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980.

[120] Peter van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters (IPL)*, 6(3):80–82, 1977.