A Tale of Twines, Quadrangles and Colorful Hierarchies

Rasmus Killmann Brogaard Petersen

PhD Dissertation

Department of Computer Science
Aarhus University
Denmark

A Tale of Twines, Quadrangles and Colorful Hierarchies

A Dissertation
Presented to the Faculty of Natural Sciences
of Aarhus University
in Partial Fulfillment of the Requirements
for the PhD Degree

by
Rasmus Killmann Brogaard Petersen
July 29, 2022

Abstract

This dissertation is based upon two papers and one manuscript. The first paper "A Lower Bound for Jumbled Indexing" examines the problem of jumbled indexing: Given an input string S on an alphabet $\Sigma = \{\sigma_1, \ldots \sigma_\delta\}$, store it in a data structure such that given frequencies $f_1, \ldots f_\delta$, report all substrings of S where the frequency of character σ_i is equal to f_i. In other words the problem is a matter of matching the occurrences of letters as opposed to their relative position in the substring. We present the first unconditional lower bound for the problem and furthermore we present the first lower bound which holds for the binary alphabet. We show that if a data structure has a query time of $O(n^{0.5-o(1)} + k)$, where k is the size of the output, then the data structure must use $\Omega(n^{2-o(1)})$ space. This shows that reporting the substrings matching a frequency is significantly harder than deciding whether such a match exists, even for a binary alphabet.

In the second paper "Rectangle Stabbing and Orthogonal Range Reporting Lower Bounds in Moderate Dimensions" we study classical problems within computational geometry, namely Rectangle Stabbing, Orthogonal Range Reporting and Dominance Reporting, although in an unusual setting, namely when the dimension is $c\log(n)$ (for some constant c). We show that if a data structure achieves $Q(n) = O(n^{1-\gamma})$ query time, for $\gamma \geq \frac{2}{2+\log c}$, then the space usage must be $\Omega\left(n^{1-\gamma} n^{\sqrt{c\gamma}/e - o(\sqrt{c\gamma})}\right)$. This overcomes shortcomings of previous proofs in which results could not be extended past $d = \Omega(\sqrt{\log n})$.

The manuscript "Hierarchical Categories in Colored Range Searching" extends the problem of colored range counting. In colored range counting, the colors are assumed to be independent and in this body of work we investigate different ways of assigning a hierarchy among the colors. We study two natural variants and when the underlying hierarchy has a tree structure, we show interesting connections to some well-established problems. When the hierarchy is a general DAG we show a conditional lower bound by reducing the orthogonal vectors problem to both variants. This shows that neither problem can be solved in less than quadratic time unless the well established Strong Exponential Time Hypothesis fails. Interestingly for one of the variants, we give a data structure which uses quadratic preprocessing time, $O(n^{3/2})$ space and $O(\log n)$ query time. This is one of the rare instances where there is a polynomial gap between the preprocessing time and the space usage of a data structure.

Resumé

Denne afhandling omhandler forskellige geometriske problemstillinger, som skal behandles med forskellige datastrukturer. Det primære fokus har været at bevise nedre grænser for hvor hurtigt et problem kan løses eller hvor meget plads en datastruktur skal bruge. Den første artikel omhandler et strengproblem. Givet en streng S over et alfabet $\Sigma = \{\sigma_1, \ldots \sigma_\delta\}$, vil vi gemme S i en datastruktur hvor givet en mængde af frekvenser $f_1, \ldots f_\delta$ bliver alle delstrenge af S, hvor σ_i forekommer f_i gange, rapporteret hurtigt. Dermed er vi interesserede i at finde strenge som har ens frekvenser og dermed, ulig andre strengproblemer, er vi ligeglade med den relative position af de enkelte karakterer i strengen. Vi viser at hvis man bruger $O(n^{0.5-o(1)} + k)$ tid på at rapportere delstrengene skal man bruge $\Omega(n^{2-o(1)})$ plads. Dette er den første nedre grænse hvor det også gælder for det binære alfabet. Dette beviser samtidig at det at rapportere delstrengene er sværere end at besvare hvorvidt der findes mindst en delstreng som matcher frekvenserne.

Den næste artikel omhandler tre klassiske geometriproblemer. I det første problem har man en mængde af rektangler i $\mathbb{R}^d$ og givet et enkelt punkt i $\mathbb{R}^d$ vil man gerne rapportere hvilke rektangler som indeholder punktet. I det andet problem har man en mængde af punkter i $\mathbb{R}^d$ og givet et rektangel vil man besvare hvilke punkter er indenfor rektanglet. Det sidste problem er givet en mængde af punkter i $\mathbb{R}^d$ og derefter et nyt punkt, hvilke punkter er mindre end det nye punkt i alle koordinater. Vi viser at hvis dimensionen er høj nok, er disse tre problemer ækvivalente. Vi viser at hvis man bruger $O(n^{1-\gamma})$ tid, hvor $\gamma \geq \frac{2}{2+\log c}$, skal man bruge $\Omega\left(n^{1-\gamma} n^{\sqrt{c\gamma}/e - o(\sqrt{c\gamma})}\right)$ plads.

Den tredje artikel udvider endnu et klassisk geometriproblem. Givet en mængde af punkter i $\mathbb{R}^d$, hvor alle punkter er blevet tildelt en farve og derefter givet et rektangel, vil vi tælle hvor mange forskellige farver der findes indenfor rektanglen. Dette problem har været studeret længe, men i tidligere studier antager man at farverne er uafhængige. I denne artikel beskriver vi to naturlige udvidelse af problemet, hvor vi danner struktur imellem de forskellige farver. Når strukturen er en træstruktur, viser vi for begge problemer interessante forbindelser til allerede etablerede problemer. Når strukturen er en orienteret acyklisk graf viser vi en reduktion fra ortogonale vektorer problemet og dermed får vi en kvadratisk betinget nedre grænse for begge varianter. For den ene variant beskriver vi en datastruktur, som bruger kvadratisk forbehandling, men kun $O(n^{3/2})$ plads og kan besvare forespørgsler i $O(\log n)$ tid. Dermed er der et polynomisk hul imellem forbehandlingstiden og pladsen en datastruktur skal bruge.

Acknowledgments

First and foremost I want to thank Katrine for her everlasting support and companionship throughout this project and in life. Thank you to my entire family for always believing in me and being there through the ups and downs. A special thank you goes to my Mother and Father for raising me to be the man I am today. Thanks to my two annoying little sisters, whom I love more than anything. Another huge thank you goes to Katrines family for providing the best second family anyone could hope for.

I also want to thank my fellow students along the way. From primary school, I want to thank Christian, Emil and Claus for an equal fight against mathematics and the school yard goals. From high school, I want to thank Mie Malene and Ditte for the times we had in and out of the class room. From university, a special thanks goes to Søren for battling our way through practical implementation problems to algebra and Fourier analysis. Another thanks goes to Thor, Bardur, Christoffer and Mark for being amazing study partners through tough subjects. Lastly I want to thank Kostas and Pingan for being amazing office mates throughout these studies.

Another huge thanks goes to all my teachers throughout my years in school. From primary school a special thanks goes to Iben for helping me through some tough times. From high school I want to thank Peter and Marianne for showing me a world outside of math. At university I want to thank Gerth for introducing me to algorithms and Kasper for nudging me into becoming a teaching assistant. A special thanks goes to Moshe for showing me Israel and Bar Ilan. The biggest thanks goes to my supervisor Peyman, thank you for showing me a way through research and thank you for being patient with me when things were rugged.

Rasmus Killmann Brogaard Petersen,
Aarhus, July 29, 2022.

Contents

Part I

Overview

Chapter 1

Introduction

This thesis is based upon two papers and one manuscript. The first paper "A Lower Bound for Jumbled Indexing" was presented at the Symposium on Discrete Algorithms (SoDA) 2020 and describes the first unconditional lower bound for the jumbled indexing problem. The second paper "Rectangle Stabbing and Orthogonal Range Reporting Lower Bounds in Moderate Dimensions" was presented at the Canadian Conference on Computational Geometry (CCCG) 2021 and gives a lower bound trade-off for rectangle stabbing, orthogonal range reporting and dominance reporting in moderate dimension. Subsequently, this paper was invited to the Journal of Computational Geometry and it is this journal version which is included in this thesis. The last manuscript titled "Hierarchical Categories in Colored Range Searching" is currently in submission for the International Symposium on Algorithms and Computation (ISAAC) 2022 and presents an extension on the classical problem of colored range counting.

The dissertation is structured in the following way: The thesis is split into two major parts, part I which is an overview and summary of the two papers and the manuscript and part II which contains the two papers and the manuscript as they were presented or submitted. In section 1.1, I present the general area of range searching, including orthogonal range searching and colored counting. In section 1.2, I discuss the different ways one can lower bound problems and data structures through conditional lower bounds and unconditional lower bounds. In chapter 2, I present the overall ideas and intuition which led to the paper "A Lower Bound for Jumbled Indexing", furthermore I present some simplified versions of the proofs from the paper. In chapter 3, the intuition behind the second paper "Rectangle Stabbing and Orthogonal Range Reporting Lower Bounds in Moderate Dimensions" and some auxiliary proofs which were omitted from the original paper is presented. In chapter 4, a summary of the ideas and motivation for our manuscript to ISAAC is given.

My Contribution

The first paper "A Lower Bound for Jumbled Indexing" was a project I continued which Ingo van Duijn, Jesper Sindahl Nielsen and Peyman Afshani had already started

working on. They had the initial idea for the string construction in the lower bound, but their construction used more than three character. I helped improve the construction down to two characters. Furthermore the connection to additive combinatorics was something Peyman and I worked out together, which is a vital piece in the lower bound when the number of elementary intervals is low. I wrote the ternary section while Peyman Afshani wrote the binary section.

The second paper "Rectangle Stabbing and Orthogonal Range Reporting Lower Bounds in Moderate Dimensions" was a new project Peyman Afshani and I worked on together. In this paper I was involved in the idea generation and wrote most of the paper on my own, only with minor modifications from Peyman Afshani.

The manuscript "Hierarchical Categories in Colored Range Searching" was again a project Peyman Afshani and I started. All the ideas were thought out together with Peyman Afshani and the construction of the conditional lower bound was given by Kasper Green Larsen. I wrote all technical details apart from the "Lower Bounds and Equivalence" section.

1.1 Range Searching

In Danish the field of computer science is called "Datalogi" which if directly translated means "Datalogy", i.e., the field of study that revolves around data. The question of what data is, is a broad question, but one of the typical representations of data is through records in a database. A record in a database is a tuple in which specific information is stored in each entry. Storing data is more useful when we can answer some specific questions (a.k.a "queries") about the stored data. In the following we will introduce the arguable simplest form of query on such a database, i.e., range searching.

Motivating Example

To motivate range searching, we study the following example. Consider a database of records that includes every American football player in Denmark. Assume that a record in this database includes numerical entries such as height, weight, age and years of playing, while there might also be some non-numerical entries such as the player's team. Now imagine that our database allows a user to issue the following queries. An orthogonal range is an axis-aligned rectangle of the form $\prod_{i=1}^{d}[a_i, b_i]$. Translated into our example database, we could construct the query of all players having a height of more than 2 meters, weighing between 100 kg and 160 kg, being less than 30 years old and having played less than 5 years, in the following way $q = [200, \infty) \times [100, 160] \times [0, 30] \times [0, 5]$. This is also known as orthogonal range reporting. Now given such a query, the database must output all tuples which satisfies the query q. The non numerical values can be used to filter your data records, e.g., by favourite team, i.e., "Aarhus Tigers" or they can be thought of as giving the data points a category. This would give us a colored point set in which we can answer queries like how many teams have players satisfying our query q (a colored counting query), which teams have such players (a colored reporting query) or whether there exists a team with such a player (an existence query). The question of how many players each team has which satisfy the query range is referred to as "type-2" colored range counting.

In general, range searching queries have the following structure: Given a set S of n points in $\mathbb{R}^d$, let R be a family of subsets of $\mathbb{R}^d$, referred to as ranges. The ranges can be any geometric object within $\mathbb{R}^d$ but typical ranges studied are rectangles, halfspaces, simplices or balls. The goal is to preprocess S into a data structure such that given a query range $\gamma \in R$, the points in $S \cap \gamma$ can be reported efficiently. The problem can be trivially solved using linear space and linear query time, by simply storing the points and given a query γ, checking for each point whether it lies within γ (since our ranges have constant complexity). This approach is insufficient when there are numerous queries being asked after one another. Therefore, we are interested in finding a way to preprocess S and build a data structure such that queries can be answered faster than in linear time.

The preprocessing time used to build the data structure is denoted by $P(n)$, the

size of the data structure is denoted by $S(n)$ and finally the worst case time complexity of a query is denoted by $Q(n)$. Within this context, the space consumption and query time is often more important than the preprocessing time (as long as the preprocessing time is somewhat reasonable). For reporting queries, the query time will depend on how many elements are output, since the data structure must output each element. We use k to denote the output size.

There are a few different models of computation in which we study algorithms. The random access model (RAM) counts the number of basic arithmetic operations (addition, multiplication, comparisons etc.) performed by the data structure, while the access of the data is free. The real random access model is an extension of the RAM model, where the algorithm can utilize real registers as well. The pointer machine model is a broad model with many different variations but it always disallows random access and hence memory can only be accessed through pointers.

Under many scenarios, the data structure needs to store all the points and thus $\Omega(n)$ is the trivial lower bound. Additionally, in the pointer machine model as well as real RAM, almost all natural problems have an $\Omega(\log n)$ query lower bound. In particular, binary search in either model requires $\Omega(\log n)$ time. In the pointer machine model, this is because $\Omega(\log n)$ pointer navigations are needed in the worst-case to access an element where as in the real RAM model $\Omega(\log n)$ comparisons are required to do the binary search (see various algebraic decision tree lower bounds for this [28]). Therefore the aim for a specific range searching question is whether there exists a linear space data structure with logarithmic query time.

Orthogonal Range Searching

In orthogonal range searching the query ranges are axis-parallel rectangles. One of the most fundamental data structure techniques to solve orthogonal range searching are range trees, which were introduced by Bentley [30]. If we look at the technique for the $2D$ problem, we have a set S of n points in $\mathbb{R}^2$. We build a binary search tree T in which every leaf contains a point and the point set is sorted according to x-coordinate, i.e., the i'th left most leaf contains the i'th smallest x-coordinate. For each internal node v, we store a canonical subset $S_v \subseteq S$ of points contained in the subtree rooted at v. Let a_v be the point with the smallest x-coordinate in S_v and let b_v be the point with the largest x-coordinate in S_v. At the interior node v, we store a_v, b_v and S_v sorted on their y-coordinate (this sorted array can be thought of as a range tree of dimension 1). Each level of the range tree stores a linear number of points, hence the size of the data structure becomes $O(n\log(n))$ and it can be constructed in $O(n\log(n))$ time. Given a query range $[x_1,x_2]\times[y_1,y_2]$, we can answer the reporting question by traversing T from the root. For a given node v we check if $[a_v,b_v]\subseteq[x_1,x_2]$ and if $[a_v,b_v]$ is fully contained in $[x_1,x_2]$ we report all points $p_i \in S_v$ where the y-coordinate of p_i lies in $[y_1,y_2]$, which can be done through a binary search of the sorted array. If $[a_v,b_v]$ has no overlap with $[x_1,x_2]$ output nothing. If $[a_v,b_v]$ intersects $[x_1,x_2]$ but is not fully contained, we recurse on the children of v. Finally if v is a leaf, we report the point contained in v. The query takes $O(\log^2(n)+k)$, but can be improved to

$O(\log(n)+k)$ by applying another well-known technique called fractional cascading. A d-dimensional range tree can be extended to a $d+1$ dimensional range tree by storing a d-dimensional range tree in each internal node as S_v, often referred to as a multilevel data structure. This gives a $\log(n)$ penalty in space and query time. Since the invention of range trees, several improvements have been made.

Colored Counting

Regular range searching encapsulates numerical data, but real world data sets may contain other data types as well. Another common data type in real world data sets is nominal data which is data that has no natural order. Colored or categorical range searching deals with such nominal data types. Each point is associated with a color/category which is stored in some data structure. For a query range, there are several interesting questions which can be answered, such as reporting the colors within the query range, counting the colors within the query range or to count the number of points of every color within the query range (known as "type 2"). In this section focus is on counting the different colors within the query range, as this is the question on which we build upon in chapter 4.

1D colored range counting can be solved with linear space and logarithmic query time. This is due to a reduction by Gupta et al. [55], which transforms the problem into an unweighted 2D orthogonal range counting problem which can be solved within the said bounds [40]. The reduction is quite simple and elegant:

Given a 1D colored range counting point set do the following. For a color c_i look at all the points of that color $p_1, p_2, p_3, \ldots p_m$, such that $p_i < p_{i+1}$, for $1 \leq i \leq m-1$. Then they are mapped into the 2D point set $(p_1, -\infty), (p_2, p_1), \ldots (p_m, p_{m-1})$. A query interval $[a,b]$ is mapped into the three sided range $[a,b] \times (-\infty, a]$. Now we claim that atleast one point p_i is contained in $[a,b]$ iff exactly one point is contained in the three sided range $[a,b] \times (-\infty, a]$. Let p_i be the smallest point contained in $[a,b]$, then $a \leq p_i \leq b$ and $p_{i-1} < a$. Then the point (p_i, p_{i-1}) is within $[a,b] \times (-\infty, a]$. Any point within the query interval, which is not the smallest, will get a y coordinate which is greater than a, hence it will be above the three sided range. If no p_i is within $[a,b]$ then every p_i is either smaller than a or bigger than b and hence the transformed points will all lie to the left or right of the three sided range. If we do this transformation for all colors c_i we get the desired transformation. An illustration of the transformation can be found in [62].

Later Larsen and Walderveen [69] showed that 2D unweighted range counting can be transformed into 1D colored range counting and thereby showing equivalence between the two problems. The reduction is yet another simple transformation:

The first key observation is that any orthogonal counting range in 2D is equivalent to four dominance counting points in 2D. Let the top right corner be p_{tr}, the top left corner p_{tl}, the bottom left corner p_{bl} and the bottom right corner p_{br}. Now if we query all four points in a dominance counting data structure, we see that the number of points inside the original query range becomes $\#p_{tr} - \#p_{tl} - \#p_{br} + \#p_{bl}$ where $\#$ denotes the output size (an illustration can be seen in fig. 1.1). So it is sufficient to

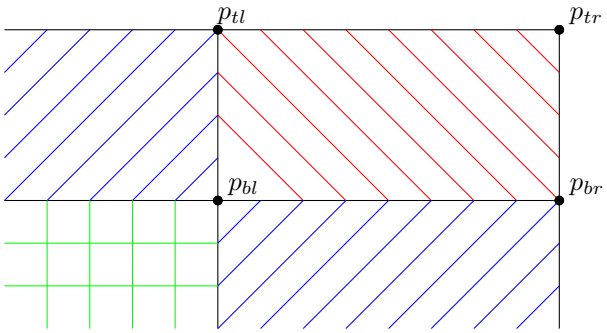

Figure 1.1: A visualisation of the four dominance points. The red lines are the points we want to count, the blue is counted by p_{tr} so we subtract it with p_{tl} and p_{br}. This means we subtract the green lines two times, hence we add p_{bl} as well.

reduce 2D dominance counting to 1D colored counting. Furthermore, we assume that the points in the 2D point set only have positive integer coordinates, this can be done by a reduction to rank space.

For every 2D point $p_i = (x_i, y_i)$ create the points $-x_i$ and y_i. We create two data structures D_1 and D_2. In D_1, we give the two new points the same unique color and in D_2 we give them two distinct colors. This gives us two data structures of $2n$ points, in D_1 the number of colors is n and in D_2 the number of colors is $2n$. Now for a query point $q = (x_q, y_q)$ we query D_1 and D_2 with the range $[-x_q, y_q]$ and we claim that the difference between the outputs of D_1 and D_2 is the number of dominated points in the original 2D problem. If a point $p_i = (x_i, y_i)$ is dominated by q, then the two points $-x_i$ and y_i will be within the range $[-x_q, y_q]$ and hence the difference of the two data structures will be $2 - 1 = 1$. If p_i is only dominated in one of the coordinates, then both data structures counts one and hence their difference will be zero. Now lastly, if p_i is not dominated in any of the two coordinates by the query point then both data structures will count zero. This completes the transformation.

These two reductions show that 1D colored counting is equivalent to 2D range counting. When the dimension increases, the problem becomes harder and the linear space and logarithmic query time becomes very likely impossible. Kaplan et al. [62] showed that answering m queries on a set of n points requires $\Omega(n^{\omega/2-o(1)})$ time, where ω is the boolean matrix multiplication exponent. Assuming the boolean matrix multiplication conjecture, this shows that $P(n) + mQ(n) \geq n^{3/2-o(1)}$ for any data structure which tries to solve the 2D colored range counting problem.

An extension of the colored range counting problem, in which the colors have some internal structure, will be discussed in chapter 4.

1.2 Lower Bounds

My first encounter with algorithms and the design of algorithms happened many years before I attended university. As a child, many summers were spent at my grandparents' house and one of our favourite activities were to play cards with my granddad. In this particular game of cards each person were dealt a hand of seven cards. In order to be ready for the game, it made sense to do some kind of ordering of the cards and therefore I came up with an ordering among the playing cards. The hearts were to the left, followed by the diamonds, followed by the spades and lastly the clubs (this is the ordering that made sense to an eight year old kid). Among the categories the ace was the smallest followed by 2 through 10 and lastly the jack was smaller than the queen which was smaller than the king. As a kid, little thought was given into how to sort these seven cards. Maybe it was due to the fact that I possessed little to no knowledge of algorithm design or maybe due to the fact that with seven cards pretty much any sorting algorithm would seem rather efficient. Anyway, the sorting algorithm I used was to first scour through the cards to find the smallest card and put it to the far left and repeat the process until every card was in its correct place.

The above sorting algorithm is not optimal. Seeing as we use $n + (n-1) + \cdots + 1 = \frac{n(n+1)}{2} = O(n^2)$ comparisons and hence the algorithm runs in quadratic time. Sorting can be solved using $O(n \log n)$ time by merge sort or in expectation by randomized quick sort. Now the natural question is whether this can be improved? It turns out that this is not possible due to the following lower bound.

A combinatorial lower bound can be given for the sorting problem. Since any sorting algorithm based on comparisons can be formulated as a decision tree we just need to determine the height of a decision tree in which each permutation appears as a reachable leaf. The number of permutations is $n!$ and since the number of leaves in a tree of height h is at most 2^h we get that

$$n! \leq 2^h \leftrightarrow \log(n!) \leq h.$$

Which gives a lower bound for sorting our cards of $\Omega(n \log(n))$. In nature, this is a combinatorial lower bound and these are the lower bounds which are the most predominant when looking at problems in the RAM. In order to achieve other lower bounds for algorithmic problems we have two paths to look at. The first approach is to change the model of computation into models such as the pointer machine model or the cell probe model. Another approach is from complexity theory in which we can do conditional lower bounds. Both approaches have pros and cons and these differences will be discussed later in this section.

Conditional Lower Bounds

The area of conditional lower bounds (fine grained complexity) is quite a new field and it has received significant attention during the last decade. The idea stems from classic complexity theory that every computer science student knows such as the infamous P vs NP question. When proving a problem p to be NP-hard, one can prove

it by doing a reduction from an already known NP-hard problem p'. Now if one can compute a reduction mapping from p' to p in polynomial time, we know that p is NP-hard as well. A key ingredient here is that the computation of the reduction is polynomial, such that the instance is not "solved" in the reduction itself. The idea with fine grained complexity is to utilize such reductions from some pre conjectured hard problems into problems that we want to argue are hard to solve. Now we need to place an even larger constraint upon the reduction, since a polynomial reduction might be too powerful and thereby one could solve the instance in the reduction.

There are a few famous base conjectures in fine grained complexity and one of the most famous conjectures actually stems from classic complexity theory. The strong exponential time hypothesis is a conjecture built around the well known k-SAT problem. In the k-SAT problem, the input is a boolean formula given in conjunctive normal form (CNF), in which each clause has exactly k literals, decide whether or not the formula is satisfiable. A CNF formula is a formula consisting of the conjunction of several clauses, where each clause consists of a number of literal disjunctions. This leads to the strong exponential time hypothesis.

Hypothesis 1.1 (SETH)**.** *For every $\varepsilon > 0$ there exists a k such that k-SAT requires at least $2^{(1-\varepsilon)n}$ time to be solved.*

k-SAT is known to be NP-hard for $k \geq 3$ and assuming $P \neq NP$ no sub exponential time algorithm exists for k-SAT and hence the SETH is in some way an even stronger hypothesis than $P \neq NP$.

One of the most used base conjectures for fined grained complexity is the orthogonal vectors (OV) hypothesis. It follows from SETH[90] and the problem of orthogonal vectors and the hypothesis is defined in the following way:

Problem 1.1. *Given two sets A and B of n binary vectors of polylog dimension, determine if there exists $a \in A$ and $b \in B$ such that a and b are orthogonal.*

Hypothesis 1.2 (OV)**.** *The Orthogonal Vectors problem requires $n^{2-o(1)}$ time to be solved.*

In order to use the OV-hypothesis, we need to construct a reduction in less than quadratic time from the orthogonal vectors problem to our desired problem. Say we have a problem p and we want to show that it takes at least quadratic time to solve p. If we can compute a reduction r which takes the orthogonal vectors problem and formulate it in terms of p, such that a solution to p results in two vectors being orthogonal in the OV problem, we know that p must use at least quadratic time to be solved otherwise the OV-hypothesis is false. The reason is that if p could be solved in less than quadratic time, any OV instance could be reduced to p and then solved. Therefore it is also key that the reduction itself takes sub quadratic time and usually the reduction time is linear. Fine grained complexity has been widely used to prove many interesting lower bounds in the last decade in which progress had been stalled for a while. An interesting example of such is that assuming SETH, it has been shown

that quadratic running time is essentially optimal for similarity measures such as Edit distance [24], Longest Common Subsequence [48] and Fréchet distance [32]. A list of results, all though not comprehensive, can be seen here [1–9, 19, 20, 25, 65, 66, 87]. All of the listed results have been published within the last decade making it quite a young field.

Restricting the Model of Computation

Another approach to proving lower bounds for algorithmic problems is to restrict the model of computation. The main challenge of changing the model of computation is that it might not capture the algorithmic setting that we are used to, i.e., RAM. Nevertheless, restricting the model of computation does not have to be a purely theoretical question since, as we will see, other models can also describe how many common data structures operate.

One model of computation which has proven efficient in proving lower bounds on reporting problems is the Pointer Machine Model [85]. In this model of computation the underlying data structure navigates the memory solely through pointers, as the name suggests. Since the memory is accessed through pointers, this model gives bounds for any tree based data structure. Consider some abstract reporting problem where the input is a set $\mathscr{U}$ and each query q reports a subset $q_{\mathscr{U}}$ of $\mathbb{U}$. The data structure is a directed graph $\mathscr{G}$, where every node has a constant out degree (sometimes referred to as having out degree two). Every node in the graph can store one element from the input set $\mathscr{U}$. Furthermore, there is a special root node $r(\mathscr{G})$. Any additional information can be stored and accessed for free by the data structure and furthermore the data structure is computationally unlimited. Given a query q the algorithm must start at the root node $r(\mathscr{G})$ and explore a connected subgraph $\mathscr{G}' \subseteq \mathscr{G}$, such that each element of the query $q_{\mathscr{U}}$ is contained in at least one node of the subgraph $\mathscr{G}'$. The worst case size of the subgraph explored is the running time of the algorithm and the total size of $\mathscr{G}$ is the space consumption of the data structure.

One immediate difference from the conditional lower bounds is that the bounds becomes a space-query trade-off and hence the bounds in the pointer machine model says something about the space consumption of the data structure. One reason that this model is useful for proving lower bounds is that we have multiple frameworks for proving such space-query trade-offs.

The first framework was given by Bernard Chazelle [42, 44]. Consider a reporting problem on a set of n elements, then for any data structure which achieves $S(n)$ space and $Q(n)$ query time, if there exists a set of queries $\mathscr{Q}$, such that each $q \in \mathscr{Q}$ outputs $\Omega(Q(n))$ elements and for any α queries they have a small intersection, i.e., for any α queries they have few output elements in common. If these two conditions are met we get a space-query trade-off. A similar framework were given by Afshani [10], in which the data structure operates on a geometric stabbing problem. In this framework we aim to tile the unit cube such that any point is covered atleast $Q(n)$ times and for any α different shapes used to tile the unit cube they have a small intersection volume. Again we gain a space-query trade-off. Interestingly both frameworks follow

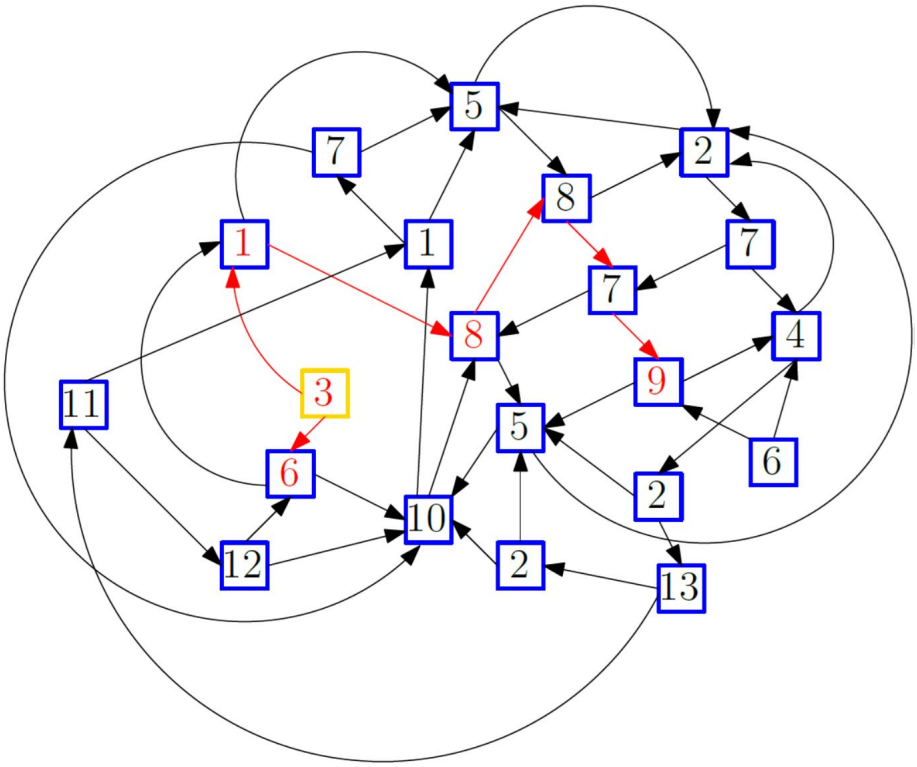

Figure 1.2: A data structure in the pointer machine model. The yellow node is the root, the red pointers connects the subgraph $\mathscr{G}'$ explored at query time and the red numbers are the query output.

the same recipe, they make sure that each query has a large output set and also make sure that if we take a number of queries, they will have a small overlap. Furthermore, we see that ensuring that each query has a large output and ensuring small overlap are two conditions which pull in two different directions. If each query outputs a large number of elements, one would expect that α queries have a lot of elements in common. Therefore, when using these frameworks the goal is to balance these two conditions.

The pointer machine model has proven especially effective in encapsulating the complexity of certain geometric problems. For orthogonal range reporting optimal lower bounds have been provided up until $3D$ [15], which gives a query time of $O(\log(n)+k)$ using optimal space $n(\log(n)/\log\log(n))^2$. The model, however, is not limited to geometric settings as a lower bound for dynamic fractional cascading was presented [11] proving a worst case query time of $\Omega(\log(n)\sqrt{\log\log(n)})$.

Another powerful model of computation is the cell probe model. Instead of following pointers, as in the pointer machine model, the data structure is limited to store everything in cells of size $\log(n)$, where n is the input size of the problem. Now, the cost of an operation is the number of cells it reads or writes (probes). Any

computation of which cell to probe is free. This is clearly a way too powerful model of computation for real world data structures and hence if we can prove lower bounds in this model, we know they hold in the more restrictive word-RAM.

The cell probe model has proven very effective in proving lower bounds for certain data structure problems which are in fact tight, see [51, 67, 75–77, 88]. A number of techniques exists for proving lower bounds on data structures. For static data structures, a technique by Larsen [68] can prove lower bounds of the form $Q(n) = \Omega(\log(m)/\log(\alpha))$ where $\alpha = S(n)/n$ and m is the number of distinct possible queries. Note that this is the largest possible lower bound one can hope to prove using this technique. In other words, if for a problem we have a large number of queries (i.e., m is large), it is not guaranteed that it is possible to prove a large lower bound. Hence, the method cannot show lower bounds higher than $Q(n) = \Omega(\log(m))$ with linear space. For data structures where updates are allowed, i.e., a dynamic data structure, techniques exists [67, 88] that gives lower bounds of $Q(n) = \Omega(\log(m)\log(n)/\log^2(uw))$ where u is the update time, w is the cell size and n is the input size and the number of updates performed on the dynamic data structure. Hence, the techniques gives us roughly a log factor more in the dynamic setting.

Recently Chakraborty, Kamma and Larsen [34] showed a tight cell probe lower bound for a variant of boolean matrix vector multiplication in which the data structure uses very little space. They proved that for succinct boolean matrix-vector multiplication any data structure storing r bits on the side, with $n < r < n^2$ must have query time $Q(n)$ satisfying $Q(n) \cdot r = \tilde{\Omega}(n^3)$ and for $r < n$ any data structure must have query time $\tilde{\Omega}(n^2)$. This gives some justification to the fact that the cell probe model still produces relevant lower bounds for central problems.

A Discussion of Lower Bound Models

As mentioned above, unconditional and conditional lower bounds have certain pros and cons. The unconditional lower bounds, where we change the model of computation, is the older of the two techniques dating back to the birth of computer science. The big advantage of using unconditional lower bounds is that they are unconditional, whereas the conditional lower bounds become obsolete if the base conjectures are refuted. On the other hand, conditional lower bounds can prove arbitrary high lower bounds as long as the base conjectures used are hard. This is in contrast to the unconditional lower bounds which have natural barriers. From the polynomial time hierarchy, we know that there exists problems for any power p in which the true complexity is $\Theta(n^p)$, the problem might just be to find base conjectures on which we can build the conditional lower bounds. Unconditional lower bounds also seem more suitable for proving space lower bounds although some work has been done in coming up with space conjectures for certain data structure problems [53]. Another interesting aspect of the pointer machine model is that it gives a concrete hard input instance as opposed to conditional lower bounds that are obtained through reductions. Nevertheless, fine grained complexity has settled many long standing problems.

Chapter 2

Jumbled Indexing

How does one compare if two strings S_1 and S_2 are similar? This is a fundamental question in both theoretical computer science and in real life applications. When thinking of a similarity measure between two strings, we can look at both the characters in the strings and their relative positioning. One of the most famous similarity measures between two strings is the Hamming distance [82] if we assume that S_1 and S_2 are of equal length. In the Hamming distance measure, the distance between S_1 and S_2 is the number of characters needed to be replaced in S_1 in order for the string to become equal to S_2, i.e., the number of positions in which the two strings have different characters (two example strings are shown in figure 2.1). This is widely used and accepted as the standard for string similarity, but as we will see there are applications in which the relative positioning of the characters in the two strings is less important and instead the frequency of each letter is the crucial similarity between the two strings.

As seen in figure 2.1 two strings can have a large hamming distance while still being fairly similar to the human eye. If we discard the notion of relative position and instead only focus on the frequencies at which characters appear in the strings we arrive at the notion of jumble matching. Given two strings S_1 and S_2 of length n over some alphabet $\Sigma = \{\sigma_1, \dots \sigma_\lambda\}$, we say that S_1 and S_2 jumble match if they have the same Parikh vector [74]. A Parikh vector ψ is a frequency vector $(f_1, f_2, \dots f_\lambda)$ where f_i describes the number of occurrences of $\sigma_i \in \Sigma$ in the string. Now S_1 and S_2 are equivalent if and only if $\psi(S_1)$ and $\psi(S_2)$ are equivalent, i.e., in other words S_2 can be obtained from S_1 simply by permuting S_1. This similarity measure is the foundation for the problem of jumbled indexing.

$$S_1 = 0\ 1\ 0\ 1\ 0\ 1$$

$$S_2 = 1\ 0\ 1\ 0\ 1\ 0$$

Figure 2.1: The two strings have hamming distance 6, since they differ at every index.

Jumbled indexing is a data structure problem and the input is a string S of length n over some alphabet $\Sigma = \{\sigma_1, \ldots \sigma_\lambda\}$, which we want to store in some data structure such that we can answer queries. A query consists of a Parikh vector $\psi = \{f_1 \ldots f_\lambda\}$. Note that the sum of the frequencies in ψ can be less than n. Generally speaking, a string indexing problem can have a number of variants, an existence variant where the goal is to simply answer whether there exists a substring in S which matches a query, a reporting variant where we want to output every matching position and furthermore a counting variant where the goal is to count how many matching positions there is. In this thesis the main focus is on the reporting variant and since every substring is uniquely defined from their starting position and their length (which is given by the sum of the frequencies in the query) it is sufficient to output all starting indices of matching substrings in the reporting variant.

2.1 Applications

The creation of Parikh vectors originates from the study of context-free languages by Rohit Parikh in the sixties [74]. Although they give rise to an interesting problem for theoretical computer scientists, the application of Parikh vectors extends into other fields as well. Various applications within molecular biology can be found in particular in the interpretation of mass spectrometry data. Mass spectrometry data is the output of an experiment which consists of molecular masses of sample molecules, where the molecular structure can in specific cases be determined with regards to a few candidates [31]. Examples of such structures could be the amino acids in a protein or the nucleotides of DNA. Since the goal is to find occurrences of these few candidates and since the relative positioning among the amino acids or the nucleotides does not matter [29], we can use the existential version of jumbled indexing. We simply create the Parikh vectors of all the candidates and run them as input queries on our database of mass spectrometry strings. An efficient algorithm for the reporting variant of the problem could even tell the positions in the string where these candidate amino acids or nucleotides reside.

Another interesting application of jumbled indexing is yet again from bioinformatics. Within the field of comparative genomics, the goal is to cluster different DNA sequences using different metrics in order to get an overview of the similarities and dissimilarities between species. Groups of genes in some "close" proximity needs to be identified in order to cluster the data [73]. One of such metrics is to look at the frequencies of the nucleotides and hence by using jumbled pattern matching one can cluster the data.

Few fields have seen as explosive an increase to the amount of data being collected as the field of gene sequencing. Although the work in our paper is of purely theoretical nature, this increase in data calls for efficient algorithms for jumbled indexing. Further references can be found here [33].

2.2 Current Bounds

The natural starting point is to find an upper bound for the problem, i.e., find an algorithm which solves it. Obviously, the problem is quite trivial for a unary alphabet, since for a string of length n and a frequency $f_1 = k$ we simply report all starting positions i in S such that $n-i \geq k$, hence we get $n-k+1$ reported substrings. However the complexity of the problem increases as we move to a binary alphabet.

Even after significant attention in the community, progress on this problem from the upper bound point of view, has been limited. There exist two different algorithms. The first algorithm stores no additional information, but uses a lot of time to process each query and the second algorithm stores a lot of additional information, but can answer queries efficiently.

The first approach is the sliding window algorithm. In this approach no additional information is stored and therefore the only information available at query time is the string itself. The main idea is to see that given a Parikh vector in the binary setting, one can think of the query as a balance between zeroes and ones. We define the balance of a binary Parikh Vector as $f_2 - f_1$, hence the difference between the number of ones and zeroes. Say we are given $\psi = \{3,5\}$, this gives a balance of $+2$ in the window of size $3+5=8$. Hence we need to find continuous substrings which has a balance of $+2$. This gives us our algorithm, starting from the beginning of the string, keep track of the balance in the window of size equal to $f_1 + f_2$ while sliding from left to right. Whenever we move one position to the right, we update the balance accordingly. If the character removed to the left of the window is equal to the new character added in the end of the window, the balance stays the same. If the added character is a one and the deleted character is a zero we add 2 to the balance and if the added character is a zero and the deleted character is a one we subtract 2 from the balance. An example is given in Figure fig. 2.2 where the first window have three zeroes and five ones, hence the correct balance of $+2$. When moving the window one position to the right we take away a one and add a zero, hence the balance becomes 0 and therefore we do not have a match with our query. Overall this approach gives us an algorithm which uses $O(1)$ space and takes $O(n)$ for each query.

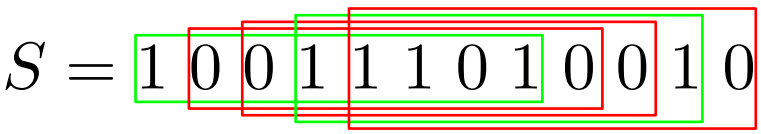

Figure 2.2: The sliding window algorithm with query $\psi = \{3,5\}$. Green corresponds to matching windows and red corresponds to non matching windows.

While the sliding window algorithm does not use any additional space it suffers from its linear query time. The other extreme is to store every possible $O(n^2)$ substring of S and given a query ψ we look up the answers in optimal time. Interestingly making progress on these simple solutions seems difficult.

Both approaches can easily be extended to larger alphabets. With the sliding window approach the query time becomes $O(n\log(\lambda))$, since we can update the window frequency through binary search over the alphabet size. The second solution where we store all of the substrings does not depend on the alphabet and hence can be used for larger alphabets as well. These are the upper bounds known for the reporting variant of the problem.

Bounds for the Existential Variant

For the binary existential variant an adaptation, of the sliding window algorithm can be used to get an optimal algorithm. The trick here is to do a preprocessing step, where for each window size keep track of the minimum balance and the maximum balance. If we store these values, then when given a query, we check if the query balance is between the minimum and maximum value for that query window size. If the query balance lies between the minimum and maximum, we say that the string contains a continuous substring which matches the query ψ otherwise it does not have a continuous substring which matches the query. The reason this works is that for any window size, when we use the sliding window approach, the balances encountered will be continuous values with offset 2. Hence, if a balance lies between the minimum and maximum encountered in the string, we know that the sliding window must have encountered that balance when moving from either the minimum balance to the maximum balance or the other way around. This yiels an algorithm with $O(n)$ space and $O(1)$ query time.

On the other hand Kociumaka et al. [64] presented an algorithm for the existential variant for a constant sized alphabet which solves the problem in $O(m^{\varepsilon})$ and uses $O(n^{2-\varepsilon})$ space, where m is the sum of the frequencies in the queried Parikh vector. Together with our result in chapter 5 this proves that the existential variant is easier than the reporting variant something which was not known before our work.

Interestingly the existential variant for the binary case has received a lot of attention, where the preprocessing time has been improved in a long line of work. The construction was first presented in [46] by Cicalese et al. where they presented the version discussed above, with linear space, constant query time, but quadratic preprocessing time. In a series of results [33, 70, 71] the preprocessing time was improved by polylog factors. The latest breakthrough remains the result of Chan and Lewenstein [37] where they showed a "bounded universe" version of the $(min,+)$-convolution can be solved in $\tilde{O}(n^{1.859})$ time and that reduces the preprocessing time of the binary existential jumbled indexing problem by polynomial factors as opposed to previous polylogarithmic improvements. The improvement even extends to constant sized alphabets, although for alphabets of size three and above the query time is no longer constant but requires $O(n^{1-\alpha})$ for some constant α.

The only lower bound present before our work was a conditional lower bound by Amir et al. [23] which proved Chan and Lewensteins construction is essentially optimal. Amir et al. showed that if the size of the alphabet is a constant $\lambda \geq 3$, there is a constant α_λ dependent on λ such that it is not possible to get $O(n^{2-\alpha_\lambda})$

preprocessing time and $O(n^{1-\alpha_\lambda})$ query time. As opposed to our result, their lower bound does not apply for the binary case, does not say anything about the space and furthermore is conditional. As seen above, space might be a crucial component since the binary alphabet has a solution with linear space while the preprocessing time is large.

Our Space-Time Trade-off

Our work led to the space-time trade-off lower bound presented in chapter 5. We presented a trade-off between the query time and space usage for the reporting variant of jumbled indexing in the pointer machine model. If a data structure can report all of the k matches to a jumbled indexing query in $O(n^{0.5} - o(1) + k)$ query time, then it must use $\Omega(n^{2-o(1)})$ space. This essentially states that if a data structure uses less than quadratic space it will use atleast squareroot query time or on the other hand if a data structure handles queries faster than squareroot it must use quadratic space.

Our result is unconditional in the pointer machine model, it gives a bound on the space of the data structure and lastly it applies to the binary alphabet, all of which prior results were unable to encapsulate. Since our result applies to the binary alphabet it follows that reporting all matches to a Parikh vector is significantly harder than checking if a single match exists. At the time this surprised us, since we believed that in order for jumbled indexing to be difficult the alphabet should have size at least three.

The disadvantage of our result is that it demands that the data structure operates in the restrictive pointer machine model. As opposed to the usual random access model (RAM), anytime the data structure needs to access an element it needs to go through a series of pointers as opposed to the constant time access of elements in the RAM. On the other hand the pointer machine model assumes nothing about the computational power of the data structure, hence the bound works for any data structure using pointers, regardless of the underlying computational cost, and it can also capture tree-based solution to jumbled indexing.

Conditional lower bounds are proven through a reduction of a conjectured hard problem. Therefore no hard input instance for the algorithm is provided. Unconditional lower bounds on the other hand provide a hard input instance, which helps deepen the understanding of the problem from the upper bound point of view as well. Furthermore, as briefly mentioned above, conditional lower bounds give a bound on the preprocessing time and query time, while the unconditional lower bound gives a trade-off between space and query time. Therefore both conditional and unconditional lower bounds can be used to describe the complexity of theoretical questions.

Next Steps

Jumbled indexing, both in the existential and reporting variant, have seen little to no improvement since our results. In the classic models of computation the boundary of our knowledge is still our results and prior results. For the existential variant,

the bounds are almost tight, with some room for improvement in determining the exact constants for α in the preprocessing time and query time. Another interesting area of research could be to find a trade-off between the query time and space usage for the existential variant, since as described above, there exists a solution with large preprocessing time but only linear space, which solves the existential variant in constant time for the binary alphabet. Hence, such a bound would help prove optimality for the binary case.

For the reporting variant the immediate approaches of either the sliding window algorithm and the store everything algorithm, remains the state of the art from the upper bound point of view. The best known lower bound is still our square root to quadratic trade-off, but this still leaves some room for improvement. Therefore a space-time trade-off of linear to quadratic would settle the problem.

Even though progress has slowed down on jumbled indexing in the traditional models of computation, interesting work is still being conducted in newer models of computation. An interesting approach is to look at the problem in the quantum setting. With the potential rise of quantum computers, theoretical computer scientist have started working in models which would accommodate this different way of doing computations. A recent result [61] looked at the reporting variant in the quantum model of computation. They present an algorithm running in $O(\sqrt{n})$ time with constant preprocessing time. Note that this does not break the existing bounds, since the model of computation is completely different from the random access model or the pointer machine model in which the lower bounds are built.

2.3 Lower Bound Construction

The goal of this section is to convey the main ideas and thoughts behind our lower bound construction. For further technical details, we refer to the paper in chapter 5.

The main construction of our unconditional lower bound consists of creating a hard input instance. This means creating an input instance which any algorithm would struggle to solve efficiently. In this section the intuition behind the lower bound construction for a ternary alphabet $\Sigma = \{0, 1, \#\}$ will be conveyed, as it presents the main ideas. The extension to the binary version will be discussed shortly at the end of this section.

The Input String

We need to build an input string S of size $\Theta(n)$ for the reporting version of the jumbled indexing problem. The construction consists of three parts, a left part, a middle part and a right part. The left part consists of balanced binary blocks separated by the third character. For some parameter s, the left part of the string have blocks consisting of zeroes and ones repeated s times. This gives us a number of blocks looking like $\#(01)^s$ and we repeat it n/s times. This gives us n/s binary balanced blocks and each block starts with # and is followed by s two bit chunks of 01. Now since these blocks are completely balanced we want to create a random pattern in the right part of the

string and force the query patterns to match only substrings starting in the left part of the string and ending in the right part of the string.

The middle part of the string starts after the n/s binary balanced blocks and consists of two #. This ensures that we can force the queries to span over the middle part of the string.

The right part of the string is going to consist of blocks of random bits. We create n/s blocks again, each of length $2s+1$, e.g., $x_1x_2\ldots x_{2s+1}$, where $x_{2s+1} = \#$. For every odd index i, we pick x_i to be uniformly random from $\{0,1\}$. For every even index we duplicate the previous value, i.e., $x_i = x_{i-1}$. Another way of describing the process of picking these random blocks is to pick a binary string of size s uniformly at random $d_1,\ldots d_s$ and then duplicate each of the random bits resulting in the string $d_1d_1,\ldots d_sd_s$ and then append # at the end of the block (the construction can be seen in figure 2.3). This gives us a string of size $\Theta(n)$ and the goal is to create a number of hard queries for this particular input string.

$$\#010101\ldots01$$
$$\#\#$$
$$001111\ldots00\#$$

Figure 2.3: The top string is a balanced left block, the middle is the separating part and the bottom is an example of a random right block. Note that we have n/s left and right blocks.

The Difficult Questions

Now in order to define our hard queries we need to define the notion of a valid substring. A substring $x_i,\ldots x_j$ is valid if $x_i = \#$, $x_j = \#$ and there exists indices k and $k+1$ such that $x_k = \#$ and $x_{k+1} = \#$ where $i < k$ and $k+1 < j$. In plain English this means that every valid substring must start with a left block, span the middle and then end with a right block. Hence, it follows that the number of valid substrings in our input string becomes $(\frac{n}{s})^2$, since we have n/s left blocks in which the substring can start and we have n/s right blocks in which the valid substring can end.

We prove in the paper that the probability that a substring has balance larger than roughly $\sqrt{n}$ is $2n^{-c}$. For $c = 4$ we get a probability of $2\frac{1}{n^4}$ and since we have $(\frac{n}{s})^2$ valid substrings, using the union bound we have that with high probability in every valid substring the difference between ones and zeroes is less than $\sqrt{n}$. Since this happens with high probability we assume that the valid substrings chosen abide to this balance constraint.

Now the goal is to create a query set in which each query matches a lot of valid substrings. Each query is defined by a 3-dimensional Parikh vector $\psi_i =$

$(\psi_i(0)\psi_i(1), \psi_i(\#))$. Since the query needs to output all of the matching indices we want the output from each query to be large. We want a valid query to start with #, include the middle ## and end with a #. This attribute can be assured with the following definition:

Definition 2.1. *A Parikh vector* $\psi = (\psi(0), \psi(1), \psi(\#))$ *is valid if*

(i) $\psi(0) + \psi(1)$ *is a multiple of* $2s$.

(ii) $\psi(\#) = \frac{\psi(0)+\psi(1)}{2s} + 2$.

(iii) $|\psi(0) - \psi(1)| \leq \sqrt{n}/\delta$.

The first condition ensures that we match a whole number of left/right blocks, the second condition ensures that we start and end with # and furthermore spans the two middle characters. Lastly, the bound on the balance makes sure that valid queries only match valid substrings.

To utilize the framework, we need to ensure that every query matches a lot of substrings and that the overlap between the outputs of the different queries is not to large. We will ensure the first condition in the following way.

To ensure that every query matches a lot of valid substrings we define the notion of a good and a bad Parikh vector. A Parikh vector is bad if it matches less than $\frac{\sqrt{n}\delta}{4s}$ valid substrings in our input string, otherwise it is good. The number of valid substrings is $(\frac{n}{s})^2$ and each valid substring matches a Parikh vector with high probability. The total number of Parikh vectors is $\frac{n}{s}\frac{\sqrt{n}}{\delta}$ and a bad Parikh vector match at most $\frac{\sqrt{n}\delta}{4s}$ and hence we get that all the bad Parikh vectors match at most $\frac{\sqrt{n}\delta}{4s} \cdot \frac{n}{s}\frac{\sqrt{n}}{\delta} = \frac{1}{4}(\frac{n}{s})^2$ valid substrings. Hence, at most one fourths of the valid substrings is matched by a bad Parikh vector. This also means that three fourth of the valid substrings matches a good Parikh vector. We therefore chose the set of good Parikh vectors as our query set. This ensures that they all match roughly $\sqrt{n}$ valid substrings.

The next step is to ensure that the overlap between the different queries is not large. The intuition here is that even if all the queries output $\sqrt{n}$ positions, if they all output the same positions, the data structure could solve the problem easily by simply storing the same output regardless of the query. Therefore, we want to ensure that if we take a number of queries (α queries) that their overlap is not too big. This will ensure that any data structure working on this instance will have to store several different things since the queries are spread out and in that way we either force a large space consumption or the data structure has to use a lot of time to retrieve the data on query time. The framework behind this intuition is the framework of Chazelle[42, 44]. So if every α queries have little overlap with each other, together with the fact that each query outputs roughly $\sqrt{n}$ positions, we get the trade-off we set out to get.

Ensuring Small Overlap between Queries

The proof consists of looking at two cases. The case where we have few elementary intervals and the case where we have many elementary intervals, so what is an elementary interval? Given our query set Q take out any α queries from Q, i.e., $q_1, ..., q_\alpha$ and look at all the positions $B = \{p_1, ..., p_\beta\}$ in our input string in which these α queries match. Look at the lengths of the queries $A = \{|q_1|, ..., |q_\alpha|\}$, then we define $D = A + B$ and define it as being in increasing order. An elementary interval is defined as the interval represented by two consecutive integers in D, i.e., $[d_i, d_{i+1} - 1]$. We define the number of elementary intervals as M.

The goal now is to show that given α queries from our query set, it is very unlikely that they have β output points in common. When the number of elementary intervals is large we obtain the bound through a union bound argument. However, when the number of elementary intervals is small, our union bound breaks down and hence we have to turn to additive combinatorics. By showing that if we have few elementary intervals we must have an arithmetic progression and afterwards proving that such an arithmetic progression is very unlikely, we obtain a contradiction and hence the α queries cannot match at the same β positions.

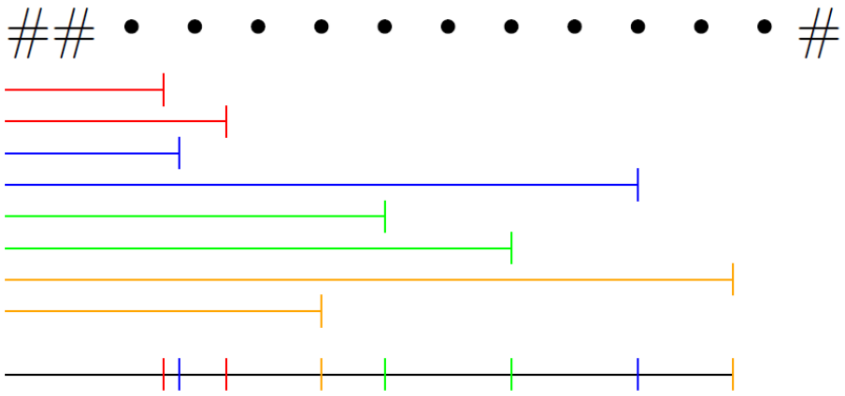

Figure 2.4: An illustration of the elementary intervals in the right part of the string. The different colors represents the different queries and at the bottom the elementary intervals formed from D is shown.

In the case of many elementary intervals we note that each elementary interval has size atleast $2s$, since that is the size of a right block. In order for these elementary intervals to align, each elementary interval must have a fixed value. The probability for each elementary interval to have this fixed value is $O(1/\sqrt{s})$ and since they are all disjoint the probability for a random string to have all of the fixed balances in all the elementary intervals is no more than $O((1/\sqrt{s})^M)$. If we apply a union bound we have β values for the matching positions, α different values for the size of the queries and finally we have α possible values for the balances $b_1, \ldots, b_\alpha$. b_i is defined as the

balance in the interval $p_1 + |q_1|$ to $p_1 + |q_i|$ for $i \in \{2, \ldots, \alpha\}$ (which is sufficient to calculate the balances of all the elementary intervals, see lemma 5.4 for a proof). This gives us:

$$O\left(n^{2\alpha+\beta} \cdot \left(\frac{1}{\sqrt{s}}\right)^M\right) = \Theta\left(n^{-\beta}\right),$$

when we choose s and M accordingly. This settles the case for many elementary intervals among our α queries.

For the case of few elementary intervals the above argument breaks down, as M is not big enough to kill the $n^{2\alpha+\beta}$ term. Hence, we need to turn our attention elsewhere. If there are few elementary intervals we can say that there must exist a k arithmetic progression formed by the elementary intervals. This means that the size of the k consecutive elementary intervals must be some fixed value d. The probability of these k elementary intervals having these fixed balances is $O((1/\sqrt{s})^k)$. The total number of arithmetic progressions that can be formed is $O(n^2)$, since we have $O(n)$ choices for the starting point and $O(n)$ choices for the common difference in the arithmetic progression. Therefore we have that the expected number of arithmetic progressions becomes:

$$\mathbb{E}[\#\text{k-term arithmetic progressions}] \leq (\frac{1}{\sqrt{s}})^k n^2 < n^{-2}.$$

So on one hand we know that an arithmetic progression have to exist, while the expected number of arithmetic progressions is less than n^{-2}. Hence, with high probability there is no arithmetic progression, which is a contradiction. This settles the case with few elementary intervals, since the α queries wont match at β common positions.

This concludes the lower bound construction. Every query in our query set has a large output and for any α queries they have little overlap in which positions they need to output. This is exactly the conditions needed for Chazelles framework and we get our lower bound by plugging in the values for α and β. The query output was roughly $\Omega(\sqrt{n})$ and we get a bound for the space from Chazelle:

$$S(n) = \Omega\left(\frac{\sum_{q \in \mathcal{Q}} |q|}{\alpha 2^{O(\beta)}}\right) = \Omega\left(n^{2-o(1)}\right).$$

2.4 The Binary Construction

The obvious challenge, when moving from three characters to two characters, is that we cannot split our blocks like we did above. If we just remove the third character # and consider a window (crossing the middle) which we slide with two characters at a time to the right. For each slide to the right, the window would lose a 01 on the left while gaining a 00 or a 11 on the right, both with equal probability. Essentially

this mimics a random walk: the balance goes up by one or goes down by one with equal probability. Hence, if we match at a position i, we will match again when the random walk from i returns to the same balance. This is also known as the structure of the zeroes of a simple random walk, meaning $\sum_{i=1}^{j} X_i = 0$ where X_i is $+1$ or -1 with equal probability. The "clustered" zeroes of a random walk means that if a string matches at position i, it is likely to match positions close to i as well. Consequently, this can break the second requirement of our lower bound framework because if two queries match the string at position i, it implies that they will also have a lot more common matching positions close to i. In the ternary alphabet case, we handled this problem by making sure that the matches are at least $2s$ apart and this meant that if two queries matched at a position i, at position $i+2s$ they were very unlikely to match again since we have moved $2s$ steps in our random walk. Since we do not have the extra character here, we need to come up with a different approach. Instead we introduce the notion of a random walk in windy conditions. The left part of the string is going to consist of 0^s1^s chunks and now when you slide the window to the right the balance can stay the same or increase/decrease depending on where the "wind" is coming from. If the left part of the window is in a zero block the wind pushes the sum up, such that the sum can only increase or stay the same. On the other hand if we are in a one block, the wind pushes the sum down and hence the sum can only decrease or stay the same. The "windy random walk" essentially ensures that if a query matches at position i, it can get very few lucky matches close to it, but then the next match will have to be $2s$ steps away, taking us back to the case of the ternary alphabet.

Furthermore, we need the queries to span the middle and since we do not have the special character we replace ## by 1^{10n}. When we define the notion of a valid substring and query we just add that the Parikh vector needs these extra $10n$ ones. Lastly the right part of the string becomes a random binary string of length $2n$. The construction can be seen in figure 2.5.

Even though some technical lemmas have to be proven in order for the binary case to be complete the main ideas and intuition still stem from the ternary case. For a thorough walk-through of our construction we refer to the paper.

$$0^s1^s0^s1^s\ldots0^s1^s$$
$$1^{10n}$$
$$01011\ldots101011$$

Figure 2.5: Our binary input string. The left string consists of n/s chunks of 0^s1^s, the middle part consists of $10n$ ones and lastly the right part consists of a random binary string of length $2n$.

Chapter 3

Rectangle Stabbing and Orthogonal Range Reporting

Computational geometry revolves around seeking solutions to geometric problems. Rectangle stabbing, orthogonal range reporting and dominance reporting are among the most fundamental problems in computational geometry. They are the subject of this section.

3.1 Rectangle Stabbing, Orthogonal Range- and Dominance Reporting

Rectangle stabbing is defined in the following way: given a set of input axis-aligned rectangles store them in a data structure such that given a query point, the data structure can report which rectangles contain the query point. The name of rectangle stabbing originates from thinking of the ranges as sticky notes and the query point as a pin being stabbed into them. In $1D$ the problem can be solved in a number of different ways including the use of segment trees or interval trees. This yields an optimal linear space complexity and a $O(\log(n)+k)$ query time (where k is the output size). In $2D$ Chazelle [39] matched the bounds of the one dimensional problem achieving an optimal data structure using linear space and $O(\log(n)+k)$. By applying range trees for higher dimensions, the $2D$ solution can be extended by paying an extra log factor in both space and query time for each dimension added, i.e., obtaining a query time of $O(\log^{d-1}(n)+k)$ and a space consumption of $O(n\log^{d-2}(n))$ for $d \geq 3$.

Orthogonal range reporting is in some sense the dual of the rectangle stabbing problem: Given a set of input points store them in a data structure such that given an axis-orthogonal query range the data structure can report which points are contained in the query range. So the role of points and ranges are swapped when compared to the rectangle stabbing problem. Orthogonal range reporting is a fundamental problem not only in computational geometry but across computer science in general. It is a fundamental question in database queries, each point is a database record and the

query is a select query upon the database. A more general overview of the problem is given in section 1.1.

Dominance reporting is defined as given a set of input points, store them in a data structure such that given a query point $q = (q_1, q_2, \ldots q_d)$, we can output all input points $x_i = (x_{i,1}, x_{i,2}, \ldots x_{i,d})$ where $x_{i,j} < q_j$ for $1 \leq j \leq d$. The name dominance reporting stems from the fact that we report all points which are dominated by the query point, i.e., the query point is greater in every dimension. In $1D$ the problem is essentially a binary search followed by outputting every element below the target, i.e., a $O(\log(n)+k)$ query time and linear space solution. In $2D$ Chazelle and Edelsbrunner[43] presented a linear space data structure achieving $O(\log^2(n)+k)$ query time, while the best known solution was presented by Alstrup et al.[22], achieving a query time of $O(k+1)$ while maintaining linear space (which of course also holds for 1D). In $3D$ Chan[35] presented a solution obtaining linear space and $O(\log\log(n)+k)$ query time. Recently Nekrich [72] presented a linear space solution for $4D$, which gives a query time of $O(\log^{1+\varepsilon}(n)+k\log^{\varepsilon}(n))$. Furthermore Nekrich presented an extension achieving dominance queries in d dimension using $O(n\log^{d-4+\varepsilon}(n))$ space and achieving $O(\log\log(n)(\log(n)/\log\log(n))^{d-3}+k)$.

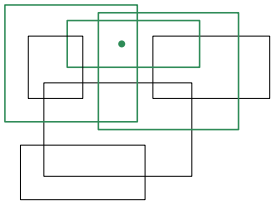

Figure 3.1: Rectangle stabbing

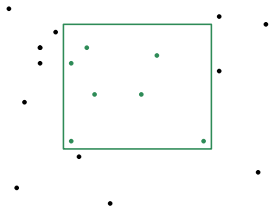

Figure 3.2: Orthogonal range reporting

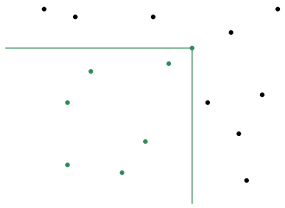

Figure 3.3: Dominance reporting

When in constant dimensions rectangle stabbing, orthogonal range reporting and dominance reporting have different complexities. While as we will see below, the three problems becomes equivalent when the dimension is non-constant, i.e., the problems can be reduced to one another with only a constant blow up in dimension. All of the extensions into general d dimension discussed above suffers from the so called "*curse of dimensionality*" and hence these approaches are unsuitable for the case where the dimension is non-constant, e.g., when the dimension is $c\log(n)$ for some constant c. The best solution for non constant dimension was presented by Chan [36] where they devised a data structure for $c\log(n)$ dimension which achieves $Q(n) = O(n^{1-1/c\log(c)})$ expected query time and $S(n) = n^{1+\delta}$ for any $\delta > 0$.

Equivalence in High Dimension

When the dimension is non constant, we can reduce the three problems to one another. The first step is to show that rectangle stabbing and orthogonal range reporting in d

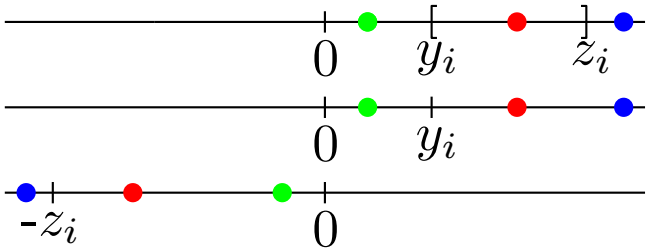

Figure 3.4: The different cases of lemma 3.1.

dimension can be reduced to dominance reporting in $2d$ dimension. The key ingredient is to see that the relation between a point being contained in an orthogonal range can be reduced to a dominance problem.

Lemma 3.1. *Given an orthogonal range* $R = [y_1,z_1] \times [y_2,z_2] \times \cdots \times [y_d,z_d]$ *and a point* $p = (x_1,x_2,\ldots,x_d)$ *the task of determining whether* $p \in R$ *is equivalent to determining if the* $2D$ *point* $q = (x_1,-x_1,x_2,-x_2,\ldots x_d,-x_d)$ *dominates the* $2D$ *point* $w = (y_1,-z_1,y_2,-z_2\ldots y_d,-z_d)$.

Proof. Lets focus on one interval $[y_i,z_i]$ and one coordinate x_i. Clearly if the relation holds for any interval, it holds for the entire orthogonal range. Now we are left with three cases, the case where $x_i < y_i$, the case where $x_i \in [y_i,z_i]$ and the case where $z_i < x_i$. When $x_i < y_i$ we get that the first coordinate in the transformation resolves in q not dominating w since $x_i < y_i$. When $x_i \in [y_i,z_i]$, we get that $x_i > y_i$ and that $x_i < z_i$ which in turn means that $-x_i > -z_i$ and hence q dominates w in both coordinates. When $x_i > z_i$ we get that $-x_i < -z_i$ and hence q does not dominate z (a visualisation can be seen in fig. 3.4). In either case, we get the correct relation and this concludes our proof. □

By applying lemma 3.1, we get a reduction from rectangle stabbing and orthogonal range reporting in d dimension into dominance reporting in $2d$ dimension. Now we only need to prove that dominance reporting can be reduced to rectangle stabbing and orthogonal range reporting. Clearly this is possible, dominance reporting can be thought of as a special case of orthogonal range reporting in which the query point $(x_1,x_2,\ldots x_d)$ becomes a d-sided orthogonal range where $[x_1,-\infty] \times [x_2,-\infty] \times \ldots [x_d,-\infty]$. On the other hand, every input point $(x_1,...,x_i)$ can be thought of as an input range to rectangle stabbing in which it becomes $[x_1,\infty] \times [x_2,\infty] \times \cdots \times [x_d,\infty]$. It is quite obvious that these d-sided variants of orthogonal range reporting and rectangle stabbing are easier than the regular problems and hence we see that dominance reporting can be reduced to rectangle stabbing and orthogonal range reporting. Hence, we have equivalence between these three problems when a factor 2 increase in dimension is tolerable.

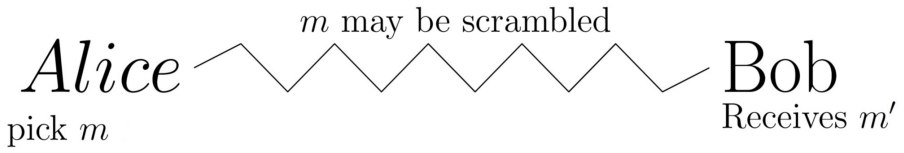

Figure 3.5: Alice and Bob communicating over an insecure line.

3.2 Faulty Communication

On our way to proving a lower bound for rectangle stabbing, orthogonal range reporting and dominance reporting, we encountered a connection to error correcting codes.

Error Correcting Codes

Formally we have two parties Alice and Bob. Alice wants to send a binary message m to Bob, but it goes through a faulty communication line and hence Bob receives m' which is a version of m where a number of bits have been flipped. There are two versions used to retrieve information about m. Error detection codes can detect if any bits have been flipped, i.e., it can detect whether $m = m'$. Error correcting codes can be used to retrieve m, given m'. Of course, there are limits to how powerful such an error correcting code can be and hence an error correcting code can retrieve the original message m if and only if the number of bits flipped is under some threshold. The origin of error correcting codes dates back to Richard Hamming who studied the phenomenon in the late 40'es [86]. Hamming invented the Hamming code which is a family of linear error correcting codes which interestingly can be used for both error detection and error correcting. These Hamming codes were presented in *A Mathematical Theory of Communication* [82] by Claude Shannon.

In our quest to achieve a lower bound in $c\log(n)$ dimension for rectangle stabbing, orthogonal range reporting and dominance reporting we obtained the following result related to error correcting codes:

Theorem 3.1. *Given r, d and h, such that* $r < \frac{d}{4}$*, it holds that for any* $N \leq (\frac{d}{r})^{rx}/(2e)$*, there exists a set S of N interesting strings such that for any h strings* $s_1, \ldots, s_h \in S$ *the string* $s_1 \vee \cdots \vee s_h$ *has more than M 1's, where M can be chosen to be* $\min\{\frac{d^y r^{1-y}}{e}, \frac{1}{2}rhz\}$*, for parameters x, y, and z such that* $x+y+z=1$ *and* $x,y,z > 0$*.*

Where we define an interesting string to be a binary string of length d and containing exactly r 1's. This immediately gives rise to a set of error correction codes. Alice and Bob agrees on a set of messages S from the above set. For Alice to send a message she chooses $m \in S$, now as long as $\forall s_i, s_j | i \neq j$ we have that $d(s_i, s_j) \geq 2\delta + 1$ (the distance function being the Hamming distance), Bob can recover any m as long as the number of bits corrupted during transmission is at most δ.

Now if s_i and s_j have exactly r ones, then the fact that $d(s_i, s_j) \geq 2\delta + 1$ is equivalent to their or having at least $r + 2\delta + 1$ ones, i.e., $|s_i \vee s_j|_1 \geq r + 2\delta + 1$.

Hence theorem 3.1 gives a set of error correcting codes if we set $h = 2$, where the pairwise hamming distance is atleast $M - r$. For $h > 2$ we are not aware of any application or interpretation in the context of error detection/correction.

As mentioned earlier, this connection arose as we were trying to design a lower bound for the rectangle stabbing, orthogonal range reporting and dominance reporting when the dimension is $c\log(n)$. The drawback of theorem 3.1 is that our construction is existential, hence there is no description of how to create these error correcting codes. Therefore, this result only states that such error correcting codes exists.

3.3 Lower Bound Construction

The underlying machinery of our lower bound is a framework by Afshani[10] and it has some similarities to the framework by Chazelle used in the jumbled indexing paper. The goal is to create t different shapes in $c\log(n)$ dimension. For each of these shapes, we tile the unit cube, thereby creating a cover of the unit cube. Any rectangle stabbing query inside the unit cube will result in at least t outputs, hence the output will be large. On one hand we need to ensure that if we take any h shapes, that their overlap is small, i.e., that the intersection of these shapes has a low volume. On the other hand if we take a number of random points inside the unit cube, the data structure needs to output many different shapes for each query point. This means that the data structure cannot "cheat" by storing the needed information closely together. This results in a trade-off between the space and query time.

The correlation to the error correcting codes stems from the fact that we can think of each shape as a binary string, where a one at position i corresponds to cutting the i'th dimension in half. On the other hand a zero corresponds to the i'th dimension spanning the entire unit distance. Now if any h shapes have a low intersection it corresponds to their binary strings having many 1's in their OR. Meaning that we can create binary strings such that for any h strings $s_1, s_2, \ldots s_h$ it holds that $s_1 \vee s_2 \vee \cdots \vee s_h$ contains more than M 1's. If we create our shapes in correspondence to these binary strings, the intersection of any of the h shapes will be at most 2^{-M}.

Proof by Contradiction

The proof is a non constructive proof, hence as mentioned earlier the construction of these error correcting codes is not given. The idea is to do a contradiction argument in which we get two different ways of counting the different ways of picking N strings of length d with r ones. One is the exact count of picking these N strings and the other is an over count which builds from the negated statement. Hence, by assuming the negated statement we get that the exact count is larger than the over count and thereby get a contradiction. The negated statement of theorem 3.1 becomes:

Statement 3.1. *Given r, d and h, then for all sets of N interesting strings there exists a subset $s_1, s_2, \ldots s_h$ such that $s_1 \vee s_2 \vee \cdots \vee s_h$ contains less than M 1's.*

The exact count is to consider all strings of length d with r ones and then counting all the ways of picking a set of N strings from these interesting strings. This gives the count $\binom{\binom{d}{r}}{N}$. When assuming the negated statement, we get that every set S of N interesting strings has a subset of h strings such that $s_1 \vee \cdots \vee s_h$ has less than M ones. The idea of the over count is to first choose these h strings $s_1, \ldots s_h$ and then pick the remaining strings of S. To choose $s_1, \ldots s_h$ we pick the string $s_0 = s_1 \vee \cdots \vee s_h$. We pick M positions that contain 1's in order to get s_0, which gives $\binom{d}{M}$. Now to pick $s_1, \ldots s_h$ we take the number of ways to pick h interesting strings among these M positions: $\binom{\binom{M}{r}}{h}$. Now we are left with picking the rest of S giving us $\binom{\binom{d}{r}}{N-h}$. Combining these three expressions gives our over count and by choosing the parameters accordingly we make sure that

$$\binom{\binom{d}{r}}{N} > \binom{d}{M}\binom{\binom{M}{r}}{h}\binom{\binom{d}{r}}{N-h}.$$

This results in a contradiction and thus the negated statement is refuted. With parameter fitting we get the query-space trade-off:

Theorem 3.2. *Consider a pointer machine data structure that solves the rectangle stabbing problem in $c\log(n)$ dimensions for some parameter $c \geq 4$ and achieves query time $Q(n) = o(n)$. Let γ be such that $Q(n) = n^{1-\gamma}$. If $\gamma \geq \frac{2}{2+\log c}$, then the data structure must use $\Omega\left(n^{1-\gamma} n^{\sqrt{c\gamma}/e - o(\sqrt{c\gamma})}\right)$ space.*

This states that if we want sub-linear query time any data structure for the rectangle stabbing, orthogonal range reporting or for the dominance reporting problem must use a large amount of space. There is still a gap between our lower bound and the upper bound by Chan, since our γ needs to be roughly $1/\log(c)$ in order for the construction to go through, while the upper bound of Chan achieves a query time with $\gamma = 1/(c\log(c))$. An interesting line of future work would be to work on bridging this gap.

Chapter 4

Hierarchical Data

As mentioned in section 1.1, data can be non-orderly in the sense that each point in the point set is, on top of their numerical values, associated with a category often referred to as their color. In previous works, the assumption has been that the categories themselves are independent, meaning that there is no relation between the different categories. This may not encapsulate real world examples since categories can have some inherent structure. If we look at the example from section 1.1, one could think of the categories as forming a hierarchical structure. Say the root node is the category American football, which has children which indicates the different countries in the world in which American football is played and every team playing in a country is a child of that country (this is what we denote as a category tree). Now a natural question could be how many teams in Denmark have a player which is taller than 2 meters and heavier than 130 kg. A way to formalise this is through what we denote as sub-category range counting (SCRC). In sub-category range counting, we are given a point set of n points in $\mathbb{R}^d$ where each point is associated with a category from an underlying category tree. Given a query range in $\mathbb{R}^d$ and a query node from the category tree we want to count the number of points lying in the query range and have a category in the sub tree rooted at the query node. For our example this would give us a query range of $[0,200] \times [130,\infty)$ and a query node of "Denmark".

Another natural question to ask upon a category tree is how many categories are "connected" to the points in the query range. A different way of phrasing this is to look at how many different ancestors the points within the query range has in the

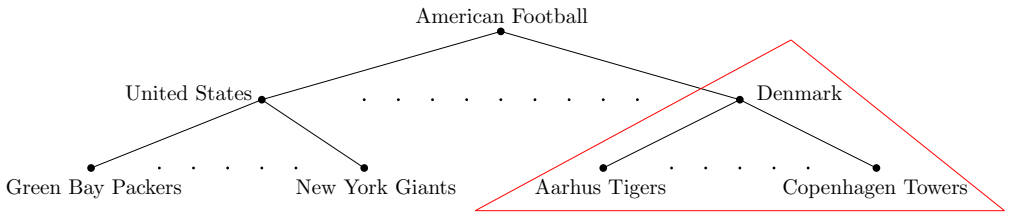

Figure 4.1: A category tree where we query the subtree rooted at "Denmark".

category tree. Given a point set in $\mathbb{R}^d$ and a category tree the goal is for a given query range to count the number of nodes in the category tree which can reach a node which is a category of a point within the query. This type of question we call hierarchical category counting (HCC).

For both SCRC and HCC they are atleast as hard as regular colored range counting. For HCC we present a simple proof by taking the difference of two output sizes from two different instances of HCC. For SCRC we can simply put all of our regular colors in the leaf nodes and use the root as the query node. In either case the regular colored range counting can be solved by our extended variants and since colored range counting is hard for 2D and above, we focus on SCRC and HCC in 1D.

Another interesting extension of colors where internal structure is added was studied by He and Kazi [58]. Here they count the number of colors/categories on the path between two query nodes in the underlying category tree.

4.1 Category Tree

We will start off by looking at both SCRC and HCC in 1D when the underlying category structure is a tree. For both problems we show interesting connections and equivalences to already well studied problems.

Sub-Category Range Counting

For the SCRC problem, we close a loop and show equivalence for the following five problems:

1. SCRC when the underlying category structure is a single path on a one-dimensional input set P.
2. 3-sided distinct y-coordinate counting for a point set P in $\mathbb{R}^2$.
3. 3-sided color counting for a point set P in $\mathbb{R}^2$.
4. Dominance color counting for a point set P in $\mathbb{R}^3$.
5. Dominance counting for a point set P in $\mathbb{R}^3$.

We observe that SCRC on a single path is equivalent to 3-sided distinct coordinate counting for a 2D point set. Given a single path and a 1D point set, we transform every point x_i of color c_j into the 2D point $(x_i, h(c_j))$, where $h(c_j)$ is the distance from the node c_j to the leaf on the single path. Now for a query range $[x_q, y_q]$ and a query node c_q we transform it into the 3-sided range $[x_q, y_q] \times (-\infty, h(c_q)]$. In the opposite direction, we transform every distinct y-coordinate into a color on the single path sorted in accordance to height and store each point with its x-coordinate and the corresponding y-color. For a 3-sided range $[x_q, y_q] \times (-\infty, z_q]$ we query the 1D point set with the range $[x_q, y_q]$ and query node as according to the y-coordinate $\lfloor z_q \rfloor$ (where $\lfloor z_q \rfloor$ is the closest y-color which is smaller than or equal to z_q).

3-sided distinct y-coordinate counting can be reduced to 3-sided color counting by creating a color for every distinct y-coordinate and assign each point its appropriate color.

3-sided color counting can be reduced to dominance color counting in 3D. The reduction goes by mapping every 2D input point (x_i, y_i) to the 3D point $(-x_i, y_i, x_i)$ and the 3-sided query range $[x_q, y_q] \times (-\infty, z_q]$ into the 3D dominance range $(-\infty, -x_q] \times (-\infty, z_q] \times (-\infty, y_q]$. Now it is not hard to verify that if a point is inside the 3-sided query range it is also dominated by the 3D dominance range. The colors of the points remain the same.

Dominance color counting in 3D was reduced to dominance counting in 3D by Saladi [81].

We close the loop by showing that Dominance counting for a point set in $\mathbb{R}^3$ can be reduced to 3-sided distinct y-coordinate counting in $\mathbb{R}^2$. The reduction is heavily influenced by the reduction by Walderveen and Larsen [69] from 2D range counting to 1D colored range counting as discussed in section 1.1.

Given a point set for dominance counting in $\mathbb{R}^3$, we transform the point set into rank space, meaning that every coordinate is integer and that the input coordinates are distinct. For an input point $p_i = (x_i, y_i, z_i)$ we create four points: $p_i^1 = (-x_i, z_i)$, $p_i^2 = (y_i, z_i)$, $p_i^3 = (-x_i, z_i)$ and $p_i^4 = (y_i, z_i + 0,5)$. We create two coordinate distinct counting structures and put p_i^1 and p_i^2 in the first structure and p_i^3 and p_i^4 in the second structure. Given a query point $p = (x, y, z)$, we create the 3-sided range $r_p = [x, y] \times [-\infty, z + 0,5]$. The claim is that the number of dominated points of p in the input is equivalent to the difference of the outputs of the two structures queried with r_p.

If p dominates p_i then $x \geq x_i$, $y \geq y_i$ and $z \geq z_i$, which means that $-x \leq x_i$ and $z_i + 0.5 \leq z + 0,5$, hence all four points are contained in r_p. The first data structure will count 1, since the two points share their second coordinate and the second data structure will count 2 since the two points have distinct second coordinates. If p does not dominate p_i at least one of $(-x_i, z_i)$ or (y_i, z_i) and $(y_i + z_i + 0,5)$ will not be in the range, which means that the difference of the two data structures output will always be zero. This concludes the reduction.

From [59] dominance counting can be solved using $O(n \log n / \log\log n)$ space and $O(\log n / \log\log n)$ query time. By splitting the tree into its heavy path decomposition we get a bunch of 1D single paths which we can solve through the reduction to dominance counting. Hence, we get a solution to SCRC in 1D on a category tree using $O(\log^2 n / \log\log n)$ space and $O(\log n / \log\log n)$ query time.

Hierarchical Colored Counting

As just shown for SCRC, we show some interesting connections to already established problems for HCC in 1D when the underlying category graph is a generalized caterpillar tree. We define a generalized caterpillar tree as a binary tree where all the degree three vertices lie on the same path. If the underlying category graph is a generalized caterpillar tree, we show that unweighted HCC in 1D is equivalent to weighted 2D

range counting and 1D colored range sum-max, i.e., summing the maximum weight across all colors within the query range[1]. The details of the proofs can be found in chapter 7.

For HCC on a 1D point set where the underlying category graph is a tree, we can use this result to gain a data structure. We split the tree into $O(\log n)$ layers of generalized caterpillar graphs by applying the heavy path decomposition and hence we can solve each of these layers simultaneous with a weighted orthogonal range counting data structure on $O(n \log n)$ points. This can be solved with $O(n \log^2(n)/\log\log n)$ space and $O(\log n/\log\log n)$ query time.

4.2 Category Directed Acyclic Graph

We now shift our attention to both SCRC and HCC in $\mathbb{R}$ but where the underlying category graph is a directed acyclic graph (DAG). Obviously, this makes the problem more challenging, but how much harder is this version as compared to the tree version? These questions becomes quite a lot harder, since we can reduce the infamous orthogonal vectors problem to both problems.

Reduction from Orthogonal Vectors

We will now reduce the orthogonal vectors problem to sub-category range counting. From the orthogonal vectors hypothesis, presented in section 1.2, this will give us a quadratic lower bound.

We need to build a category graph and an input point set to represent the orthogonal vectors problem. To build the category graph we need to build three layers of nodes. The first layer is going to consist of n nodes[2], one for every vector in A. The second layer is going to have a node for each dimension d and the third and final layer is going to consist of n nodes, one for every vector in B. For every node n_i in layer one, we create an edge to the j'th node in layer two if the i'th vector in A has a 1 in coordinate j. For every node in layer two, we create an edge to all nodes in the third layer for which the corresponding vector in B has a 1. The input point set is going to be n points where each is assigned a unique category from the third layer.

The claim is now that a vector $a \in A$ and a vector $b \in B$ are orthogonal if there is no path from the node corresponding to a in the first layer to the node corresponding to b in the third layer. Being orthogonal means that they do not share a 1 at any coordinate and therefore it follows immediately from our construction that they are orthogonal iff there is no such path between the first and third layer.

Now to check if there exists any orthogonal pair, we query n times with an interval spanning over all input points, but with a new node from the first layer in every iteration as our query node. Now if any query outputs less than n, we know that there

[1] If no point with color c_i exists, it adds nothing to the sum.

[2] In the paper version, we choose $\eta = n/\log^c(n)$ instead of n such that the total graph size is $O(n)$. To simplify the proof we use n throughout here.

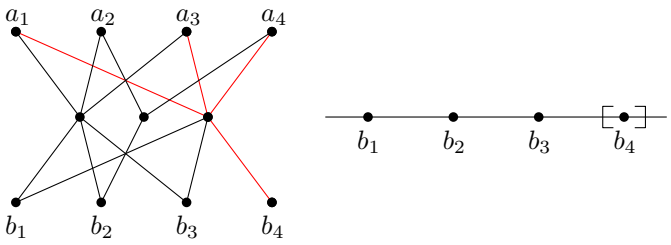

Figure 4.2: An example for HCC in 1D with four vectors in $\mathbb{R}^3$. Every query up until b_4 had an output of $4+|b_i|_1+1$, but when querying b_4 the output is $3+|b_4|_1+1 < 4+|b_4|_1+1$, hence b_4 is orthogonal to a_2.

are two orthogonal vectors. This gives us a conditional lower bound for SCRC in 1D with a DAG as a category graph of $P(n)+nQ(n) \geq n^{2-o(1)}$.

The reduction needs a slight modification in order for it to work for HCC. We create the same graph and input set, but instead of querying all input points we query every input point individually. Now for a point p_i with category b_i, if the query output becomes less than $n+|b_i|_1+1$ (where $|b_i|_1$ counts the number of ones in vector b_i) we know that atleast one vector in the first layer was not counted in the output and hence must be orthogonal to b_i. This again gives us a conditional lower bound of $P(n)+nQ(n) \geq n^{2-o(1)}$. An example with four vectors in both A and B in $\mathbb{R}^3$ is given in fig. 4.2.

Upper Bounds

At the first glance, and with this conditional lower bound in mind, it seems hopeless to find a solution with $\tilde{O}(n)$ space and polylog query time, or maybe not. The conditional lower bound states that such a solution needs $\Omega(n^2)$ preprocessing time. Although we do not achieve $\tilde{O}(n)$ space, we presented a solution with $\tilde{O}(n^{3/2})$ space for HCC in 1D on a DAG by compressing the initial data structure from the preprocessing phase. All in all, we present a data structure for the HCC problem in 1D with an underlying DAG category graph, which uses $\tilde{O}(n^2)$ preprocessing time, $\tilde{O}(n^{3/2})$ space and $O(\log n)$ query time. Thereby obtaining a polynomial gap between the preprocessing time and space usage of the data structure.

For SCRC there is a trivial upper bound of $O(n^2)$ space and $O(\log n)$ query time. For each node v_i in the category graph, store a one dimensional colored range counting structure for all nodes with colors reachable from v_i. To answer a query (I, v_q) we query the regular colored range counting structure on the structure for v_q with interval I. As opposed to the HCC, where we could optimize the trivial solution, we were not able to improve upon this trivial upper bound.

Part II

Publications

Chapter 5

A Lower Bound for Jumbled Indexing

A Lower Bound for Jumbled Indexing

Peyman Afshani* Ingo van Duijn† Rasmus Killmann‡
Jesper Sindahl Nielsen§

July 29, 2022

Abstract

In this paper we study lower bounds for a variant of *jumbled-indexing* problem: given an input string S on an alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_\lambda\}$, store it in a data structure such that given frequencies $f_1, \cdots, f_\lambda$, one can find all the substrings S' of S where the frequency of the character σ_i is f_i, for $1 \leq i \leq \lambda$. This is a very interesting and challenging text indexing problem and it has received significant attention lately, both in the string-indexing community as well as in the theory community. It is known that when $\lambda = 2$, one can use a linear-size data structure to decide whether there exists a substring that matches the query in constant time [46]. It is also known [23] that for constant $\lambda \geq 3$, then there exists a constant α_λ such that it is not possible to achieve both $O(n^{2-\alpha_\lambda})$ preprocessing time and $O(n^{1-\alpha_\lambda})$ query time.

We study a variant of the problem where the goal is to *report* all the substrings of S that match a given jumbled-indexing query. Assuming the data structure operates in the pointer machine model, we prove *unconditional space* lower bounds that also apply to *binary* alphabets: we show that if the data structure has the query time of $O(n^{0.5-o(1)} + k)$, where k is the output size of the query, then it must consume $\Omega(n^{2-o(1)})$ space. The $o(1)$ term in our lower bound is at most $\frac{1}{\log^{(10)} n}$, where $\log^{(\cdot)}$ notation denotes the iterative log function (i.e., $\log^{(10)} n = \log\log\log\log\log\log\log\log\log\log n$).

This result has a number of interesting consequences. First, it shows that reporting all the matches is significantly harder than deciding whether a match exists, at least for the binary alphabet. This was surprising for us since we believed that in order for jumbled-indexing to be difficult, the alphabet size must be at least 3. Second, from a technical point of view, we make connections between

*Aarhus University. peyman@cs.au.dk. Supported by DFF (Det Frie Forskningsråd) of Danish Council for Independent Research under grant ID DFF−7014−00404.

†Aalborg University, ingo@cs.aau.dk

‡Aarhus University. killmann@cs.au.dk. Supported by DFF (Det Frie Forskningsråd) of Danish Council for Independent Research under grant ID DFF−7014−00404.

§Aarhus University, jasn@cs.au.dk

this problem and the area of additive combinatorics as well as random walks. Previously, Chan and Lewenstein [37] had shown the connections between this problem and the area of additive combinatorics from an upper bound point of view, however, we use fundamental theorems from additive combinatorics but these tools are quite different from the theorems used in [37]. In our opinion, the fact that additive combinatorics appears in our lower bound approach as well as in the upper bound approach of Chan and Lewenstein [37] is noteworthy and demands further investigation.

We believe our results open up a few new venues for research. For example, we believe it is interesting to study whether it is possible to build a data structure that uses $O(n)$ space s.t., it can report all the matches to a jumble-indexing query in $O(n^{2/3}+k)$ time; this is even interesting for a binary alphabet.

5.1 Introduction

In this paper, we study lower bounds for a challenging string indexing problem known as *Jumbled indexing* or *histogram indexing*. Here, unlike the usual types of pattern matching, we are concerned with the frequency of the letters in the match. To be specific, let S_1 and S_2 be two strings over an alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_\lambda\}$. We say S_1 and S_2 *jumble-match* if they are commutatively equivalent, i.e., if one string can be obtained from the other one by permuting its characters. Here, we will only use the word *match* to denote the notion of a jumble-match. A common way to describe a match is to use the notion of *Parikh vectors*. The Parikh vector $\psi = (f_1, \cdots, f_\lambda)$ of a string S is the frequency of the alphabet characters in a given string, i.e., f_i is the number of occurrences of the character σ_i in S and thus two strings match if and only if they have the same Parikh vector.

Parikh vectors were named after Rohit Parikh, and they were initially studied due to their connection to context-free languages [74]. The concept of match, a.k.a histograms matching, has a wide range of applications such as computational biology and bioinformatics [27, 29]. See also [33] for more references and also the discussion of jumble matching in mass spectrometry.

Previous Results

From a theoretical point of view, the notion leads to a number of natural problems, such as, *jumbled pattern matching* where the input is a string S of length n and a Parikh vector $\psi = (f_1, \cdots, f_\lambda)$ and the goal is to know whether S matches ψ. This can be solved easily with a sliding window algorithm in $O(n \log \lambda)$ time: initiate a conceptual "window" $f_1 + \cdots + f_\lambda$ characters wide at the beginning of the string and just compute the frequency of each character in the window. Sliding the window, one character to the right changes the frequencies of at most two characters in the window and they can be updated in $O(\log \lambda)$ time. The result is a straightforward and efficient algorithm for jumbled pattern matching. Typically pattern matching problems require non-trivial ideas for an efficient solution (e.g., see the classical pattern matching algorithm of Knuth et al. [63]) so this is a rather unusual case.

However, we can also consider jumble-matching as a data structure problem: preprocess an input string S in a data structure for matching queries. This leads to the problem of *jumbled indexing* that unlike the jumbled pattern matching problem ends up being much more challenging and difficult to answer. Jumbled indexing has received quite a bit of attention lately as it leads to various intriguing problems. We can even identify additional variants of this problem: in the most popular variant, one that we call the *the existential variant*, we simply want to know if S contains a substring that matches ψ, in *the reporting variant* we want to report all the positions in S where a match occurs or *the counting variant* where we want to output the number of matches. All these problems can be solved trivially by the sliding window method which requires no preprocessing time and it has the query time of $O(n \log \lambda)$. At the other extreme, we can also solve all the three variants by merely storing all the $O(n^2)$ substrings of S and given a query ψ, we can simply look up the answers in optimal time. In general, it seems difficult to make progress on this problem beyond these trivial solutions.

Nonetheless, it turns out that the existential variant can be solved efficiently for a binary alphabet (i.e., $\lambda = 2$) using a clever idea: the sliding window algorithm only needs to keep track of one frequency (e.g., number of 0s) so just record the minimum and the maximum number of 0s for every window size to get a linear-sized data structure. Then, given a query, look up to see if its number of 0s is between the minimum and the maximum number of 0s ever seen in a window of that size, for an $O(1)$ query time.

Furthermore Kociumaka et al. [64] provided an algorithm for the existential variant for a constant sized alphabet, having space $O(n^{2-\varepsilon})$ and query time $O(m^{\varepsilon})$, where m is the pattern length (note that m could be $O(n)$). With their result and the lower bound presented in this paper the reporting variant seems harder than the existential variant.

This, however, only solves the existential variant of the jumbled indexing, and not the reporting or counting variants. Interestingly, even the simple idea for the binary case has a lot of nuances. This idea was first used in [46] to give a data structure of linear size, but the preprocessing time was quadratic. In a series of results, the processing time was improved by polylog factors [33, 70, 71] and the jumbled indexing problem was connected to other fundamental problems such as $(\min,+)$-convolution and $(\min,+)$-matrix multiplication. These connections also made it possible to improve the preprocessing time by more than polylog factors by applying Ryan Williams breakthrough idea [89] but the latest breakthrough came by Chan and Lewenstein [37] where they showed "bounded universe" version of $(\min,+)$-convolution can be solved in $\tilde{O}(n^{1.859})$ time and subsequently, this also reduces the preprocessing time of the jumbled indexing problem by polynomial factors rather than polylogarithmic. The improvement, in fact, applies to alphabets of constant size rather than just binary, although for non-binary alphabets the query is no longer constant or logarithmic but is bounded by $O(n^{1-\alpha})$ for some constant α. Interestingly, the results used ideas from additive combinatorics.

On the lower bound side, Amir et al. [23] proved conditional lower bounds that

showed this trade-off is essentially best possible: they showed that if the size of the alphabet is a constant $\lambda \geq 3$, there is a constant α_λ such that it is not possible to get both $O(n^{2-\alpha_\lambda})$ preprocessing time and $O(n^{1-\alpha_\lambda})$ query time. Not only is their lower bound conditional, and it does not apply to binary alphabets, it also says nothing about the space requirement of the data structures. This is actually very important as the case of the binary alphabet shows that it could be possible to solve the problem with linear space while the preprocessing time could be large.

Our Results

In this paper, we consider the *reporting variant* of the problem. We prove that in the pointer machine model, if a data structure can report all the k matches to a jumbled indexing query in $O(n^{0.5-o(1)}+k)$ time, then it needs to use $O(n^{2-o(1)})$ space. This lower bound is *unconditional* (unlike the previous ones), it also lower bounds the *space* of the data structures (again, unlike the previous ones), and perhaps most importantly, it applies to *binary* alphabets. This latter point shows that reporting all the matches is significantly harder than deciding whether a match exists, at least for the binary alphabet. This was surprising for us since we believed that in order for jumbled-indexing to be difficult, the alphabet size must be at least 3.

The drawback of our lower bound is that the data structure is required to operate in the pointer machine model. Our lower bound model is somewhat more restrictive than the RAM model since it requires the data structure to operate through pointers instead of having constant time access to the elements stored but on the other hand computation and information is free in our lower bound model. One upside is that our lower bounds are unconditional. Another upside is that our approach provides a hard input instance. Conditional lower bounds on the other hand are proved through reductions and therefore no hard instance is presented. This means that pointer machine lower bounds and the conditional lower bounds can be viewed as complementary as they tell us different things, each increasing our understanding of the problems involved differently.

An interesting technical aspect of our lower bound is that some fundamental theorems from additive combinatorics also appear in our proof. This is somewhat similar to [37], but the tools that we use are quite different from those that appeared in [37], likely because we are dealing with a lower bound question. Nonetheless, the fact that additive combinatorics appears in our lower bound approach as well as in the upper bound approach of Chan and Lewenstein [37] is noteworthy and perhaps not a coincidence.

We believe our results open up a few new venues for research. For example, we believe it is interesting to study whether it is possible to build a data structure that uses $O(n)$ space s.t., it can report all the matches to a jumble indexing query in $O(n^{2/3}+k)$ time; this is even interesting for a binary alphabet.

5.2 Technical Preliminaries

Here, we review some of the technical tools and preliminaries that we need for our lower bound. We will need to use some of the existing lower bound framework in the pointer machine model in addition to some (heavy) tools from additive combinatorics as well as ideas from random walks.

Framework and Model of Computation

The Pointer Machine Model ([85]) models data structures that solely use pointers to access memory locations (e.g., any tree-based data structure). The focus of this paper is on a variant that is often used when proving lower bounds. Consider an abstract "reporting" problem where the input is a set $\mathscr{U}$ and each query q reports a (implicitly defined) subset, $q_{\mathscr{U}}$, of $\mathscr{U}$. The data structure is represented as a directed graph G with a root node $r(G)$, where each vertex corresponds to one memory cell with out degree two. Each vertex can store one element from the universe $\mathscr{U}$ and edges between vertices represent pointers between memory cells. All additional information can be stored and accessed for free by the data structure. The only restriction is that given a query q, the data structure must start at the root node $r(G)$ and explore a connected sub graph $G' \subseteq G$ and such that each element of $q_{\mathscr{U}}$ is stored in at least one vertex of G'. The size of G' gives a lower bound on the query time and the size of the graph G gives a lower bound on the space required to solve the reporting problem. The first lower bound framework in the pointer machine model was given by Bernard Chazelle [42, 44]. This is the framework we will apply to the reporting variant of jumbled indexing.

Theorem 5.1. *Consider a reporting problem on a set S of n input elements and assume a data structure of size $S(n)$ achieves the query time of $Q(n) + \gamma k$ for the problem where k is the output size and γ is a fixed parameter (typically a constant). If there exists a set $\mathscr{Q} \subset 2^S$, where each set $q \in \mathscr{Q}$ have a corresponding query who outputs the elements in q, such that the following hold.*

(cond. I) *For every $q \in \mathscr{Q}$, $|q| = \Omega(Q(n))$.*

(cond. II) *For any α queries $q_1, ..., q_\alpha \in \mathscr{Q}$, we have $|q_1 \cap ... \cap q_\alpha| \leq \beta$.*

Then $S(n) = \Omega(\frac{\sum_{q \in Q} |q|}{\alpha 2^{O(\beta+\gamma)}})$.

Note that from the way we define $\mathscr{Q} \subset 2^S$ and with a slight abuse of notation, we consider each element of $\mathscr{Q}$ as a query and we denote $\mathscr{Q}$ as the set of difficult queries.

Modelling jumbled-indexing. $S = x_1, \cdots, x_n$ is our input string on an alphabet Σ of size λ. We make the convention that the data structure only need to find a pointer that points to the first character of each match (e.g., just x_i instead of the entire substring $x_i \cdots x_j$). This is well-defined since for a fixed query ψ at most one substring can

match ψ from each starting position. Also, since all information other than pointers are free in our model, we can assume the memory graph G only stores pointers to characters of S. So, if a query ψ matches positions $i_1, \cdots, i_k$, then the data structure should find k memory cells that store pointers to characters $x_{i_1}, \cdots, x_{i_k}$.

Additive Combinatorics

Let A and B be subsets of an additive group $\mathscr{G}$, e.g., the set of positive integers $\mathbb{Z}_{\geq 0}$. The sumset of A and B, denoted by $A+B$ is defined as the set $\{a+b | a \in A, b \in B\}$. Furthermore, a set $C \subset \mathscr{G}$ is a k-term arithmetical progression if $C = \{c+id : 0 \leq i \leq k-1\}$ for some values $c, d \in \mathscr{G}$. The value d is known as the *common difference* of the arithmetic progression.

One of the fundamental results in additive combinatorics is an explicit bound on the maximal number of integers, $r_k(n)$, that can be selected from the interval $[1,n]$ without including a k-term arithmetical progression. This concept has a very old and rich history. In 1936, Erdős and Turán conjectured that we must have $r_k(n) = o(n)$, meaning, any sufficiently "dense" subset of $[1,n]$ must contain a k-term arithmetic progression. This was proven by Szemerédi [84] in a celebrated and influential result in 1975. Szemerédi's bound of $r_k(n) = o(n)$ suffices for our lower bound, however, his proof does not show an explicit bound for the value of $w_k(n) = \frac{n}{r_k(n)}$. So instead, we will use Gowers' breakthrough result that does in fact express an explicit bound, as this would give us a more quantifiable lower bound. Gower's groundbreaking result [54] introduced a number of novel concepts in additive combinatorics and also gave an explicit bound for the ratio between n and $r_k(n)$, and it was one of the primary reasons he was awarded the prestigious fields medal.

Theorem 5.2. *For every positive integers k there is a constant $c = c(k) > 0$ such that every subset of $\{1,2,...,N\}$ of size at least $N(\log\log N)^{-c}$ contains an arithmetic progression of length k. Moreover, c can be taken to be $2^{-2^{k+9}}$.*

Szemerédi's proof, combined with results of Balog and Szemerédi [26] (or Freiman [52]) would show that if $A+B$ is small, then A contains an arithmetic progression. Here, we will use a slightly different presentation of this fact by Rusza [80].

Theorem 5.3. *Let A and B be subsets of an additive group $\mathscr{G}$ such that $|A| = |B| = m$. If we have*

$$|A+B| < \frac{1}{\sqrt{2}} w_k(m)^{1/4} m$$

then A contains a k-term arithmetic progression.

5.3 The Lower Bound

This section is devoted to proving our lower bound result for ternary alphabets. In section 5.4 we will generalize upon this proof to extend the lower bound to the binary

case. We chose to do this since an argument that uses three characters conveys a lot of the major ideas, and it is much cleaner. Nonetheless, dealing with a binary alphabet has a lot of unique challenges and thus it requires a number of non-trivial ideas and we will not encounter any of them here. At any rate, our focus is to prove the following theorem.

Theorem 5.4. *For any data structure working in the pointer machine model, which tries to solve the reporting jumbled indexing problem, where $\lambda \geq 3$ and achieves query time $Q(n)+k = O(n^{0.5-o(1)})+k$, where k is the output size, must use $S(n) = \Omega(n^{2-o(1)})$ space.*

We will use the lower bound framework introduced in Subsection 5.2. For this end, we will need to first construct an input string.

Our Construction

Notations and conventions. Our input string S will be constructed randomly. Our construction depends on a (constant) number of events that each holds with high probability, and here the high probability is taken to mean with probability at least $1 - O(1/n)$. As we have a constant number of such events, it follows that all of them hold with high probability. To unclutter the proof, we will implicitly assume that all such events hold, instead of conditioning each statement on all of these events.

The input string. We start off by constructing S. The size of our string is $\Theta(n)$ and we use an alphabet of size three, that is, $\Sigma = \{0, 1, \#\}$. Our construction splits the string in two parts, this is done by putting two characters ## in the middle of the string. In the first half of the string we repeat $\frac{n}{s}$ times the pattern $\#(01)^s$, for a parameter s to be determined later; in other words, in the first half, we have $\frac{n}{s}$ "chunks" and each chunk starts with #, then continues with s 01 two bit chunks. The second half is slightly more complicated. In the second half we also have $\frac{n}{s}$ chunks but these chunks are created differently: each chunk is a string of length $2s+1$, e.g., $x_1 x_2 \cdots x_{2s+1}$ and the last character of the chunk is set to #, i.e., $x_{2s+1} = \#$. Furthermore, for each odd index i, we pick x_i randomly and uniformly in $\{0, 1\}$. For each even index i, we set $x_i = x_{i-1}$ (see Figure 5.1 for an example). Another way of describing this process is to say that we can start with a uniformly and randomly selected 01 string of size s, e.g., $d_1, \ldots, d_s$, then duplicate each character to obtain the string $d_1 d_1 d_2 d_2, \ldots, d_s d_s$ and then append the character # to the end. Clearly, the size of S is $\Theta(n)$.

$$S = \overbrace{\#(01)^s\#(01)^s\#(01)^s \ldots \#(01)^s}^{\text{Repeating pattern of } \#(01)^s}\#\#\overbrace{00111100\ldots00\#\ldots\#11001100\ldots11\#}^{\text{Random pattern of 00 and 11 in chunks of size } 2s}$$

Figure 5.1: The input string S

Defining the Queries

In this subsection we will focus on cond. I from 5.1, hence we want to show that there exists a query set $\mathcal{Q}$ where each query has output size at least $\Omega(\sqrt{n}/s\sqrt{\log n})$, where s is a parameter to be determined later. We first discuss how to pick our query set.

Define the concept of a *valid substring* to be a substring $x_i, ..., x_j$ where $x_i = \#$, $x_j = \#$ and there exists $x_k = \#$, $x_{k+1} = \#$ where $i < k$ and $k+1 < j$. Another way of phrasing this is to say that a valid substring starts and ends on the character # and includes the middle two ##. It follows that the number of valid substrings in S according to this definition is $(\frac{n}{s})^2$, since we have n/s valid starting positions and for each starting position we have n/s ending positions. For a fixed valid substring we can prove the following Lemma (see the appendix for the full proof). The *balance* of a substring is defined as (the absolute value of) the difference between the number of 1s and 0s in the substring.

Lemma 5.1. *For a fixed valid substring, the probability that the substring has balance larger than $\sqrt{n}/\delta$ is less than $2n^{-c}$, for $\delta = \frac{1}{\sqrt{c\log n}}$.*

The probability in the above Lemma is bounded by $2\frac{1}{n^4}$ if we set $c = 4$. Since the number of valid substrings is $(\frac{n}{s})^2$, using union bound we have that with high probability (i.e., probability at least $1 - 1/n$), for every valid substring with x_0 0s and x_1 1s we have $|x_0 - x_1| \leq \sqrt{n}/\delta$. In the rest of the proof, we assume that this event holds for any valid substring.

We now build our set of "difficult" queries $\mathcal{Q}$. Each query $q_i \in \mathcal{Q}$ is defined by a 3-dimensional Parikh vector $\psi_i = (\psi_i(0), \psi_i(1), \psi_i(\#))$. q_i is (the set of) all the starting positions of the substrings that match ψ_i, meaning, in each substring the frequency of the character $v \in \{0, 1, \#\}$ is $\psi_i(v)$. The data structure is required to output all elements of q_i (see Subsection 5.2), given the query ψ_i. In our construction, ψ_i will be chosen such that any matching substring must include the middle ## characters and it should start and end with #. This is done in the following way.

Definition 5.1. *We say a Parikh vector $\psi = (\psi(0), \psi(1), \psi(\#))$ is valid if*

(i) $\psi(0) + \psi(1)$ *is a multiple of* $2s$,

(ii) $\psi(\#) = \frac{\psi(0)+\psi(1)}{2s} + 2$ *and*

(iii) $|\psi(0) - \psi(1)| \leq \sqrt{n}/\delta$.

Our set of difficult queries will be a subset of all the valid Parikh vectors (more details to follow).

Lemma 5.2. *Any substring of x that matches a valid Parikh vector ψ, starts and ends with # and it contains the middle character ##. Also, any valid substring matches some valid Parikh vector w.h.p.*

Proof. Observe that between every two consecutive # characters, except the two in the middle, there are $2s$ 0s and 1s. As a result, any substring x_{ij} of x that contains k instances of #, must also contain at least $(k-2)2s$ many 0s and 1s. The equality only holds if x_{ij} starts with # and ends with # and also includes the two # characters in the middle. The last claim in the Lemma follows from the definition of a valid Parikh vector and Lemma 5.1. □

To settle cond. | in Chazelles framework 5.1, we introduce the following definition.

Definition 5.2. *We say a valid Parikh vector ψ is bad if it matches less than $\frac{\sqrt{n}\delta}{4s}$ valid substrings of S, and good otherwise. Similarly, a valid substring that does not match a good Parikh vector is called bad.*

Remember that the number of valid substrings is $\left(\frac{n}{s}\right)^2$ and each valid substring matches a valid Parikh vector w.h.p. However, a bad Parikh vector can only match $\frac{\sqrt{n}\delta}{4s}$ substrings and since the number of valid Parikh vectors is at most $\frac{n}{s}\frac{\sqrt{n}}{\delta}$, the total number of substrings that can match bad Parikh vectors is $\frac{\sqrt{n}\delta}{4s} \cdot \frac{n}{s}\frac{\sqrt{n}}{\delta} = \frac{1}{4}\left(\frac{n}{s}\right)^2$. So at most one fourth of the valid substrings can match a bad Parikh vector. This implies at least three fourths of the valid substrings match a good Parikh vector. We define our query set $\mathcal{Q}$ as the set of all good Parikh vectors. This means that for all $q \in \mathcal{Q}$ the output size $|q|$ is at least $\frac{\sqrt{n}\delta}{4s} = \frac{\sqrt{n}}{8s\sqrt{\log(n)}}$, by definition of a good Parikh vector. This settles cond. | in Chazelles framework 5.1.

The probability of having a balanced string. Let $T = t_1 t_2 \cdots t_m$ be a random string of 01, i.e., each t_i is uniformly set to 0 or 1. We prove that for any fixed value $0 \leq i \leq n$ the probability of having exactly i 0s in T is at most $\Theta(1/\sqrt{m})$. To prove this it suffices to consider the case when $i = m/2$. In the following, we will assume that m is an even number. The proof of the following is in the appendix.

Lemma 5.3. *The probability of having i 0s in a random binary string of length m is at most $\Theta(1/\sqrt{m})$.*

Proving the Lower Bound

In subsection 5.3 we showed that each query in our set of queries $\mathcal{Q}$ has an output size of $\Omega(\frac{\sqrt{n}}{s\sqrt{\log(n)}})$ which settled the first condition in Chazelles framework 5.1. Here, we focus on the second condition which states that for any α queries in $\mathcal{Q}$, the size of their intersection must be less than β, i.e., for any $q_1, .., q_\alpha \in \mathcal{Q}$ we must have $|q_1 \cap ... \cap q_\alpha| < \beta$. In this subsection, we will prove that if α and β are chosen properly, then with high probability there will be no α query that match β positions.

To do that, we assume the contrary: assume α queries match at the same β positions $p_1, p_2, ..., p_\beta$. Ultimately, we will choose $\alpha = \beta$ but to keep the proof clear, we will for now treat them as different parameters. A query q_j matches S from position p_i until position $p_i + |q_j| - 1$ as well as from position p_{i+1} until $p_{i+1} + |q_j| - 1$.

From our construction of our input string we have that the interval from p_i to p_{i+1} is balanced for all $i \in \{1,2,...,\beta-1\}$, and this implies that the interval from $p_i+|q_j|$ to $p_{i+1}+|q_j|-1$ is also balanced. Lets order the queries such that[1] $|q_1| < |q_2| < ... < |q_\beta|$. This means that $p_1+|q_1|$ is the leftmost endpoint which is to the right of the middle of the string.

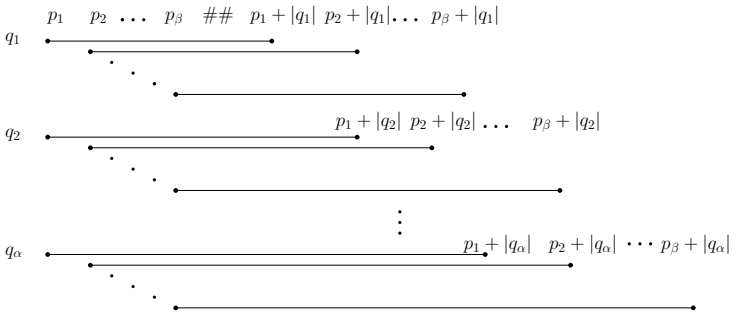

Figure 5.2: Overview of α queries matching at β positions.

To prove our desired high probability bound, we need a notion of elementary intervals.

Definition 5.3. *Let $A = \{|q_1|,\cdots,|q_\alpha|\}$ and $B = \{p_1,\cdots,p_\beta\}$ and consider the set $D = A+B$. Let $D = \{d_1,\cdots,d_M\}$ be the elements of D in sorted order. An elementary interval is the interval represented by two consecutive integers in D, specifically, it is defined as $[d_i, d_{i+1}-1]$ for $1 \le i \le M$.*

Let b_i be the balance in the interval from $p_1+|q_1|$ to $p_1+|q_i|$ for $i \in \{2,..,\alpha\}$. We can make the following crucial observation (for the full proof, see the appendix).

Lemma 5.4. *Given $p_1,...,p_\beta$, $|q_1|,|q_2|,...,|q_\alpha|$ and $b_2,b_3,...,b_\alpha$ it is possible to exactly calculate the balance of all the elementary intervals.*

Observe that the elementary intervals are completely disjoint and they all lie in the second half of the string S which is essentially a random string. Intuitively, this means that the balance of each elementary interval is essentially a random value (not uniform however). Our idea here is that knowing the values $p_1,...,p_\beta$, $|q_1|,|q_2|,...,|q_\alpha|$ and $b_2,b_3,...,b_\alpha$ should not help us deduce all of these random balance values. To do that, we consider two cases: if M is large, then we can show that this essentially follows from a relatively simple argument and by applying the union bound. However, M could be small enough to break our first union bound. In this case, we turn to additive combinatorics and observe that this case implies that the set B must contain an arithmetic progression. Next, we argue that it is very unlikely for any valid Parikh vector to match S at positions that form an arithmetic progression. By combining these two case, we manage to show that it is very unlikely for α queries to match at β positions, for some proper choice of parameters.

[1]Note that the inequalities are strict as otherwise two queries would be identical.

Many Elementary Intervals

We will start off by looking at the case where the number of elementary intervals is large, i.e., $M \geq \frac{4}{\varepsilon}(\alpha+\beta)$, where ε is a variable to be determined later.

Theorem 5.5. *If $M \geq \frac{4}{\varepsilon}(\alpha+\beta)$, the probability of α queries matching the same β positions is $\Theta(n^{-\beta})$.*

Proof. Consider a fixed set of values of $p_1,...,p_\beta,|q_1|,...,|q_\alpha|$ and $b_1,...,b_\alpha$. Since $p_1,\cdots,p_\beta$ and $|q_1|,\cdots,|q_\alpha|$ are fixed values, the position of each elementary interval is also fixed. From Lemma 5.4, we also know that the balance in each elementary interval is also fixed. One main observation here is that each elementary interval has size at least $2s$ since queries must match at positions that start with # and these positions are at least $2s$ characters apart. Now, look at the probability of our input string having exactly this set of fixed balance values at this fixed set of intervals: We have M elementary intervals of size at least $2s$, they all have a fixed balance but we know from Lemma 5.3 that the probability of having this balance is at most $O(1/\sqrt{s})$ (notice that if an interval is bigger than $2s$, this probability will only become smaller). The intervals are disjoint, and thus the probability of having a fixed balance is independent, hence the probability for a random string to have all of the balances in the M elementary intervals is no more than $O((1/\sqrt{s})^M)$. We now use union bound and consider all possible combinations of values $p_1,...,p_\beta,|q_1|,...,|q_\alpha|$ and $b_1,...,b_\alpha$: we observe that for each one we have at most n choices and thus in total there are $n^{2\alpha+\beta}$ combinations of numbers that corresponds to these values. This means, the overall probability is bounded by

$$O\left(n^{2\alpha+\beta}\cdot\left(\frac{1}{\sqrt{s}}\right)^M\right) \leq O\left(n^{2\alpha+\beta}\cdot n^{-\frac{\varepsilon}{2}\left(\frac{4}{\varepsilon}(\alpha+\beta)\right)}\right) = \Theta\left(n^{-\beta}\right).$$

Where $s = n^\varepsilon$, $M \geq \frac{4}{\varepsilon}(\alpha+\beta)$ and ε is a parameter to be determined later. □

Few Elementary Intervals

We will now focus on the case when the number of elementary intervals is small. This means that we assume $M \leq \frac{4}{\varepsilon}(\alpha+\beta)$. The idea here is to use tools from additive combinatorics and argue that there must exist an arithmetic progression between the set of starting positions. Then, we show that for a proper choice of parameters, this is unlikely. Together with the result from section 5.3, this resolves the second condition from Chazelles framework 5.1.

Remember that we have $A = \{|q_1|,\cdots,|q_\alpha|\}$ and $B = \{p_1,\cdots,p_\beta\}$ and $M = |A+B|$. We set $\alpha = \beta = m = \log\log n$ and furthermore we fix $\varepsilon = 1/\log^{(10)}(n)$ where the $\log^{(\ell)} n$ function is the iterated log function. Observe that with this choice of parameters, our construction is now fully described and we have no other parameters left to play with. Remember that we are dealing with the case $M \leq \frac{4}{\varepsilon}(\alpha+\beta)$ which is equivalent to the condition $|A+B| \leq \frac{4}{\varepsilon}(|A|+|B|)$. We claim that B contains a

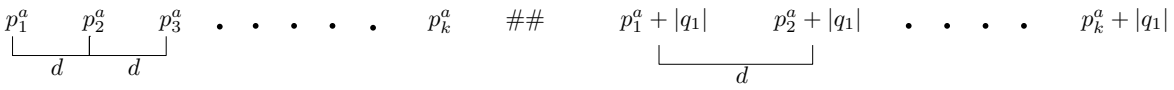

Figure 5.3: Figure of k-arithmetic progression with common difference d among points.

k-arithmetic progression, where $k = \log^{(6)}(m)$. To prove this claim, we first verify that for large enough n, we have $w_k(m) = (\log\log m)^{\frac{1}{2^{2^{k+9}}}} \geq \log^{(7)}(m)$:

$$w_k(m) = (\log\log m)^{\frac{1}{2^{2^{k+9}}}} \geq \log^{(7)}(m) \Leftarrow \frac{1}{2^{2^{k+9}}}\log^{(3)}(m) \geq \log^{(8)}(m) \Leftarrow$$
$$\log^{(4)}(m) - 2^{k+9} \geq \log^{(9)}(m) \Leftarrow \log^{(4)}(m) - \log^{(9)}(m) \geq 2^{k+9} \Leftarrow \log^{(6)}(m) = k.$$

From 5.3 we know that B contains an arithmetic progression when $|A+B| \leq \frac{1}{\sqrt{2}}w_k(m)^{1/4}(m)$. So, to show that B contains a k-term arithmetic progression, it suffices to verify that $|A+B| = M \leq \frac{8}{\varepsilon}m \leq \frac{1}{\sqrt{2}}w_k(m)^{1/4}(m)$. Observe that $\frac{8}{\varepsilon}m = 8\log^{(10)}(n)m = 8\log^{(8)}(m)m$. So we want to show that

$$8\log^{(8)}(m) \leq \frac{1}{\sqrt{2}}w_k(m)^{\frac{1}{4}}.$$

However, since we already proved that $w_k(m) = (\log\log m)^{\frac{1}{2^{2^{k+9}}}} \geq \log^{(7)}(m)$, it suffices to show that

$$c(\log^{(8)}(m))^4 \leq \log^{(7)}(m), \tag{5.1}$$

where c is some constant. However, this inequality clearly holds for large enough n.

So far we have proven that if $M \leq \frac{4}{\varepsilon}(\alpha+\beta)$, then B contains a k-arithmetic progression, where $k = \log^{(6)}(m)$. In the following, we want to prove that the expected number of such arithmetic progressions in B is less than $1/O(n^2)$, and this implies that there is no k-arithmetic progression w.h.p. If there is no k-arithmetic progression, then we can conclude that when $M \leq \frac{4}{\varepsilon}(\alpha+\beta)$, there are no α queries matching at the same β positions.

We know that B contains a k-arithmetic progression. Let $B_a = \{p_1^a, p_2^a, ..., p_k^a\} \subseteq B$ be the set of positions in this k-arithmetic progression. For a fixed query q and fixed set B we have that the common difference in the arithmetic progression is d, this is depicted in figure 5.3.

The probability of seeing a fixed k-term arithmetic progression is at most $(\frac{1}{\sqrt{s}})^k$, since this requires the existence of k disjoint and balanced intervals of the form $[p_i + |q|, p_{i+1} + |q|]$. Clearly, this probability is at most $(\frac{1}{\sqrt{s}})^k$ by Lemma 5.3. The crucial observation here is that the total number of k-term arithmetic progression is very small: we have less than n values for the common difference d and less than n different starting positions, hence the number of ways to construct these k consecutive

balanced strings in a string of total length n is at most n^2. By linearity of expectation, the expected number of k-term arithmetic progressions becomes

$$\mathbb{E}[\#\text{k-term arithmetic progressions}] \leq (\frac{1}{\sqrt{s}})^k n^2.$$

Now we want to prove that this expected number is less than n^{-2}, i.e. that w.h.p there are no k-term arithmetic progressions. We know that $k = \log^{(6)} m = \log^{(8)} n$ and $s = n^{\varepsilon}$. Thus,

$$(\frac{1}{\sqrt{s}})^k n^2 = n^{-\frac{\varepsilon k}{2}+2} = n^{-\frac{\log^{(8)}(n)}{2\log^{(10)}(n)}+2} < n^{-2}$$

where the last inequality follows from the fact that $-\frac{\log^{(8)}(n)}{2\log^{(10)}(n)} < -4$ for all sufficiently large n. This concludes the case where we have few elementary intervals.

Putting it all together

We now have all the ingredients to stitch our lower bound together. The first condition of Chazelles framework 5.1 was settled in subsection 5.3, where we showed that any query $q \in \mathcal{Q}$ has an output size of at least $\Omega(\sqrt{n}/s\sqrt{\log(n)}) = \Omega(n^{0.5-o(1)})$. Furthermore, the total size of our good vectors is $\Omega\left((n/s)^2\right)$. In subsection 5.3 and 5.3 we settled the second condition, hence we showed that for any α queries $q_1, ..., q_\alpha$ the size of their intersection is less than β, i.e. $|q_1 \cap ... \cap q_\alpha| \leq \beta$, w.h.p.. This leads us to the following bound on the space

$$S(n) = \Omega\left(\frac{\sum_{q\in\mathcal{Q}}|q|}{\alpha 2^{O(\beta)}}\right) = \Omega\left(\frac{(n/s)^2}{\log\log n 2^{O(\log\log n)}}\right) = \Omega\left(n^{2-o(1)}\right).$$

where $s = n^{\varepsilon}$ and $\varepsilon = \frac{1}{\log^{(10)}(n)}$. This concludes our proof of Theorem 5.4.

5.4 The Binary Alphabet

In this section, we generalize our lower bound from the previous section to a binary alphabet. We believe this is an important contribution specially when juxtaposed against the case of existential queries where for a binary alphabet we could obtain linear-sized data structures with $O(\log n)$ query times (or even constant query times, depending on the model). In other words, this lower bound suggests that reporting queries are strictly more difficult than existential queries, at least for binary alphabets. Therefore this section is devoted to proving our main result in the binary setting.

Theorem 5.6. *For any data structure working in the pointer machine model, which tries to solve the reporting jumbled indexing problem, where $\lambda \geq 2$ and achieves query time $Q(n)+k = O(n^{\frac{1}{2}-o(1)})+k$, where k is the output size, must use $S(n) = \Omega(n^{2-o(1)})$ space.*

The big picture of our lower bound is pretty much the same. However, the case of a binary alphabet requires dealing with some additional challenges, including some noteworthy challenges that come from the area of random walks. It is easier to talk about those challenges once we have presented our construction.

The construction of our string. Our input string S will be constructed randomly and it depends on a (constant) number of events that each holds with high probability and here the high probability is taken to mean with probability at least $1 - O(1/n)$. As we have a constant number of such events, it follows that all of them hold with high probability. To unclutter the proof, we will implicitly assume that all such events hold, instead of conditioning each statement on all these high probability events holding. In the first part of the string, we have $\frac{n}{s}$ chunks of size $2s$ where each chunk contains s 0s followed by s 1s. Then, in the middle of the string we place $10n$ 1s. Finally, in the last part of the string we place a completely random 01 string of length $2n$ (see Figure 5.4). The total length of the string is $14n$.

$$S = \overbrace{0^s1^s0^s1^s0^s1^s\ldots0^s1^s}^{\text{Repeating pattern of } 0^s1^s}\overbrace{111\ldots1}^{10n}\overbrace{110100101011\ldots10}^{\text{random 01 string of length 2n}}$$

Figure 5.4: Example of an input string S

The challenges. Obviously, the source of our challenges is that we do not have an extra # character to use in our construction and this requires some non-trivial changes. Imagine if we did the obvious thing and we just removed all the extra # characters from the previous construction. Now, if we consider a window (one that crosses the middle of the string) and slide it two characters to the right, it loses a 01 sequence on the left but it gains 00 or 11 on the right, both with equal probability. As a result, the number of 0s in the window behaves exactly like a random walk: it goes up or down by one with 0.5 probability. This means, if we match at a position i, then the next matching positions are when the random walk returns to this balance. This is equivalent to the structure of the zeros of a simple random walk, i.e., the values j for which $\sum_{i=1}^{j} X_i = 0$ where each X_i is +1 or -1 with probability 0.5. This has been studied at least since 1940s and it has some very strange and unintuitive properties. For instance, its number of zeros does not have a symmetric distribution and thus it does not follow the "law of large numbers" [45, 49]. It also has a lot of "clustered" zeros and this would break a lot of our reasoning. In the previous proof, we circumvented the "clustered" zeroes problem by using the extra character and making sure that two matches happen at least distance $2s$ apart. Here, we do not have this luxury so instead we use the sequence 0^s1^s in the first half of the string. This gives us a "simple random walk with variable wind" problem! Here, as we slide the window over the 0^s part, we may lose or stay at the same number of 0s with probability 0.5 but while we slide the window over the 1^s part, we may gain or stay at the same number of 0s with probability 0.5. So informally speaking, this is kind of "a simple random walk in a

windy climate" where the wind pushes us to the left for s steps and then to the right for the next s steps and so on. Informally, the consequence of this change is that once we pass the origin, we cannot get back to the origin until the "wind" changes direction but it happens every s steps and thus there will be large gaps of of size s between a lot of the matches. It turns out, this is sufficient for a lower bound.

Because of the aforementioned challenges, the description of our set of queries is more complicated and it requires a number of additional concepts. We will do this in the following subsection.

Constructing the Set of Queries

We define the concept of a *valid substring*: a substring $x_i, \cdots, x_j$ is valid if $i \leq 2n$ and $j \geq 12n$, i.e., it starts at the first part of S, includes the middle part of S in its entirety, and then ends in the last part of S. Clearly, the number of valid substrings is exactly $(2n)^2 = 4n^2$. Similar to Lemma 5.1, we can prove the following lemma using a simple application of Chernoff bound.

Lemma 5.5. *Consider a fixed valid substring, and let x_0 and x_1 be the number of 0s and 1s in the window, respectively. Assuming $s \leq \sqrt{n}$, we have*

$$\Pr[|x_1 - 10n - x_0| > s + \sqrt{cn\log n} \geq 2\sqrt{cn\log n}] = O(n^{-c}).$$

Proof. A valid window has $10n$ 1s at its center and up to s 0s or 1s from the first part of the string. But the last part of the string is completely random and thus the lemma follows by Lemma 5.1. □

The probability in the above lemma is bounded by $\frac{1}{n^4}$ if we set $c = 4$. Furthermore, as the number of valid substrings is $4n^2$, it follows that with high probability (to be specific, with probability at least $1 - 1/n$), for every valid substring with x_0 0s and x_1 1s, we will have $|x_1 - 10n - x_0| \leq 4\sqrt{n\log n}$. As discussed at the beginning, in the rest of this proof we assume that this event holds.

Definition 5.4. *We say a Parikh vector $\psi = (\psi(0), \psi(1))$ is valid if*

(i) $\psi(1) \geq 10n$

(ii) and $|\psi(1) - 10n - \psi(0)| \leq 4\sqrt{n\log n}$.

Definition 5.5. *We say a valid Parikh vector $\psi = (\psi(0), \psi(1))$ is bad if it matches less than $\delta\sqrt{\frac{n}{\log n}}$ valid substrings of S, and good otherwise, for an appropriate parameter δ to be chosen later. Similarly, a valid substring that does not match a good Parikh vector is called bad.*

Lemma 5.6. *There exists a fixed constant δ, such that the number of bad substrings is at most n^2.*

For a full proof, we refer to the appendix. As a result, w.h.p, the majority of valid substrings will be good and thus they will be matching a good Parikh vector. Consider a valid Parikh vector ψ and assume a substring starting at position i matches ψ. If the substring is not a valid substring, then we call the match *invalid*.

Lemma 5.7. *W.h.p, the number of invalid matches of any fixed valid Parikh vector ψ is at most $s+2\log n$.*

Proof. Assume there are w substrings that match ψ that start at positions $i_1,\cdots,i_w$ and end at positions $j_1,\cdots,j_w$. Since ψ is valid, we have $\psi(1)\geq 10n$. For a match from position i_ℓ to j_ℓ to be invalid, we must either have $i_\ell > 2n$ or $j_\ell < 12n$. Reorder the matches such that $i_1 < \cdots < i_x$ are the matches that belong to the former case. It follows that $2n < i_\ell < 10n$ for every $1\leq \ell \leq x$ which implies all the characters from position i_1 to i_x are 1s. This means that the only way these matches can happen is for all characters between j_1 and j_x to be also 1s. Subsequently, this implies that all the values $i_1,\cdots,i_x$ are consecutive and subsequently all the values $j_1,\cdots,j_x$ are also consecutive. However, the values $j_1,\cdots,j_x$ are in the last part of the string which is a completely random 01 string. The probability of having x consecutive values of 1 at a fixed given position is 2^{-x}. If $x\geq 2\log n$, then the probability of observing some x consecutive 1s at some point is at most $2n\cdot 2^{-2\log n} < 1-\frac{1}{n}$, by union bound. Thus, $x\leq 2\log n$ with high probability.

Now consider the other case and once again, reorder the matches such that $i_1 < \cdots < i_y$ and $j_1 < \cdots < j_y$ denote the invalid matches of latter case. So $j_\ell < 12n$ for $1\leq \ell \leq y$. In this case, we also have $j_\ell > 2n$ which implies all the characters from positions j_1 to j_y are 1s which implies all the characters from positions i_1 to i_y are also 1s. However, these positions are in the first part of the string where the maximum number of consecutive 1s is at most s. □

As discussed earlier, we will assume that the above event holds, meaning, for each valid Parikh vector we have at most $s+2\log n$ invalid matches. Ultimately, s will be bounded by $n^{o(1)}$ which would imply the number of invalid matches is negligible compared to the number of matches of a good Parikh vector (which matches $\Omega(\sqrt{n/\log n})$ positions).

Definition 5.6. *Consider a valid match of a valid Parikh vector ψ at position i. If the character at position i is a 0 (reps. a 1) then we call the match a* 0*-match (resp.* 1*-match). Furthermore, a* 0*-match (resp.* 1*-match) at position i is an immediate match if ψ also* 0*-matches (resp.* 1*-matches) position $i-1$.*

Lemma 5.8. *Consider a fixed valid Parikh vector ψ. Let $\mathscr{I}$ be the random variable that counts the number of immediate matches of ψ and let $\mathscr{M}$ be the random variable that counts the number of valid matches of ψ, where the randomness is over the choices of our string S. Then, we have $E(\mathscr{I})\leq \frac{E(\mathscr{M})}{2}$.*

Proof. For a value $j > 12n$, let $\mathscr{M}_j$ denote a 01 random variable which is 1 if a valid match to ψ ends at position j and $\mathscr{Y}_j$ be a 01 random variable which is 1 if an immediate match to ψ ends at position j. Clearly, we have $\mathscr{M} = \sum_{j=12n}^{14n} \mathscr{M}_j$ and $\mathscr{I} = \sum_{j=12n}^{14n} \mathscr{Y}_j$. Observe that if a match to ψ ends at position j, then it should begin at position $j - |\psi| + 1$. Consider our input string $S = x_1, \cdots, x_{14n}$. Remember that only the last $2n$ characters of S actually vary and the first $12n$ characters of S are fixed. Furthermore, for each x_i, $12n+1 \leq i \leq 14n$, x_i is 0 with probability $\frac{1}{2}$ and 1 with the same probability. Define the random variable X_j, $12n+1 \leq j \leq 14n$, where X_j is 1 if $x_j = x_{j-|\psi|}$ and 0 otherwise. Observe that in the definition of X_j, the right hand side is not a random variable but the left hand size is (this is because by assumption, j lies in the last part of S whereas $j - |\psi|$ must lie in the first part of S). Thus, $Y_{j+1} = \mathscr{M}_j X_{j+1}$, which in plain English means that for position $j+1$ to be the end of an immediate match, first, we must have a match ending at position j, and the character x_{j+1} must be equal to the character $x_{j-|\psi|+1}$. The crucial observation here is that $\mathscr{M}_j$ and X_{j+1} are independent random variables since $\mathscr{M}_j$ only depends on the string $x_1, \cdots, x_j$ and x_{j+1} is sampled independently. Thus,

$$\mathbb{E}(\mathscr{I}) = \mathbb{E}(\sum_{j=12n}^{14n} \mathscr{Y}_j) = \sum_{j=12n}^{14n} \mathbb{E}(\mathscr{Y}_j) = \sum_{j=12n}^{14n} \mathbb{E}(\mathscr{M}_{j-1} X_j) = \sum_{j=12n}^{14n} \mathbb{E}(\mathscr{M}_{j-1})\mathbb{E}(X_j) \leq \sum_{j=12n}^{14n} \frac{\mathbb{E}(\mathscr{M}_{j-1})}{2} \leq \frac{\mathbb{E}(\mathscr{M})}{2}.$$

□

Remember that definitions of $\mathscr{I}$ and $\mathscr{M}$ depend on the Parikh vector ψ (we have chosen to omit this dependency from our notation to reduce the clutter). If $\mathscr{I} \leq \frac{3}{4}\mathscr{M}$, then we call ψ *suitable*. By Lemma 5.8 and using Markov inequality, we get $\Pr[\mathscr{I} \geq \frac{3}{4}\mathscr{M}] \leq \frac{2}{3}$ which means a valid good Parikh vector is suitable with probability at least $\frac{1}{3}$.

Definition 5.7. *The set of queries $\mathscr{Q}$ is the set of all the valid, good, and suitable Parikh vectors.*

Lemma 5.9. *We have $\mathbb{E}(\sum_{q \in \mathscr{Q}} |q|) \geq n^2$.*

Proof. Each query $q \in \mathscr{Q}$ is good and thus by definition it matches $\delta\sqrt{\frac{n}{\log n}}$ positions. We have $4n^2$ valid substring and by Lemma 5.5 and definition of valid Parikh vectors, each valid substring matches some valid Parikh vector but By Lemma 5.6, at most n^2 of them match the bad Parikh vectors, meaning, the number of substrings that match good, and valid Parikh vectors is at least $3n^2$. By Lemma 5.8, a valid and good Parikh vector will be suitable with probability at least $\frac{1}{3}$ and thus it will be included in our query set. Thus, by linearity of expectation we get that $\mathbb{E}(\sum_{q \in \mathscr{Q}} |q|) \geq n^2$. □

We now focus on the second condition of the lower bound framework. Here, we need to show that with high probability, we cannot have α queries $\psi_1, \cdots, \psi_\alpha$ that all

match the same β positions $p_1, \cdots, p_\beta$, for appropriate choice of α and β. Here, we need additional ideas to generalize our lower bound to the binary alphabet. Basically, we would like to ignore the immediate matches as they could potentially break the second requirement of our framework.

Consider a fixed suitable Parikh vector ψ and assume it matches k positions in S. Among the k matches, at most $s+2\log n$ of them could be to invalid substrings by Lemma 5.7. Since ψ is suitable, it follows that the number of immediate matches of ψ is at most $2k/3$. Thus, at least $k/3$ of the matches are non-immediate and thus at least $k/3-s-2\log n$ of them will be both non-immediate and valid. Furthermore, at least half of these will be b-matches for some $b \in \{0,1\}$. But since ψ is good, we have that $k \geq \delta\sqrt{\frac{n}{\log n}}$. Assuming $s = n^{o(1)}$, it follows that $k/6 - s/2 - \log n > k/10$ and thus there are at least $k/10$ valid, non-immediate b-matches. We call these the *interesting* matches of ψ and we focus only on these output. Thus, if a suitable Parikh vector ψ has k matches, then it has at least $k/10$ interesting matches. Now consider the data structure: it has $Q(n)+\gamma k$ time to report all the k matches of the query. If we focus on reporting only the interesting matches, the data structure has still the same amount of time but we have $Q(n)+\gamma k \leq Q(n)+10\gamma k'$ which implies we can solely focus on the interesting matches by increasing the constant γ by a constant factor. For a query $q_i \in Q$, let q_i' be the subset of q_i consisting of all the interesting matches.

Lemma 5.10. *Consider a suitable Parikh vector ψ and assume ψ matches S at positions i_1 and i_2 where both of these are interesting matches. Then we have $|i_1 - i_2| \geq s$.*

Proof. W.l.o.g, assume $i_1 < i_2$ and assume both of them are 0-matches. By our assumptions, ψ matches the substring from position i_1 to $j_1 = i_1 + |\psi| - 1$. Thus, $x_{i_1} = 0$. After the match at i_1, there will be $t \geq 0$ additional immediate matches, i.e., as far as $x_{i_1} = x_{i_1+2} = \cdots = x_{i_1+t-1} = 0$ and $x_{j_1+1} = x_{j_1+2} = \cdots = x_{j_1+t} = 0$ (see Figure 5.5). However, we will either have $x_{i_1+t} = 1$ or $x_{j_1+t+1} = 1$. In either case, there won't be another 0-match until we slide the window over the current chunk of 0s, and over the next chunk of s 1s, meaning, we will have $i_2 - i_1 \geq s$. □

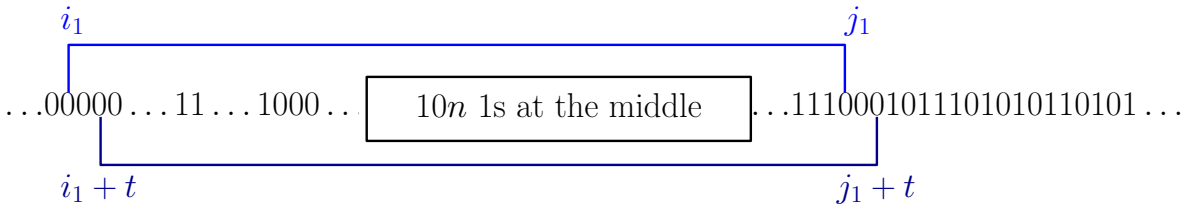

Figure 5.5: The gap between two interesting matches is at least s.

In the previous proof, we set α and β equal and set both of them to $m = \log\log n$. We also defined a notion of elementary intervals and considered two cases. Here, not

only do we have to make different choices for α and β, but we also need to consider an additional case.

To satisfy the second condition of our lower bound framework, we need to consider α queries $q'_1, \cdots, q'_\alpha$ that match β positions $p_1, \cdots, p_\beta$. As before, assume we have $|q'_1| < \cdots < |q'_\alpha|$ and $p_1 < \cdots < p_\beta$. We define the sets $A = \{|q'_1|, \cdots, |q'_\alpha|\}$ and $B = \{p_1, \cdots, p_\beta\}$, and the resulting set $D = A + B$ defines our set of elementary intervals. Similarly, define b_i to be the balance of the string (i.e., the difference between 0s and 1s) in the substring starting from index $p_1 + |q'_1|$ till the index $p_1 + |q'_i|$. We call b_i the *the initial balance*. We have a very similar lemma to Lemma 5.4, full proof is in the appendix.

Lemma 5.11. *Given $p_1, ..., p_\beta$, $|q'_1|, |q'_2|, ..., |q'_\alpha|$ and $b_2, b_3, ..., b_\alpha$ it is possible to exactly calculate the balance in all elementary intervals from $p_i + |q_k|$ to $p_j + |q_l|$.*

Here, we set $\alpha = s^5 m$, $\beta = m$ and $m = \log\log n$. Consider the elements of A modulo s. Observe that it is possible to pick a subset $A' \subset A$ such that all the elements of A' are congruent modulo s^5 and $|A'| = |A|/s^5 = m$. Define $D' = A' + B$. Next, we define the sets $A_s = \{\lceil \frac{a}{s} \rceil | a \in A'\}$ and $B_s = \{\lceil \frac{b}{s} \rceil | b \in B\}$. Observe that for every $a_1, a_2 \in A'$ we have $|a_1 - a_2| \geq s^5$ and also for every two $b_1, b_2 \in B$ we have $|b_1 - b_2| \geq s$, by Lemma 5.10 and thus $|A_s| = |A'|$ and $|B_s| = |B|$. Define the set $D_s = A_s + B_s$. The set D_s defines the set of *truncated elementary intervals*.

Lots of Truncated Elementary Intervals

In this case, we assume $|D_s| \geq \frac{24}{\varepsilon} m$ where ε is such that $s = n^\varepsilon$. We calculate the probability that two fixed sets A' and B describe aforementioned set of queries and matching positions with this condition. Let $I = \{b_2, \cdots, b_{|A'|}\}$ be the set of initial balances that correspond to the queries represented by A'. As A', B, and I are fixed, by Lemma 5.11, we know the balance of every elementary interval. Let D_3 be a subset of D_s such that for every two elements $d_1, d_2 \in D_3$ we have $|d_1 - d_2| \geq 3$. We can pick D_3 to contain at least $|D_s|/3 \geq \frac{8}{\varepsilon}|A'|$ elements. Consider two consecutive integers $\ell_1, \ell_2 \in D_3$, $\ell_1 < \ell_2$. By definition of D_3 we know that $\ell_1 + 3 \leq \ell_2$ and by definition of D, we know there exists values $a_1, a_2 \in A'$, $b_1, b_2 \in B$ such that $\ell_1 = \lceil \frac{a_1}{s} \rceil + \lceil \frac{b_1}{s} \rceil$ and $\ell_2 = \lceil \frac{a_2}{s} \rceil + \lceil \frac{b_2}{s} \rceil$. Observe that these equalities imply that

$$a_1 + b_1 \leq s\left(\lceil \frac{a_1}{s} \rceil + \lceil \frac{b_1}{s} \rceil\right) = s\ell_1 \qquad \text{and} \qquad s\ell_2 - 2s = s\left(\lceil \frac{a_2}{s} \rceil + \lceil \frac{b_2}{s} \rceil - 2\right) \leq a_2 + b_2$$

and thus $a_2 + b_2 \geq a_1 + b_1 + s$. In other words, every two consecutive integers in the set D_3, imply the existence of an interval (not necessarily elemental) of length at least s. However, the balance of this interval is fixed. As the interval has length at least s, this probability of finding this fixed balance in S is at most $\frac{1}{\sqrt{s}}$ by Lemma 5.3. Thus, the probability that the sets A', B and I describe $|A'|$ queries that match at $|B|$ positions is at most

$$\left(\frac{1}{\sqrt{s}}\right)^{|D_3|}.$$

As before, we do a union bound and the number of choices for sets A', B and I is at most $n^{|A'|+|B|+|I|} \leq n^{3m}$. Thus, the probability that the set A', B and I describe a set of queries and matching positions is at most

$$n^{3m}\left(\frac{1}{\sqrt{s}}\right)^{|D_3|} \leq n^{3m}\left(\frac{1}{\sqrt{s}}\right)^{\frac{8}{\varepsilon}m} = n^{3m-4m} < n^{-2}$$

where the last inequality assume that $m > 2$ which holds for large enough n.

Few Truncated Elementary Intervals

In this case, we have $|D_s| < \frac{24}{\varepsilon}|A'|$. We can verify that choosing $k = \log^{(6)} n$ and $\varepsilon = \frac{1}{\log^{(10)} n}$ will make sure that $\frac{24}{\varepsilon} \leq \frac{1}{\sqrt{2}} w_k(m)^{1/4}$; this is the same calculation that we have done in Page 53 and it will result in Equation 5.1 in Page 53 but with a different constant for c. We can now use Theorem 5.3 on the sets A' and B: we can gleam that B_s contain arithmetic progressions of length k. Let d_b be the common difference of the k-term progression in B_s. Relabel the values in B_s such that $b_1 < \cdots < b_k$ represents the values of the k-term arithmetic progression in B_s. We consider two additional cases.

Large d_b

This case corresponds to when $d_b \geq s^3$. Now fix the values $b_1, \cdots, b_k$. Recall that p_i is the value in B such that $b_i = \lceil \frac{p_i}{s} \rceil$ but note that fixing b_i does not determine the value of p_i.

Now, consider one query $q'_1 \in A'$ and let ψ be the corresponding Parikh vector. We now fix ψ as well. Recall that q'_1 is the set of interesting matches of ψ. By the definition of an interesting match, all interesting matches are either 0-matches or 1-matches. W.l.o.g., assume q'_1 consists of only 0-matches. Thus, the (fixed) value of $b_i = \lceil \frac{p_i}{s} \rceil$ (uniquely) identifies a block 0^s that contains a match! Now define the following indices: $\ell_i = sb_i$ and $\ell'_i = sb_i + s - 1$ and $r_i = \ell_i + |\psi|$ and $r'_i = \ell'_i + |\psi|$. By what we have argued, $\ell_i \leq p_i \leq \ell'_i$ and that all the characters in our string S between indices ℓ_i and ℓ'_i are 0. If a match to the query ψ starts at some position p_i, $\ell_i \leq p_i \leq \ell'_i$, then the match ends at the position $p_i + |\psi| - 1$, between indices r_i and r'_i. However, observe that the balance of the substring between positions p_i and p_{i+1} is between $-s$ and s. This in turn implies that the balance of the substring between positions $p_i + |\psi|$ and $p_{i+1} + |\psi| - 1$ is also bounded by the same amount. However, these positions are s indices away from r_i and r_{i+1} respectively. This implies that the balance of the substring between indices r_i and r_{i+1} is between $-3s$ and $3s$ and the probability of this event is at most

$$\frac{6s}{\sqrt{r_{i+1} - r_i}}. \tag{5.2}$$

Crucially, note that the fixed sequence $b_1, \cdots, b_k$ and the value of $|\psi|$ uniquely determines the indices $r_1, \cdots, r_k$ and that by our assumptions, $b_{i+1} \geq b_i + s^3$ which in turn

implies $r_{i+1} \geq r_i + s^4$ and thus Equation 5.2 is upper bounded by

$$\frac{6s}{s^2} = \frac{6}{s}.$$

Thus, for fixed values of $b_1, \cdots, b_k$ and $|\psi|$, the probability that we get a match at positions $p_1, \cdots, p_k$ is at most $\left(\frac{6}{s}\right)^k$. We can now do the union bound again. We have at most n choices for b_1 and at most n choices for the common difference and n choices for $|\psi|$. Thus, the probability that α queries match at β positions in this case is bounded by

$$n^3 \left(\frac{6}{s}\right)^k = n^3 6^{\log^{(6)} n} n^{-\frac{\log^{(6)} n}{\log^{(10)} n}} < n^{-2}$$

for large enough n.

Small d_b

This is the last case and here we have $d_b < s^3$. It turns out this is the easy case. Consider the arithmetic progression $b_1, \cdots, b_k$ and let $p_1, \cdots, p_k$ be the corresponding matching positions.

Thus, observe that $b_k \leq b_1 + d_b k \leq b_1 + k s^3 < b_1 + \frac{s^4}{2}$. The last inequality follows since $s = n^\varepsilon$ and $k = \log^{(10)} n$. As a result, it follows that $p_k \leq p_1 + \frac{s^5}{2}$. Let $B' = \{p_1, \cdots, p_k\}$. Now consider the elements of A': these are the elements that are congruent modulo s^5. Thus, the smallest difference between any two values in A' is at least s^5 which is larger than the largest distance between values in B'. This implies that $|A' + B'| = |A'||B'| = mk$. Furthermore, it follows that the difference between the consecutive elements of $A' + B'$ is at least s since the elements of B' are at least distance s apart. We can now go back to our argument for when we had a lot of elementary intervals. And in fact our argument here is very much similar to the one presented in Subsection 5.3. We have at most $n^{|A'|+|B'|} \leq n^{2m}$ choices for picking the sets A' and B'. Together with at most $m-1$ initial balance values, we have at most n^{3m} different choices for fixing the elementary intervals. Each elementary interval now has size at least s and thus the probability of finding a fixed balance at these fixed elementary intervals is at most $(\frac{1}{\sqrt{s}})^{|A'||B'|} = (\frac{1}{\sqrt{s}})^{mk}$. Thus, by union bound, the total probability is bounded by $n^{3m}(\frac{1}{\sqrt{s}})^{mk} < n^{-2}$ for large enough n.

This settles the second condition in 5.1 for the binary alphabet and thereby concludes our proof of Theorem 5.6 which settles our lower bound for the binary case.

5.5 Acknowledgements

The authors would like to thank Moshe Lewenstein, Seth Pettie, and Tsvi Kopelowitz for various useful discussions regarding the jumbled indexing problem.

5.6 Appendix

Lemma 5.1. *For a fixed valid substring, the probability that the substring has balance larger than $\sqrt{n}/\delta$ is less than $2n^{-c}$, for $\delta = \frac{1}{\sqrt{c\log n}}$.*

Proof. Let x_i be the random variable taking value 0 or 1 if the i'th character to the right of the middle is a 0 or a 1 respectively. Define $X = \sum x_i$, now in order for the valid substring to be imbalanced by more than $\sqrt{n}/\varepsilon$, it must be the case that $X \geq \sqrt{n}/\varepsilon + n'/2$ or $X \leq \sqrt{n}/\varepsilon + n'/2$. If we look at the first of these cases (and notice that they are symmetric) we get the probability

$$\Pr[X \geq \frac{\sqrt{n}}{\varepsilon} + \frac{n'}{2}] \leq \Pr[X \geq \frac{\sqrt{n'}}{\varepsilon} + \frac{n'}{2}]$$

where the inequality comes from the fact that $n' \leq n$ so the right event will always happen if the left event happened. We can now apply the Chernoff bound

$$\begin{aligned}\Pr[X \geq \frac{\sqrt{n'}}{\varepsilon} + \frac{n'}{2}] &= \Pr[X \geq (1 + \frac{2\varepsilon}{\sqrt{n'}})\frac{n'}{2}] \\ &\leq e^{-\frac{\frac{2}{\varepsilon\sqrt{n'}}\frac{n'}{2}}{2+\frac{2}{\varepsilon\sqrt{n}}}}.\end{aligned}$$

If we clean the expression for constants we get

$$\begin{aligned}e^{-\frac{\frac{2}{\varepsilon\sqrt{n'}}\frac{n'}{2}}{2+\frac{2}{\varepsilon\sqrt{n}}}} &\leq e^{-\frac{n'}{\varepsilon^2 n}} \\ &\leq e^{-\frac{1}{\varepsilon^2}}\end{aligned}$$

since $n' \leq n$. If we insert $\varepsilon = \frac{1}{\sqrt{c\log n}}$ we get

$$e^{-\frac{1}{\varepsilon^2}} = e^{-c\log n} = n^{-c}.$$

For the second case we notice that the argument is symmetric and therefore the total probability of the underlying substring being more imbalanced than $\sqrt{n}/\varepsilon$ becomes $2n^{-c}$. □

Lemma 5.3. *The probability of having i 0s in a random binary string of length m is at most $\Theta(1/\sqrt{m})$.*

Proof. We start of by looking at the case when $i = m/2$, i.e., the probability of having a balanced string in a random binary string of length m. Since m is even we substitute $m = 2k$ in order to make the calculations more readable. In a $2k$ character long binary string there are 4^k different ways of constructing the binary string. Out of these $\binom{2k}{k}$ gives us a balanced string, since we need the same amount of zeroes and ones in the

string, in order for it to be balanced. Therefore the probability of getting a balanced string in our random $m = 2k$ binary string becomes

$$\binom{2k}{k}\frac{1}{4^k} = (\frac{(2k)!}{k!k!})\frac{1}{4^k},$$

by applying Stirling's approximation we get

$$\begin{aligned}
(\frac{(2k)!}{k!k!})\frac{1}{4^k} &= \Theta(\sqrt{4\pi k}(2k)^{2k}e^{-2k}\frac{1}{(\sqrt{2\pi k}k^k e^{-k})^2}\frac{1}{4^k}) \\
&= \Theta(\sqrt{4\pi k}4^k k^{2k}e^{-2k}\frac{1}{2\pi k k^{2k}e^{-2k}}\frac{1}{4^k}) \\
&= \Theta(\frac{\sqrt{4\pi k}}{2\pi k}) \\
&= \Theta(\frac{1}{\sqrt{\pi k}}).
\end{aligned}$$

Which shows that the probability of getting a balanced string in a random string of $2k = m$ binary characters is $\Theta(1/\sqrt{n})$. This proves the statement, since the binomial probability distribution is centred around its mean, hence no other value of i has a greater probability than the mean, i.e., when $i = m/2$. □

Lemma 5.4. *Given $p_1, ..., p_\beta$, $|q_1|, |q_2|, ..., |q_\alpha|$ and $b_2, b_3, ..., b_\alpha$ it is possible to exactly calculate the balance of all the elementary intervals.*

Proof. We will prove the Lemma using induction in the endpoints.

Our base case is trivial since we are given $b_2, ..., b_\alpha$ and from the fact that the interval from $p_1 + |q_1|$ to $p_2 + |q_1|$ is balanced.

This leads to the following induction hypothesis. For every elementary interval $p_i + |q_k|$ to $p_j + |q_l|$ we look at, we know the balances of all elementary intervals to the left of $p_i + |q_k|$.

Given an interval from $p_i + |q_k|$ to $p_j + |q_l|$ we want to calculate the balance of the string. The balance in the interval from $p_{j-1} + |q_l|$ to $p_j + |q_l|$ is 0. Furthermore we can exactly calculate the balance b in the interval from $p_{j-1} + |q_l|$ to $p_i + |q_k|$, since we have a chain of elementary intervals (all intervals in between $p_{j-1} + |q_l|$ and $p_i + |q_k|$) per our induction hypothesis, in which b becomes equal to all of these added up (see figure 5.6 for illustration). The balance from $p_i + |q_k|$ to $p_j + |q_l|$ then becomes $-b$.

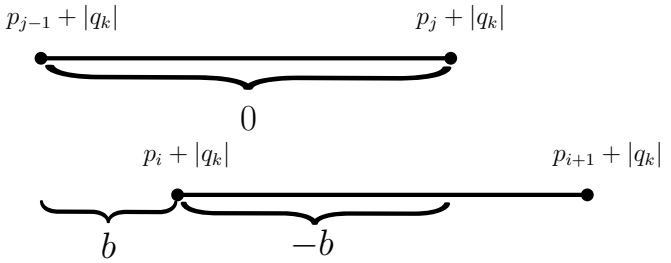

Figure 5.6: Illustration of Lemma 5.4.

□

Lemma 5.6. *There exists a fixed constant δ, such that the number of bad substrings is at most n^2.*

Proof. First we count the total number of valid Parikh vectors ψ. To do that, we simply need to count the number of choices for $\psi(1)$ and $\psi(0)$. We have $3n$ choices for $\psi(1)$ since it is a number between $10n$ and $13n$. After choosing $\psi(1)$ we have at most $8\sqrt{n\log n}$ choices for $\psi(0)$. Thus, the total number of valid Parikh vectors is at most $13 * 8n\sqrt{n\log n}$.

A bad Parikh vector by definition can only match $\delta\sqrt{\frac{n}{\log n}}$ valid substrings, w.h.p. Thus, in total, the number of bad substrings is bounded by

$$\delta\sqrt{\frac{n}{\log n}} \cdot 13 * 8n\sqrt{n\log n} < n^2$$

for an appropriate choice of δ. □

Lemma 5.11. *Given $p_1, ..., p_\beta$, $|q'_1|, |q'_2|, ..., |q'_\alpha|$ and $b_2, b_3, ..., b_\alpha$ it is possible to exactly calculate the balance in all elementary intervals from $p_i + |q_k|$ to $p_j + |q_l|$.*

Proof. The proof is very similar and we only need one extra observation: that the left half of the string is fixed and thus the balance in the interval $[p_i, p_{i+1}]$ is a fixed function of p_i and p_{i+1}. Remember that in Lemma 5.4, the balance of the string between p_i and p_{i+1} was exactly zero whereas here it is not. Nonetheless, this does not change the main argument as we can show that we can compute balance in all the elementary interval as functions of $p_1, ..., p_\beta$, $|q_1|, |q_2|, ..., |q_\alpha|$ and $b_2, b_3, ..., b_\alpha$. □

Chapter 6

Stabbing and ranges in moderate dimensions

Rectangle Stabbing and Orthogonal Range Reporting Lower Bounds in Moderate Dimensions

Peyman Afshani * Rasmus Killmann †

July 29, 2022

Abstract

We study the orthogonal range reporting and rectangle stabbing problems in moderate dimensions, i.e., when the dimension is $c\log(n)$ for some constant c. In orthogonal range reporting, the input is a set of n points in d dimensions, and the goal is to store these n points in a data structure such that given a query rectangle, we can report all the input points contained in the rectangle. The rectangle stabbing problem is the "dual" problem where the input is a set of rectangles, and the query is a point.

Our main result is the following: assume using $S(n)$ space, we can solve either problem in $d = c\log n$ dimensions, $c \geq 4$, using $Q(n) + O(t)$ time in the pointer machine model of computation where t is the output size. Then, we show that if the query time is small, that is, $Q(n) = n^{1-\gamma}$, for $\gamma \geq \frac{2}{2+\log c}$, then the space must be $\Omega\left(n^{1-\gamma} n^{\sqrt{c\gamma}/e - o(\sqrt{c\gamma})}\right)$. Interestingly, we obtain this lower bound using a non-constructive method, and we show the existence of some codes that generalize a specific aspect of error correction codes. Our result overcomes the shortcomings of the previous lower bounds in the pointer machine model for non-constant dimension [12, 14, 16, 42], as the previous results could not be extended for $d = \Omega(\sqrt{\log n})$.

The only known lower bounds for rectangle stabbing, when the dimension is non-constant, are based on conditional lower bounds upon the best-known results on CNF-SAT [90]. Therefore, our lower bound is the first non-trivial unconditional lower bound for orthogonal range reporting and rectangle stabbing with non-constant dimension.

*Aarhus University. Supported by DFF (Det Frie Forskningsråd) of Danish Council for Independent Research under grant ID DFF−7014−00404.

†Aarhus University. Supported by DFF (Det Frie Forskningsråd) of Danish Council for Independent Research under grant ID DFF−7014−00404.

6.1 Introduction

The focus of this paper is to study two fundamental problems within the area of *range searching*. The first problem is *orthogonal range reporting* which is the problem of storing a set of n points in $\mathbb{R}^d$ in a data structure, such that given a query hyperrectangle[1] in d-dimensions the t points contained in the hyperrectangle can be reported efficiently. The second problem of *rectangle stabbing* is the "dual" in the sense that the input set and the query is swapped.

Both problems are fundamental and have attracted significant attention [17, 18]. Using standard reductions, it is possible to reduce orthogonal range searching in d dimensions to rectangle stabbing in $2d$ dimensions and vice versa[2]. While in low dimensions, the constant factor increase in dimension is very impactful, the focus of this paper is when the dimension is large, $d = c\log n$, for a parameter c, and thus, the two problems essentially become equivalent.

We look into proving unconditional space and query lower bounds for both problems, although in the somewhat restricted pointer machine model [85]. Along the way, we find an interesting connection to *error correction* codes, which is a particular set of strings/codes which can be utilized to communicate over an insecure line of communication.

Previous results

There are plenty of known results for both problems when the dimension is low, while when the dimension grows beyond a constant, the problems are less characterized.

Rectangle stabbing In the one-dimensional case, rectangle stabbing, also called interval stabbing, can be solved using a diverse range of methods [47]. The optimal solutions use linear space and have a query time of $O(\log n + t)$. In the two-dimensional case, Chazelle [39] presented an optimal data structure using linear space and $O(\log n + t)$ query time. Almost optimal results have also been obtained for $d = 3$, after a long series of papers [16, 38, 78, 79]. Using range trees, the two-dimensional and three-dimensional results can be generalized to higher dimensions by paying a $\log(n)$ factor in both space and query time for each dimension added. In the pointer machine model, it is also known that using $O(n\log^{O(1)} n)$ space, the query time must be $\Omega(\log n(\log n/\log\log n)^{d-2} + t)$ [16].

Orthogonal range reporting For $d = 1$, a simple binary search tree answers range queries in $O(\log n + t)$ time, while using linear space. For $d = 2$, the problem can be solved with the same query time but using $O(n\log n/\log\log n)$ space [39], which is optimal in the pointer machine model [42]. Much later, the optimal result was obtained for three dimensions in the pointer machine model [14]. Similar to the

[1] By hyperrectangle, we mean the Cartesian product of d intervals.

[2] Both problems can be reduced to dominance reporting in 2d dimensions.

rectangle stabbing problem, the results can be generalized to a higher dimension by paying a $\log n$ factor in both space and query time for each dimension added. In the pointer machine model, it is also known that answering queries in $O(\log^{O(1)} n + t)$ time requires $\Omega(n(\log n/\log\log n)^{d-1})$ space [42].

These approaches are unsuitable for the case where $d = c\log(n)$, since the dependency on the dimension penalizes this construction too much, often referred to in the community as the "*curse of dimensionality*". The best known upper bound, devised by Chan [36], when the dimension is $c\log(n)$, is a data structure which achieves $Q(n) = O(n^{1-1/c\log(c)})$ expected query time and space $S(n) = n^{1+\delta}$ for any $\delta > 0$.

Previous lower bounds Currently, the available lower bounds are conditional lower bounds, using the best known results on CNF-SAT [90], which gives preprocessing/query time lower bounds. Furthermore, obtaining unconditional lower bounds in unrestricted computational models, like the RAM, is completely hopeless. Fortunately, suppose we restrict the memory model of the data structure to operate in the pointer machine model. In that case, one can obtain reasonable lower bounds that are tight for a wide range of problems. In fact, for the orthogonal range reporting problem and the rectangle stabbing problem, there are a number of lower bounds available [14, 16, 42]. However, the dependency of these lower bounds on the dimension is not very good, and in fact, it is not explicitly mentioned by the lower bounds. There is one very related instance where we are aware of the explicit dependency on the dimension. Afshani and Driemel [12] study multilevel structures, motivated by answering Fréchet queries. However, by a careful look at the details of the lower bounds, one can see that the hidden constant is at least $d^{\Omega(d)}$, meaning the lower bounds become useless as d approaches $\log n/\log\log n$. They study the Cartesian product of t 2D "simplex stabbing" problems, and they show that for any data structure with $S(n)$ space and $Q(n)$ query time we must have

$$S(n) = \Omega((\frac{n}{Q(n)})^2)\frac{\left(\frac{\log(n/Q(n))}{\log\log n}\right)^{t-1}}{2^{O(2^t)}} \quad \text{and}$$

$$S(n) = \Omega\left(\frac{n}{Q(n)}\right)^2 \Theta\left(\frac{\log(n/Q(n))}{t^{3+o(1)}\log\log n}\right)^{t-1-o(t)}.$$

It has been shown that this problem can be modelled using the Cartesian product of t two-dimensional problems and furthermore it contains a subproblem involving an instance of the t-dimensional orthogonal range reporting problem. However, as it can be seen in the lower bounds, the dependency on t is such that the lower bounds become trivial as soon as t gets close to $(\log n)^{1/3}$.

Our Results

Our main result states that any data structure operating in the pointer machine model, which solves the rectangle stabbing problem (or the orthogonal range reporting problem), when the dimension is $c\log(n)$ for $c \geq 4$ and uses query time $Q(n) = n^{1-\gamma}$ for

a parameter $\gamma \geq \frac{2}{2+\log c}$, must use space: $\Omega\left(n^{1-\gamma} n^{\sqrt{c\gamma}/e - o(\sqrt{c\gamma})}\right)$. As far as we know, this is the first non-conditional lower bound for the two problems when the dimension is non-constant.

We obtain our results using an interesting extension of error correction codes. We show that given a set of binary strings S of size $O((\frac{d}{r})^{rx})$, where r is the number of 1's in the strings and d is the length of each string, then for any h of these strings $s_1, \ldots, s_h \in S$ the string $s_1 \vee \cdots \vee s_h$ has more than M 1's, where "the error parameter" M can be chosen to be $\min\{\frac{d^y r^{1-y}}{e}, \frac{1}{2}rhz\}$, where $x, y, z > 0$ are parameters which satisfy $x + y + z = 1$. In particular this gives a set of strings in which for any h strings they are fairly different. Notice that for $h = 2$ this is precisely the property sought after when constructing error correction codes.

Our technical contribution We believe our main technical contribution is a simple non-constructive idea for showing the existence of "difficult input instances". Previously, two main techniques were used to build "difficult input instances" for range searching lower bounds. The first one is a deterministic and constructive approach that builds an explicit set of points, and it was first employed by Chazelle [42] for the 2D orthogonal range reporting problem. Since then, it has been generalized to higher dimensions [13, 16]. Unfortunately, this approach suffers exponentially by the dimension, and thus it cannot be applied in dimensions above $\log n / \log\log n$. The second approach is a randomized approach that uses the classical "alteration" technique (basically a 'sample and refine' strategy) [21]. This was also used in Chazelle's original paper [42] and since then, it has been the dominant strategy. Unfortunately, applying this technique in a way that does not degrade exponentially on the dimension is challenging. It leads to lengthy calculations, making it difficult to know the optimal choices of the parameters involved. Here instead, we introduce a new and more straightforward *non-constructive* approach. We show that if the structures we are looking for do not exist, we can upper bound a certain count, and then we reach a contradiction by showing that the upper bound is smaller than the count, meaning the desired structure should exist.

6.2 Technical Preliminaries

In this section, we present some of the technicalities and preliminaries needed to prove our lower bound. We will explain the underlying model of computation, the pointer machine model, and an existing lower bound framework in this model. Finally, we will introduce the notion of error correction codes.

Model of Computation and Framework

The Pointer Machine Model [85] is an underlying model of computation which models data structures that only navigate through pointers to access the underlying memory locations, e.g., any tree-based data structure. The model has proven to be

efficient in proving lower bounds for data structure problems. Consider an abstract "reporting" problem where the input is a set $\mathscr{U}$ and each query q reports a (implicitly defined) subset $q_{\mathscr{U}}$, of $\mathscr{U}$. The underlying data structure is a directed graph $\mathscr{G}$, where each node has out-degree two and represents exactly one memory cell; furthermore, there is a special root node $r(\mathscr{G})$. Each node can store exactly one element from the universe $\mathscr{U}$ and edges between nodes represent pointers between memory cells. Any additional information can be stored and accessed for free by the data structure, and the data structure has unlimited computational resources. Given a query q, the data structure must start at the root node $r(\mathscr{G})$ and explore a connected subgraph $\mathscr{G}' \subseteq \mathscr{G}$, such that each element of the query $q_{\mathscr{U}}$ is contained in at least one node of the subgraph $\mathscr{G}'$. The size of the subgraph $\mathscr{G}'$ gives a lower bound on the query time and the size of the entire graph $\mathscr{G}$ gives a lower bound on the space required to solve the reporting problem.

One of the first lower bound frameworks in the pointer machine model was given by Bernard Chazelle [42, 44]. In this paper we will use a slightly different framework developed by Afshani [10], which states the following.

Theorem 6.1. *Assume we have a data structure for a geometric stabbing problem that uses at most $S(n)$ space and answers queries in $Q(n)+O(k)$ time in which n is the input size, and k is the output size. Assume for this problem we can construct an input set, inside the unit cube in $\mathbb{R}^d$, of n ranges such that*

1. *Every point of the unit cube is contained in t ranges in which $t \geq Q(n)$.*
2. *The volume of the intersection of every α ranges is at most v, for $\alpha < t$.*

Then, $S(n) = \Omega(tv^{-1}/2^{O(\alpha)}) = \Omega(Q(n)v^{-1}/2^{O(\alpha)})$.

We will also utilize error correction codes from the field of information theory to satisfy the second condition of the framework. In fact, it turns out that what we need is an extension of error correction codes with additional properties.

Error Detection and Correction Codes

Error-correcting codes are an old technique studied within the field of information theory and were originally studied by Richard Hamming in 1947 [86]. Hamming developed the famous Hamming code, which first appeared in Claude Shannon's *A Mathematical Theory of Communication*[82]. The Hamming code is a family of linear error-correcting codes that can be used for both error detection and error correction. The underlying problem of error detection and error correction are the following.

Alice and Bob want to transmit a string s from a fixed set of S binary strings. They communicate across an insecure line, and hence a number of bits may be flipped. In error detection Alice sends a string $s \in S$, and Bob must determine whether the received string s' is equal to the original string s sent by Alice. In error correction, Alice sends a string $s \in S$, but now Bob must recover the original string s from the received string s'. Note that any error correction scheme can also be used as an error

detection scheme, but not the other way around. In section 6.3 we look into error correction and give, in our opinion, a natural and interesting extension of the problem.

Bounding the Binomial Coefficient

Our lower bound is proven by using a counting argument. At times we need to bound the binomial coefficient to arrive at our desired result. The following simple Lemma is known, but for completeness, we present a proof in the appendix.

Lemma 6.1. *For positive integers n and k where* $1 \leq k \leq n$*, it holds that*

$$\left(\frac{n}{k}\right)^k \leq \binom{n}{k} \leq \left(\frac{en}{k}\right)^k.$$

6.3 The Lower Bound Construction

In this section, we present the details of our lower bound construction. We start this section with a discussion of error correction codes, and a subproblem related to them. We start by defining what an interesting string is, which is both used in the error correction Section 6.3 and when concluding the lower bound in Section 6.3.

Definition 6.1. *A binary string is interesting if it has length d and has exactly r* 1*'s, for some fixed parameters r and d.*

The parameters r and d will be fixed in our final lower bound.

Theorem 6.2. *Given r, d and h, such that* $r < \frac{d}{4}$*, it holds that for any* $N \leq (\frac{d}{r})^{rx}/(2e)$*, there exists a set S of N interesting strings such that for any h strings* $s_1, \ldots, s_h \in S$ *the string* $s_1 \vee \cdots \vee s_h$ *has more than M 1's, where M can be chosen to be* $\min\{\frac{d^y r^{1-y}}{e}, \frac{1}{2}rhz\}$*, for parameters x, y, and z such that* $x+y+z=1$ *and* $x,y,z>0$*.*

The $\vee$ is the logical OR operation. The error parameter M can be thought of as a measure of how distinct the h strings can be, and therefore it translates into how many flipped bits the construction can withstand and still recover. We will discuss this below.

The above Theorem helps us prove our lower bound on rectangle stabbing when the dimension is $c\log(n)$. This paper's main result is formulated in the following Theorem and will be proven at the end of this section.

Theorem 6.3. *Consider a pointer machine data structure that solves the rectangle stabbing problem in* $c\log(n)$ *dimensions, for some parameter* $c \geq 4$ *and achieves query time* $Q(n) = o(n)$*. Let* γ *be such that* $Q(n) = n^{1-\gamma}$*. If* $\gamma \geq \frac{2}{2+\log c}$*, then the data structure must use* $\Omega\left(n^{1-\gamma} n^{\sqrt{c\gamma}/e - o(\sqrt{c\gamma})}\right)$ *space.*

And as we mentioned earlier since we are in non-constant dimension this result also gives us a lower bound for orthogonal range reporting.

Corollary 6.1. *Consider a pointer machine data structure that solves the orthogonal range reporting problem in $c\log(n)$ dimensions, for some parameter $c \geq 8$ and achieves query time $Q(n) = o(n)$. Let γ be such that $Q(n) = n^{1-\gamma}$. If $\gamma \geq \frac{2}{1+\log c}$, then the data structure must use $\Omega\left(n^{1-\gamma} n^{\sqrt{\frac{c}{2}}\gamma/e - o(\sqrt{c\gamma})}\right)$.*

Codes with Large Spread

In this section, we consider a subproblem that we believe is an interesting generalization of error-correcting codes. Error-correcting codes deal with two parties, Alice and Bob, who have to communicate over an insecure line, meaning that a number of bits may be flipped in any message that they send to each other. Here, we only consider binary messages. To reduce the impact of errors, they agree to only communicate messages from a fixed set S. If Alice sends a message $s \in S$ we want Bob to be able to retrieve the correct s from the string s' he received. This is typically expressed as every two messages $s_1, s_2 \in S$ having Hamming distance at least $2\delta + 1$, for some parameter δ. Now, if at most δ bits change in the message that Alice sends to Bob, Bob can correctly identify Alice's message in S. Observe that if both s_1 and s_2 have r 1s, then the fact that their Hamming distance is at least $2\delta + 1$ is equivalent to the fact that $s_1 \vee s_2$ has at least $r + 2\delta + 1$ 1s. Thus, in Theorem 6.2, if we set $h = 2$, we get a set of error correcting codes, i.e., a set of codes whose pairwise Hamming distance is at least $M - r$.

The initial and immediate approach to prove Theorem 6.2 would be to do random sampling: consider all the strings with r 1s and pick a random sample of them with probability p; then delete the strings that result in some h strings to have few 1s in their logical OR, then optimize for the value of p that gives the most strings. Unfortunately, this natural approach seems difficult to analyse, due to very complex binomial coefficients involved. Instead, we come up with an alternative approach that we believe is simple but non-constructive.

Our non-constructive proof is a counting argument which fortunately turns out to be easy to analyse. It involves two different ways of counting the different number of ways to construct the set S. To gain a second way of counting the ways to construct S we need to assume the negated Statement of Theorem 6.2, if we obtain a contradiction by assuming this negated Statement, then the Theorem is proven. We start the proof by writing down the negated Statement.

Statement 6.1. *Given r, d and h, then for all sets of N interesting strings there exists a subset $s_1, s_2, \ldots, s_h$ such that $s_1 \vee s_2 \vee \cdots \vee s_h$ contains less than M 1's.*

The idea is to reach a contradiction by a counting argument. The total number of ways to pick N interesting strings is $\binom{\binom{d}{r}}{N}$. From Statement 6.1 we can bound this count in a different way. Assuming Statement 6.1 every set S of N interesting strings has a subset of h strings $s_1, \ldots, s_h$ such that $s_1 \vee \cdots \vee s_h$ has less than M 1's. We now describe choosing S in a different way. To do that, first we choose the strings $s_1, \ldots, s_h$ and then the remaining strings of S. To choose $s_1, \ldots, s_h$, first we choose the string

$$
\begin{array}{ll}
 & \overbrace{}^{M} \\
\text{String } 1 & 00\cdots 00\,100111\cdots 11\,00\cdots 00 \\
 & \overbrace{}^{M} \\
\text{String } 2 & 00\cdots 00\,110010\cdots 10\,00\cdots 00 \\
 & \vdots \quad\quad \vdots \quad\quad \vdots \quad\quad \vdots \\
 & \overbrace{}^{M} \\
\text{String } h & 00\cdots 00\,010111\cdots 00\,00\cdots 00
\end{array}
$$

Figure 6.1: h strings which have their r 1's concentrated around the same M positions.

$s_0 = s_1 \vee ... \vee s_h$. Then we choose $s_1, ..., s_h$ as a "subset" of s_0. To choose s_0, we need to pick M positions that contain 1s. The number of ways to pick these M positions is $\binom{d}{M}$. The number of ways to pick h interesting strings among these M positions is $\binom{\binom{M}{r}}{h}$. The ways to pick the remainder of the N interesting strings after having picked these h strings becomes $\binom{\binom{d}{r}}{N-h}$. If we combine all of this, we get an expression that overcounts the number of ways to pick N interesting strings, which means that we would expect this count to be bigger than the exact count. Therefore we would expect the following inequality to hold

$$\binom{\binom{d}{r}}{N} < \binom{d}{M}\binom{\binom{M}{r}}{h}\binom{\binom{d}{r}}{N-h}.$$

Thus, if this inequality does not hold, then it follows that Statement 6.1 is incorrect which in turn prove Theorem 2.

Arriving at a Contradiction

This section is dedicated to finalizing our proof of Theorem 6.2. The way we do this, as mentioned above, is to arrive at a contradiction of Statement 6.1. To do that, we will try to arrive at the opposite inequality, i.e., we will try to satisfy the opposite inequality. The first step is to utilize the following Lemma to restructure our expression; we refer to the appendix for the proof.

Lemma 6.2. $\frac{\binom{A}{B}}{\binom{A}{B-C}} \geq (\frac{A}{2B})^C$, *if* $B > 2$ *and* $A > 2B$.

We can apply Lemma 6.2 to our expression by setting $A = \binom{d}{r}$, $B = N$ and $C = h$. This gives a new expression we need to satisfy in order to arrive at our contradiction,

$$\left(\frac{\binom{d}{r}}{2N}\right)^h > \binom{d}{M}\binom{\binom{M}{r}}{h}.$$

Now we need to apply Lemma 6.1. The idea is to create a new Statement, where we increase the right side and/or decrease the left side. If we can satisfy this new inequality, our original inequality will also be satisfied. We apply both the lower bound on the left side since we want to decrease this expression and the upper bound on the right side since we want to increase that expression.

$$\left(\frac{\left(\frac{d}{r}\right)^r}{2N}\right)^h > \left(\frac{de}{M}\right)^M \left(\frac{e\left(\frac{eM}{r}\right)^r}{h}\right)^h \Leftarrow$$
$$\left(\frac{d}{r}\right)^r > \left(\frac{de}{M}\right)^{\frac{M}{h}} e\left(\frac{eM}{r}\right)^r 2N.$$

Where the implication follows from dividing by h in the exponent and removing the h in the denominator on the right side. We are going to proceed by splitting this expression into three pieces. By ensuring that these three inequalities are satisfied

1. $\left(\frac{d}{r}\right)^{rx} \geq 2eN$
2. $\left(\frac{d}{r}\right)^{ry} > \left(\frac{eM}{r}\right)^r$
3. $\left(\frac{d}{r}\right)^{rz} > \left(\frac{de}{M}\right)^{\frac{M}{h}}$

and as long as $x+y+z=1$ and $x,y,z>0$, we arrive at the desired contradiction. The first inequality directly gives us the upper bound on the size of N namely that $\left(\frac{d}{r}\right)^{rx}/(2e) \geq N$. For the second inequality, we get an upper bound on the value of M

$$\left(\frac{d}{r}\right)^{ry} > \left(\frac{eM}{r}\right)^r \Leftrightarrow \left(\frac{d}{r}\right)^{y} > \left(\frac{eM}{r}\right) \Leftrightarrow$$
$$\frac{d^y r^{1-y}}{e} > M.$$

The third inequality gives us a bound on M and a relation between r and d:

$$\left(\frac{d}{r}\right)^{rz} > \left(\frac{de}{M}\right)^{M/h} \Leftrightarrow rhz\log\left(\frac{d}{r}\right) > M\log\left(\frac{de}{M}\right)$$

Note that we have $M \geq r$ and thus, we can increase the RHS by replacing M with r,

$$rhz\log\left(\frac{d}{r}\right) > M\log\left(\frac{de}{M}\right) \Leftarrow$$
$$rhz\log\left(\frac{d}{r}\right) > M\log\left(\frac{de}{r}\right) \Leftarrow$$
$$rhz\log\left(\frac{d}{r}\right) > M\left(\log\left(\frac{d}{r}\right)+2\right) \Leftarrow \frac{1}{2}rhz > M.$$

Therefore, since we have two bounds on M we can choose M to be the minimum of $(\frac{d^y r^{1-y}}{e}, \frac{1}{2}rhz)$ as long as $\left(\frac{d}{r}\right)^{rx}/(2e) > N$, $r < \frac{d}{4}$, $x+y+z=1$ and $x,y,z>0$. When the parameters satisfy these conditions we arrive at a contradiction of Statement 6.1, which then proves Theorem 6.2.

Utilizing the Framework

This section is dedicated to proving a lower bound on rectangle stabbing when the dimension is $c\log(n)$ for some constant $c \geq 4$. The idea is to utilize the framework by Afshani, Theorem 6.1 and use Theorem 6.2 from the previous section to satisfy the second condition of Afshani's framework.

Assume we have a data structure that uses $S(n)$ space and has a query time of $Q(n)+O(k)$. To use the framework, we need to build a set of ranges. Each range will be an axis-aligned rectangle in d dimensions. Note that while the framework requires $t \geq Q(n)$, it is also sufficient to have $t \geq Q(n)/(2e)$. To see this, let c_0 be the constant hidden in the O-notation in the query time, meaning, we assume the query time is at most $Q(n)+c_0k$. Observe that as long as for the output size k we have $k \geq Q(n)/2e$, then $Q(n)+c_0k \leq Q(n)/2e+(5+c_0)k$. And thus, this only changes the constant factors involved in the lower bound. Thus, in the rest of the proof, we assume that we create $t \geq Q(n)/(2e)$ different shapes, and we will use each shape to cover the unit cube in $d = c\log(n)$ dimensions. For each shape, we tile the unit cube with $\frac{n}{Q(n)}$ copies. If we do that, this ensures that we have n rectangles in total. We build our shapes in the following way: for each shape, we have r side lengths of length $1/2$ and $d-r$ side lengths of unit length. This implies that each rectangle of any shape has volume $\frac{Q(n)}{n} = \frac{1}{2^r}$.

Observation 6.1. *Our shapes satisfies the first condition of Theorem 6.1.*

Every point of the unit cube has been covered by at least $Q(n)/(2e)$ different shapes, and thus the answer to any query point will have a size of at least $Q(n)/(2e)$ and as described above, this is sufficient to satisfy the first condition.

Now we are only left with making sure that these shapes can satisfy the second condition of the framework, namely that the volume of the intersection of every α shapes is at most v.

We say that a rectangle R is *represented* by a binary string s of length d if the following holds: for every i, $1 \leq i \leq d$, if s contains a 1 at position i, then R has side-length $\frac{1}{2}$ along dimension i but otherwise, R has side-length 1.

Observation 6.2. *Every rectangle represented by an interesting string has volume* $\frac{1}{2^r}$.

The next Observation reveals why we had to study this particular type of error correction codes in section 6.3.

Observation 6.3. *Let T be a set of interesting strings given by Theorem 6.2, for some parameters r,h which is to be decided later. Let $R_1,\cdots,R_h$ be h rectangles represented by h distinct strings $s_1,\cdots,s_h \in T$, then the volume of the intersection of $R_1,\cdots,R_h$ is at most 2^{-M}.*

Proof. Simply observe that $R_1 \cap \cdots \cap R_h$ will have a side-length of at most $1/2$ at dimension i if any of the strings has a 1 at position i. The Lemma follows from Theorem 2 i.e., that the logical OR of the strings has at least M 1s. □

By the above observation, it thus follows that the goal is to pick $Q(n)$ interesting strings, such that for any α strings in the set we have that $s_1 \vee ... \vee s_\alpha$ have more than β 1's. This will ensure that the corresponding shapes, for any α shapes, the volume of their intersection will be less than $v = \frac{1}{2^\beta}$. We utilize Theorem 6.2, since we have strings of length d with exactly r 1's, h becomes α and M becomes our β.

We know that $d = c\log(n)$. Let γ be such that $Q(n) = n^{1-\gamma}$. Observe that since $Q(n)/n = (1/2)^r$, we get that $r = \gamma\log(n)$. Now we are only left with making sure that these parameters fit Theorem 6.2. To do that, we must ensure that $N = Q(n)/(2e)$ satisfy:

$$\begin{aligned}
\frac{Q(n)}{2e} &\leq \left(\frac{d}{r}\right)^{rx}/(2e) \quad \Leftrightarrow \\
n^{1-\gamma} &\leq \left(\frac{c}{\gamma}\right)^{rx} \quad \Leftrightarrow \\
\log(n)(1-\gamma) &\leq \gamma\log(n)x\log\left(\frac{c}{\gamma}\right) \quad \Leftrightarrow \\
(1-\gamma)\frac{1}{x} &\leq \gamma\log\left(\frac{c}{\gamma}\right) \quad \Leftrightarrow \\
(1-\gamma)\frac{1}{x} &\leq \gamma\log(c) - \gamma\log(\gamma) \Leftarrow \\
(1-\gamma)\frac{1}{x} &\leq \gamma\log(c).
\end{aligned}$$

Where the last step follows since $\gamma \leq 1$ and thus $-\gamma\log(\gamma)$ is non-negative, and therefore, we can make the RHS smaller by removing the $-\gamma\log(\gamma)$ term. Then we just end up with $(1-\gamma)\frac{1}{x} \leq \gamma\log(c)$ and so we get that $\gamma \geq \frac{1}{x}/(\frac{1}{x} + \log(c))$. We also need that $c \geq 1$; otherwise, γ will not be between 0 and 1. Note that $\gamma \geq \frac{1}{x}/(\frac{1}{x} + \log(c))$ implies that $Q(n) \leq n^{1-\frac{1/x}{1/x+\log(c)}}$.

Now we are left with finding the possible value of $M = \beta$. Recall that by Theorem 6.2, we have:

$$\beta \leq \min\{\frac{d^y r^{1-y}}{e}, \frac{1}{2}rhz\} = \min\{\frac{(c\log(n))^y(\gamma\log(n))^{1-y}}{e}, \frac{1}{2}\gamma\log(n)\alpha z\}.$$

We pick α to balance the two terms in the min function. This yields $\alpha = \frac{2}{ez}\frac{c^y}{\gamma^y}$ and $\beta = \frac{1}{2}z\alpha r$. We are now ready to plug these values into our lower bound framework. By assumptions, $Q(n) = o(n)$ but observe that $Q(n) = n^{1-\gamma} = \frac{n}{2^{\gamma\log n}}$ which means $n^{1-\gamma} = o(n)$ and thus $r = \gamma\log n = \omega(1)$. As a result, we can afford to pick $z = \sqrt{\frac{1}{r}} = o(1)$ to make sure that $\beta = \Theta(\alpha z r) = \omega(\alpha)$. We pick $x = y = \frac{1}{2}$ which implies $\beta = \frac{\log n\sqrt{c\gamma}}{e}$. Thus, if $Q(n) = n^{1-\gamma}$ where $\gamma \geq \frac{2}{2+\log c}$, we obtain the space lower bound

$$S(n) = \Omega\left(\frac{Q(n)2^\beta}{2^{O(\alpha)}}\right) = \Omega\left(n^{1-\gamma}2^{\beta-o(\beta)}\right) = \Omega\left(n^{1-\gamma}n^{\sqrt{c\gamma}/e-o(\sqrt{c\gamma})}\right).$$

As a result, when c grows, the space bound becomes polynomially larger than n. In particular, for $c = 4$, when $Q(n) = O(n^{1-\frac{2}{2+\log 4}}) = O(\sqrt{n})$, a simple calculation yields that the claimed space lower bound is $\Omega(n^{1+\delta})$ for some constant δ. This concludes our lower bound construction and proves Theorem 6.3.

6.4 Conclusions

This paper considered the orthogonal range reporting problem and the rectangle stabbing problem in moderate dimensions. Both are classical problems in the field of data structures and computational geometry. We presented the first unconditional lower bound (in the pointer machine model) that works when the dimension is beyond logarithmic. We believe our work leads to two interesting problems. First, can we improve the query lower bound? There are still some room for improvement since the best known algorithm has query time $Q(n) = n^{1-\frac{1}{c}}$ and space $S(n) = n^{1+\varepsilon}$ for some $\varepsilon < 1$. Improving either the upper bound or the lower bound for these problems are both very interesting open problems.

Second, can we obtain similar improvements in other problems where we have instances of high-dimensional orthogonal range reporting? Two important problems are multi-level data structures, e.g., range trees and range searching with respect to the Fréchet distance [12].

6.5 Appendix

Lemma 6.1. *For positive integers n and k where* $1 \leq k \leq n$*, it holds that*

$$\left(\frac{n}{k}\right)^k \leq \binom{n}{k} \leq \left(\frac{en}{k}\right)^k.$$

Proof. We start by proving the lower bound. We first observe that for $k = 1$, the bound trivially holds

$$\left(\frac{n}{k}\right)^k = n = \binom{n}{k}.$$

Hence look at the case where $k > 1$ and let $0 < i < k \leq n$.

$$k \leq n \Leftrightarrow \frac{i}{n} \leq \frac{i}{k} \Leftrightarrow 1 - \frac{i}{k} \leq 1 - \frac{i}{n} \Leftrightarrow \frac{k-i}{k} \leq \frac{n-i}{n} \Leftrightarrow \frac{n-i}{k-i},$$

and from this we obtain

$$\left(\frac{n}{k}\right)^k = \frac{n}{k} \cdot \cdots \cdot \frac{n}{k} \leq \frac{n}{k} \cdot \cdots \cdot \frac{n-k+1}{1} = \binom{n}{k},$$

which gives us the desired lower bound. For the upper bound we see that

$$\binom{n}{k} = \frac{n!}{(n-k)!k!} = \frac{n\cdot(n-1)\cdot\dots\cdot(n-k+1)}{k!} \leq \frac{n^k}{k!}.$$

Since

$$e^k = \sum_{j=0}^{\infty} \frac{k^j}{j!}$$

we can look at the term where $j = k$ and get

$$e^k \geq \frac{k^k}{k!} \Leftrightarrow \frac{1}{k!} \leq \left(\frac{e}{k}\right)^k \Leftrightarrow \frac{n^k}{k!} \leq \left(\frac{en}{k}\right)^k,$$

which gives us the upper bound. □

Lemma 6.2. $\frac{\binom{A}{B}}{\binom{A}{B-C}} \geq (\frac{A}{2B})^C$, *if* $B > 2$ *and* $A > 2B$.

Proof. The proofs goes through, using some simple calculations and bounds,

$$\begin{aligned}\frac{\binom{A}{B}}{\binom{A}{B-C}} &= \frac{\frac{A!}{B!(A-B)!}}{\frac{A!}{(B-C)!(A-B+C)!}} = \frac{(B-C)!(A-B+C)!}{B!(A-B)!} \\ &= \frac{(A-B+1)\dots(A-B+C)}{(B-C+1)\dots B}.\end{aligned}$$

and we are going to make this expression smaller by taking the smallest term in the numerator and raising it to the number of terms in the numerator, i.e. C and we take the biggest term in the denominator and do the same, raise it to the power C we get

$$\begin{aligned}\frac{(A-B+1)\dots(A-B+C)}{(B-C+1)\dots B} &> \frac{(A-B+1)^C}{B^C} \\ &> \left(\frac{A}{B}-1\right)^C\end{aligned}$$

and lastly since we assumed that $A > 2B$ we conclude the proof

$$\left(\frac{A}{B}-1\right)^C > \left(\frac{A}{2B}\right)^C$$

□

Chapter 7

Hierarchical Categories in Colored Range Searching

Hierarchical Categories in Colored Range Searching

Peyman Afshani* Rasmus Killmann† Kasper Green Larsen‡

July 29, 2022

Abstract

In colored range counting (CRC), the input is a set of points where each point is assigned a "color" (or a "category") and the goal is to store them in a data strucuture such that the number of distinct categories inside a given query range can be counted efficiently. CRC has strong motivations as it allows data structure to deal with categorical data.

However, colors (i.e., the categories) in the CRC problem do not have any internal structure, whereas this is not the case for many datasets in practice where hierarchical categories exists or where a single input belongs to multiple categories. Motivated by these, we consider variants of the problem where such structures can be represented. We define two variants of the problem called hierarchical range counting (HCC) and sub-category colored range counting (SCRC) and consider hierarchical structures that can either be a DAG or a tree. We show that the two problems on some special trees are in fact equivalent to other well-known problems in the literature. Based on these, we also give efficient data structures when the underlying hierarchy can be represented as a tree. We show a conditional lower bound for the general case when the existing hierarchy can be any DAG, through reductions from the orthogonal vectors problem.

7.1 Introduction

Range searching is a broad area of computational geometry where the goal is to store a given set P of input points in a data structure such that one can efficiently report (*range reporting*) or count (*range counting*) the points inside a geometric query region $\mathscr{R}$. Sometimes, each point $p_i \in P$ is associated with a weight $w_i \in \mathbb{R}$ and the goal could be to find the sum of the weights in $\mathscr{R}$, or the maximum weight in $P \cap \mathscr{R}$ (*range max* problem). This is a very broad area of research and there are many well-studied variants. See a recent excellent survey by Agarwal for more information [17].

*Aarhus University. peyman@cs.au.dk.
†Aarhus University. killmann@cs.au.dk.
‡Aarhus University. larsen@cs.au.dk.

Colored (or *categorical*) range searching is an important variant where each input point is associated with a *category* which conceptually is represented as a color; the goal of the query is then to report or count the number of distinct colors inside the query range. Colored range searching has strong motivations since colors allow us to represent *nominal attributes* such as brand, manufacturer and so on and in practice a data set often contains a mix of nominal and ordinal attributes.

Colored range searching was introduced in 1993 by Janardan and Lopez [60] and it has received considerable attention since then. However, classical colored range searching only models a "flat" categorical structure where categories have no inherent structure. We consider variants of colored range counting to model such structures. We show that looking at colored range searching from this angle creates a number of interesting questions with non-trivial connections to other already existing problems.

Problem Definitions and Motivations

We begin by formally defining colored range counting.

Problem 7.1 (colored range counting)**.** *Given an input set P of n points in $\mathbb{R}^d$, a set C of colors (i.e., categories) and a function $\mathfrak{C} : P \to C$, store them in a data structure such that given a query range $\mathscr{R} \subset \mathbb{R}^d$, it can count the number of distinct colors in $\mathscr{R}$, i.e., the value $|C_{\mathscr{R}}|$ where $C_{\mathscr{R}} = \{\mathfrak{C}(p) \mid p \in P \cap \mathscr{R}\}$.*

In the weighted version, input also includes a weight function $\mathscr{W} : C \to \mathbb{R}$ and the output of the query is the weighted sum of the distinct colors in $\mathscr{R}$, i.e., the value $\sum_{c \in C_{\mathscr{R}}} \mathscr{W}(c)$.

Colored range searching assumes that the colors are completely unstructured and that each point receives exactly one color. However, these assumptions can be inadequate as hierarchical categories are quite common. For example, biological classification of living organisms are done through a tree where inclusion in a category implies inclusion in all the ancestor categories. In fact, similar phenomena happen with respect to most notions of classification (e.g., classification of industries). In other scenarios, a point may have multiple categories (e.g., a car can have a "brand", a "color", "fuel type" and so on). While it is possible to view the set of categories assigned to a point as one single category, doing so ignores a lot of structure. For example, "a blue diesel car" is both a "blue car" and also a "diesel car" but by considering "blue diesel" as a single category, this information is lost. We believe it is worthwhile to study the notion of structures within categories from a theoretical point of view. We are not in fact the first in trying to do so and we will shortly discuss some of the previous attempts.

A natural way to represent hierarchical categories is to assume that vertices of a DAG $\mathscr{G}$ represent the set of categories where an edge $\vec{e} = (u, v)$ from (a category) u to (a category) v means that u is a sub-category of v. We call $\mathscr{G}$ a *category DAG* (or a *category tree* if it is a tree). For a vertex $v \in \mathscr{G}$, we define $\mathscr{G}_{\leq}(v)$ as the subset of vertices of $\mathscr{G}$ that can reach v (i.e., "sub-categories" of v), $\mathscr{G}_{\geq}(v)$ as the subset of

vertices of $\mathscr{G}$ that v can reach (i.e., "super-categories" of v). Similarly, for a subset $H \subset V(G)$ we define $\mathscr{G}_{\geq}(H) = \cup_{v \in H} \mathscr{G}_{\geq}(v)$, and $\mathscr{G}_{\leq}(H) = \cup_{v \in H} \mathscr{G}_{\leq}(v)$.

Category trees allows us to represent hierarchical categories. Category DAGs allow us to capture cases where points can have multiple categories. Consider the car example again. We can define a category DAG $\mathscr{G}$ where a vertex $u \in \mathscr{G}$ represents the compound category $\{\text{diesel}, \text{blue}\}$ with edges to vertices d and b that represent "diesel" and "blue" categories respectively. Thus, the set $\mathscr{G}_{\leq}(d)$ represents all the "diesel" cars and it includes the category u, the "blue diesel" category, and similarly, the set $\mathscr{G}_{\leq}(b)$ represents all the "blue" cars which also includes the category u, the "blue diesel" category. Likewise, $\mathscr{G}_{\geq}(u)$ includes both b and d since a "blue diesel" car is both a "blue car" and a "diesel car".

We revisit colored range searching problems, using concepts of category DAGs or trees.

Problem 7.2 (sub-category range counting (SCRC))**.** *Consider an input point set $P \subset \mathbb{R}^d$ of n points, a DAG $\mathscr{G}$ with $O(n)$ edges, and a function $\mathfrak{C} : P \to \mathscr{G}$. The goal is to store the input in a data structure, such that given a query that consists of a query range $\mathscr{R} \subset \mathbb{R}^d$ and a query vertex $v_q \in \mathscr{G}$ it can output $|C_{\mathscr{R}} \cap \mathscr{G}_{\leq}(v)|$ where $C_{\mathscr{R}} = |\{\mathfrak{C}(p) \mid p \in P \cap \mathscr{R}\}|$.*

Problem 7.3 (hierarchical color counting (HCC))**.** *Consider an input point set $P \subset \mathbb{R}^d$ of n points, a DAG $\mathscr{G}$ with $O(n)$ edges, and a function $\mathfrak{C} : P \to \mathscr{G}$. The goal is to store the input in a data structure, such that given a query range $\mathscr{R} \subset \mathbb{R}^d$ one can output $|G_{\mathscr{R}}|$ where $G_{\mathscr{R}}$ is the set of colors in $\mathscr{R}$, i.e., $G_{\mathscr{R}} = \bigcup_{p \in \mathscr{R} \cap P} \mathscr{G}_{\geq}(\mathfrak{C}(p))$.*

In the weighted version of the problem, each vertex v (i.e., category) of $\mathscr{G}$ is associated with a weight $w(v)$ and given the query $\mathscr{R}$, the goal is to compute $\sum_{v \in G_{\mathscr{R}}} w(v)$ instead.

Related problems. Very recently, there have been other attempts to consider the structure of "colors" within the computational geometry community, e.g., He and Kazi [58] consider a problem very similar to SCRC on a tree $\mathscr{G}$; the only difference is that instead of $|C_{\mathscr{R}} \cap \mathscr{G}_{\leq}(v)|$, the query outputs $|C_{\mathscr{R}} \cap \pi(v,w)|$ where $\pi(v,w)$ is a path between two query vertices $v, w \in \mathscr{G}$. There are also other variants available. See [58] for further references.

Previous and Other Related Results

The study of colored range counting and its variants began in 1993 [60] and since then it has received considerable attention. See the survey on colored range searching and its variants [56]. The problem has at least three main variants: color range counting (CRC), color range reporting, and "type 2" color range counting (for every distinct color, count how many times it appears). Here, we only review colored range counting results.

In 1D, one can solve the CRC problem using $O(n)$ space and $O(\log n)$ query time by an elegant and simple transformation [55] which turns the problem into the

unweighted 2D orthogonal range counting problem which can be solved within said bounds [40]. Interestingly, it is also possible to show an equivalence between the two problems [69]. The problem, however, is difficult for 2D and beyond. Kaplan et al. [62] showed that answering m queries on a set of n points requires $\Omega(n^{\omega/2-o(1)})$ time where ω is the boolean matrix multiplication exponent. Under some assumptions (e.g., the boolean matrix multiplication conjectures), this shows that $P(n)+mQ(n) \geq n^{3/2-o(1)}$ where $P(\cdot)$ and $Q(\cdot)$ are the preprocessing time and the query time of any data structure that solves the 2D CRC problem.

Note that the equivalence between 1D CRC and 2D range counting also applies to the weighted case of both problems, however, the status of the weighted 2D range counting is still unresolved. It can be solved with $O(n\log n/\log\log n)$ space and $O(\log n/\log\log n)$ query time [59] but it is not known if both space and query time can be improved simultaneously (it is possible to improve one at the expense of the other). The only available lower bound is a query time lower bound of $\Omega(\log n/\log\log n)$ [75].

Some other interesting problems related to our results are defined below.

Definition 7.1. *The following problems are defined for an input that consists of a set $P \subset \mathbb{R}^d$ of n points. The goal is to build a data structure to answer the following queries.*

- *(orthogonal range counting) Given a query axis-aligned rectangle $\mathscr{R}$, count the number of points in $\mathscr{R}$. In the weighted version, the points have weights and the goal is to compute the sum of the weights in the query.*

- *(dominance range counting) This is a special case of orthogonal range counting where the query rectangle has the form $(-\infty,q_1]\times\cdots(-\infty,q_d]$ which is also known as a dominance range. Orthogonal range counting and dominance range counting are known to be equivalent if subtraction of weights are allowed (e.g., integer weights).*

- *(3-sided color counting). The input is in the plane ($d=2$) and each point is assigned a color from a set C and the query is a 3-sided range in the form of $\mathscr{R}=[q_\ell,q_r]\times(-\infty,q_t]$ and the goal is to count the number of colors in $\mathscr{R}$. In the weighted version, the colors have weights and the goal is to compute the sum of the weights of the colors.*

- *(3-sided distinct coordinate counting) This is a special case of 3-sided color counting where an input point (x_i,y_i) has color y_i; in other words, given the query $\mathscr{R}=[q_\ell,q_r]\times(-\infty,q_t]$, we would like to count the number of distinct Y-coordinates inside it.*

- *(range max) Given a weight function $\mathscr{W}: P\to\mathbb{R}$ as part of the input, at the query time we would like to find the maximum weight inside a given query range $\mathscr{R}$.*

- *(sum-max color counting) This a combination of range max and color counting queries. Assume the points in P have been assigned colors from a set C and assume we have a weight function $\mathscr{W} : P \to \mathbb{R}$ on the points. Given a query range $\mathscr{R}$, we would like to compute the output $\sum_{c \in C} X_c(\mathscr{R})$ where $X_c(\mathscr{R})$ is the maximum weight of a point with color c inside $\mathscr{R}$; if no point of color c exists in $\mathscr{R}$, then $X_c(\mathscr{R}) = 0$.*

  A sum-max color counting query includes a number of the above problems as its special case: If all points have the same weight, then it reduces to a color counting query. If all points have the same color, then it reduces to a range max query. If all points have distinct colors, then it reduces to a weighted range counting query.

Our Results

Clearly, sub-category range counting (SCRC) is at least as hard as CRC. We also observe that hierarchical color counting (HCC) is also at least as hard. Thus, getting efficient results for $d \geq 2$ seems hopeless. Consequently, we focus on the 1D problem but for two different important DAG's: when $\mathscr{G}$ is a tree and also when $\mathscr{G}$ is an arbitrary (sparse) DAG. Our main results are the following.

For the SCRC problem, first, we observe that the following problems are equivalent:

1. SCRC when $\mathscr{G}$ is a single path on a one-dimensional input P.
2. 3-sided distinct coordinate counting (for a planar point set P).
3. 3-sided color counting (for a planar point set P).
4. 3D dominance color counting (for a 3D point set P).
5. 3D dominance counting (for a 3D point set P).

We start by observing that (1) and (2) are equivalent. It is also clear that (2) reduces to (3); the reduction from (3) to (4) is standard by mapping a 2D input point (x_i, y_i) to the 3D point $(-x_i, y_i, x_i)$ and the 3-sided query range $\mathscr{R} = [q_\ell, q_r] \times (-\infty, q_t]$ to the 3D dominance range $(-\infty, -q_\ell] \times (-\infty, q_t] \times (-\infty, q_x]$. The reduction from (4) to (5) was shown by Saladi [81]. We complete the loop by observing that (5) in turn reduces to (2). Note that the weighted versions of the problems are also equivalent by following the same reductions. Next, we show that SCRC can be solved using $O(n \log^2 n / \log\log n)$ space and with query time of $O(\log n / \log\log n)$ on any category tree $\mathscr{G}$; our query time is optimal which follows from the above reductions and using known results [75].

For the HCC problem on trees, we show that (weighted) HCC in 1D can be solved using a 2D (weighted) range counting data structure on $O(n \log n)$ points. For example, this yields a solution with $O(n \log^2 n / \log\log n)$ space and with $O(\log n / \log\log n)$ query time. Interestingly, we show that the following problems are in fact equivalent:

- Unweighted HCC in 1D when $\mathcal{G}$ is a (generalized) caterpillar graph.
- Weighted 2D range counting with $\Theta(\log n)$ bit long integer weights.
- 1D Colored range sum-max.

These reductions are quite non-trivial and they show a surprising equivalence between an unweighted problem (HCC) and the weighted 2D range counting. This allows us to directly apply known lower bounds or barriers for 2D range counting. First, there is an $\Omega(\log n/\log\log n)$ lower bound [75] for 2D range counting with near-linear space ($O(n\log^{O(1)} n)$ space) and by the above reductions, the same bound also applies to unweighted HCC in 1D.

When $\mathcal{G}$ can be any arbitrary sparse DAG, the problems become more complicated. By a reduction from the orthogonal vectors problem, we show that we must either have $\Omega(n^{2-o(1)})$ preprocessing time or the query time must be almost linear $\Omega(n^{1-o(1)})$ and this holds for both SCRC and HCC. Surprisingly, for the HCC problem, we can build a data structure that has $O(\log n)$ query time using $\tilde{O}(n^{3/2})$ space, even though the data structure requires $\tilde{O}(n^2)$ preprocessing time. This is one of the rare instances where there is a polynomial gap between the space complexity and the preprocessing time of a data structure.

7.2 Technical Preliminaries

A fundamental technique to decompose trees into paths with certain properties is called the *heavy path decomposition*. The technique was originally used as part of the amortized analysis of the link/cut trees introduced by Sleator and Tarjan [83] and later used in the data structure construction for lowest common ancestor by Harel and Tarjan [57]. The decomposition is simple and gives us the following properties.

Theorem 7.1. *Given a tree T of size $O(n)$, we can partition the (vertices of the) tree into a set of paths $\pi_1, \pi_2, \ldots \pi_h$ such that on any root to leaf path in T, the number of different paths encountered is $O(\log(n))$.*

We study the HCC and SCRC problem in one dimension. First, we consider when $\mathcal{G}$ is a tree in Section 7.3 and then we consider the general case where $\mathcal{G}$ can be any (sparse) DAG in Section 7.4. The general case is more difficult to solve and we will show that through a reduction (a "conditional lower bound"). Our reduction relies on the Orthogonal Vectors conjecture which is implied by the Strong Exponential Time Hypothesis (SETH) [90].

Hypothesis 7.1 (Orthogonal vectors conjecture)**.** *Given two sets A and B each containing n boolean vectors of dimension $d = \log^{O(1)}(n)$, deciding whether there exists two orthogonal vectors $a_i \in A$ and $b_j \in B$ cannot be done faster than $n^{2-o(1)}$ time.*

Assuming this conjecture, many near optimal time lower bounds have been proven within P, including Edit Distance, Longest Common Subsequence and Fréchet distance [24, 32, 48].

Finally, we say that a binary tree T is a generalized caterpillar tree if all the degree three vertices lie on the same path (a caterpillar tree is typically defined as the legs having size 1).

7.3 Hierarchical Color Counting on Trees

In the appendix (Section 7.6), we observe that HCC is at least as hard as CRC, using a simple reduction. As a result, we focus on the 1D case. We start off by presenting a data structure to solve 1D HCC on a tree $\mathscr{G}$ and then show that unweighted HCC on generalized caterpillars is actually equivalent to weighted 2D dominance counting (up to constant factors). This allows us to conclude that HCC on generalized caterpillar graphs cannot be solved with $o(n\log n/\log\log n)$ space and $o(\log n/\log\log n)$ query time, unless the state-of-the-art on weighted 2D dominance counting can be improved.

A Data Structure

We will now focus on the 1D HCC problem on trees, as described in Problem 7.3. Our main result is the following.

Theorem 7.2. *The HCC problem (both weighted or unweighted) in $\mathbb{R}$ can be solved using $S(n) = O(n\log^2(n)/\log\log n)$ space and $Q(n) = O(\log(n)/\log\log n)$ query time.*

We prove the above theorem in two steps, using the following lemma.

Lemma 7.1. *(i) The HCC problem in $\mathbb{R}$ can be reduced to $O(\log n)$ sub-problems of the sum-max problem on n points each. (ii) A sum-max problem on n points can be reduced to a (weighted) 2D orthogonal range counting problem on n points.*

Proof. Our approach starts by looking at the underlying tree structure in $\mathscr{G}$. We split $\mathscr{G}$ into its heavy path decomposition. To prove part (i) of the lemma, we actually need to look at the specific details of the heavy-path decomposition which can be described as follows. Start from the root of $\mathscr{G}$ and follow a path to a leaf of $\mathscr{G}$ where at every node u of $\mathscr{G}$, we always descend to a child of u that has the largest subtree; this easily yields the property that after removal of π, $\mathscr{G}$ will be decomposed into a number of forests where each forest is at most half the size of $\mathscr{G}$. Then the heavy-path decomposition is built recursively, by recursing on every resulting forest. It is easily seen that the depth of the recursion is $O(\log n)$.

Let $\diamond_i$ be the set of paths obtained at the i'th-level of the recursion. An important observation here is that the paths in $\diamond_i$ are "independent" meaning, no vertex u of a path $\pi \in \diamond_i$ is a descendent or ancestor of a vertex v of a different path $\pi' \in \diamond_i$. As a result, we claim it is sufficient to solve the HCC problem on the paths $\diamond_i$, for every

$i = 1, \cdots, O(\log n)$, and then sum up the $O(\log n)$ results; the set of paths in $\diamond_i$ defines the i-th sub-problem and thus it remains to show how it can be reduced to a sum-max problem.

We now build an instance of the sum-max problem on $\diamond_i$: we use the same input set P but with different colors and also the points will receive weights, as follows. For every path in $\diamond_i$, we define a new color class, i.e., for the set C in the definition of the sum-max problem we have $|C| = |\diamond_i|$. Let c be the (original) color of a point p in the HCC problem (i.e., in graph $\mathscr{G}$). Consider the position of the color c in $\mathscr{G}$ and the path π_c that connects it to the root of $\mathscr{G}$. Let $\pi_j \in \diamond_i$ be the path that intersects π_c (if there's no such path, p is not stored in the i-th subproblem) on a vertex v. The weight of p will be a prefix sum of the weights in π_j: we start from the root of π_j and add the weights all the way to v. Note that an unweighted HCC can be thought of as a weighted instance of HCC with weights one. By the definition of the sum-max problem, the answer to a sum-max query will yield the number of vertices (or the total weights of the vertices) of the paths in $\diamond_i$ that need to be counted in the HCC problem. This concludes the proof of part (i) of the lemma.

To prove part (ii), we transform each point into an orthogonal range in 2D, inspired by the previous solutions to 1D color counting. Consider an instance of a sum-max problem where the x-coordinate of a point p is p_x, its color is $c(p)$ and its weight is $w(p)$. For a point $p^{(i)}$, denote the first point of the same color and greater weight to its left (resp. right) with $p^{(\ell)}$ (resp. $p^{(r)}$). Observe $p^{(i)}$ is only counted by a sum-max query I if we have both $p_x^{(i)} \in I$ and $I \subset (p^{(\ell)}, p^{(r)})$. Based on this observation, we associate to $p^{(i)}$ the two dimensional region $(p_x^{(\ell)}, p_x^{(i)}] \times [p_x^{(i)}, p_x^{(r)})$ (If $p^{(\ell)}$ or $p^{(r)}$ does not exist, make the region unbounded in the corresponding direction). We do this transformation for all points in our input. For a query range $I = [I_1, I_2]$ we map it to the point $q = (I_1, I_2)$; by our observation, q precisely stabs the rectangles which correspond to the heaviest points of every color that lies inside I.

Thus, we have reduced the problem to the following: our input is a set of axis-aligned rectangles where each rectangle is assigned a weight and given a query point $q = (q.x, q.y)$, the goal is to sum up the weights of the rectangles that contain q, a.k.a, an instance of the rectangle stabbing problem. By a simple known reduction, this problem reduces to 2D orthogonal range counting: simply turn an input rectangle $[x_1, y_1] \times [x_2, y_2]$ where $x_1 < x_2, y_1 < y_2$ and with weight w into four points: points (x_1, y_1) and (x_2, y_2) with weight w and points (x_1, y_2) and (x_2, y_1) with weight $-w$. Computing the answer to the dominance query with point $(q.x, q.y)$ will count w only when q is inside the rectangle as otherwise the weights w and $-w$ will cancel each other out. □

Theorem 7.2 follows easily from Lemma 7.1 since we only need $O(\log n)$ sum-max data structures on $O(n)$ points; each reduces to weighted orthogonal range counting and the final observation is that we can combine all of the $O(\log n)$ data structures in one 2D orthogonal range counting data structure on $O(n \log n)$ points. Using known results, this can be solved with $O(n \log^2(n)/\log\log n)$ space and $O(\log(n)/\log\log n)$ query time [41] although other trade-offs are also possible. For example, by plugging

in other known results for weighted 2D range counting, we can also obtain $O(n \log n)$ space and $O(\log^{2+\varepsilon} n)$ query time, for any constant $\varepsilon > 0$.

Lower Bounds and Equivalence

Here we will look at equivalent problems to the HCC problem in 1D. Aside from showing that this problem has interesting and non-trivial connections to other problems, the results in this section imply an $\Omega(\log n / \log \log n)$ query lower bound for our problem which shows that the query time of our data structure from the previous section is optimal.

Theorem 7.3. *The following problems defined on an input set P of size n are equivalent, up to a constant factor blow up in space and query time and potentially an additive term in the query time for answering predecessor queries.*

- *[P1]: Unweighted HCC on a generalized caterpillar of size $O(n)$.*
- *[P2]: The sum-max problem with $O(\log n)$ bit long integer weights.*
- *[P3]: 2D orthogonal range counting with $O(\log n)$ bits long integer weights.*

Proof. The argument in the previous section shows that P1 reduces to P2 since in a generalized caterpillar, there are only two levels in the heavy-path decomposition so there is no blow up of a $\log n$ factor in the space complexity. P2 in 1D reduces to P3 using standard techniques, using the same transformation from 1D color counting to 2D range counting [55]. The non-trivial direction is to reduce P3 to P1. We do this in a step-by-step fashion.

Claim 7.3.1. *P3 can be reduced to the orthogonal range counting problem on $\varepsilon \log n$ bit-long integer weights, for any constant $\varepsilon > 0$.*

Proof. Let $X = 2^{\varepsilon \log n}$. Given a weighted point set P for P3, store the weights modulo X in a structure for $\varepsilon \log n$ bit weights. Now, we can strip away the $\varepsilon \log n$ least significant bits of the original weights and repeat this process $1/\varepsilon$ times to prove the claim. □

Claim 7.3.2. *For any constant s, P3 can be reduced to $O(n^{1-2^{-s}})$ sub-problems of P3 on instances of size $O(n^{2^{-s}})$ where a query in the original problem can be reduced to $O(1)$ queries on some of the sub-problems.*

Proof. We adapt the grid method by Alstrup et al. [22]. Observe that as weights are integers, summing up the weights inside a query rectangle can be done using additions and subtractions of four dominance queries of the form $(-\infty, x_1] \times (-\infty, x_2]$.

Build a $\sqrt{n} \times \sqrt{n}$ grid such that each row and column contains $\sqrt{n}$ input points. Then, use a $\sqrt{n} \times \sqrt{n}$ table T to store partial sums, as follows: the cell (i, j) of T stores the sum of all the weights in grid cells (i', j') with $i' \leq i$ and $j' \leq j$. After storing T, we recurse on the set of points stored in each row as well as each column

and stop the recursion at depth s. At every sub-problem at depth s of the recursion, we have $O(n^{2^{-s}})$ points left and they become the claimed sub-problems in the lemma.

To bound the number of sub-problems, observe that if a sub-problem at depth i has m points, then it creates $2\sqrt{m}$ problems (one for every row and column) involving $\sqrt{m}$ points each in depth $i+1$. By unrolling the recursion, we can see that at depth s of the recursion, we will have $2^s n^{1-2^{-s}} = O(n^{1-2^{-s}})$ sub-problems since s is a constant, as claimed.

Now consider a query $q = (-\infty, q_x] \times (-\infty, q_y]$. Observe that by using two predecessor queries, we can find the grid cells g that contains the query point. Assume g corresponds to the cell (i, j) in the table T and consider the cell $(i-1, j-1)$. We have stored the sum of all the weights in the cells (i', j') with $i' < i$ and $j' < j$. This value gives us the total sum of the weights in the grid cells that are completely contained in q. Next, q is decomposed into two disjoint queries, one in a row containing the (q_x, q_y) point and another one in the column containing the same point. These queries can then be answered recursively until we reach the s-th level of the recursion. Thus, in total we will need to answer $2^s = O(1)$ queries. □

Claim 7.3.3. *After performing the reductions in Claims 7.3.1 and 7.3.2, P3 can be reduced to an instance of P2 where there are at most n^ε colors and where the maximum weight is at most n^ε, for any constant $\varepsilon > 0$.*

Proof. Pick s in Claim 7.3.2 such that each sub-problem has at most $0.5n^\varepsilon$ points. Consider one such sub-problem involving m points. We do a reduction inspired by Larsen and Walderveen [69]. Consider an input point $p = (x_i, y_i)$ with weight $w(p)$. p will be mapped to two points $(-x_i)$ and (y_i). They are first stored in a sum-max data structure with weight $w(p)$ and color i. They are also stored in a second sum-max data structure with weights $w(p)$ but with different colors of $2i$ and $2i+1$.

Now, given a query range $q = (-\infty, q_x] \times (-\infty, q_y]$, we query both sum-max data structures with interval $[-q_x, q_y]$ and then subtract their results. If the point p is inside q, the first data structure counts $w(p)$ once but the second data structure counts them twice and thus their subtraction includes $w(p)$ only once. If p is not inside q, none of the data structures counts $w(p)$.

Finally, note that the total number of colors is at most $2m \leq n^\varepsilon$. □

Claim 7.3.4. *An instance of P2 where there are at most n^ε colors and where the maximum weight is at most n^ε, for any constant $0.5 > \varepsilon > 0$, can be reduced to P1 on a generalized caterpillar of size $O(n)$.*

Proof. We build a caterpillar graph $\mathscr{G}$ with a central path of length n^ε. Then, we attach a path of length n^ε to every vertex on the central path; call these attached paths, *legs*. The total size of the caterpillar graph is at most $n^{2\varepsilon} \leq n$.

Consider an input point p, with color i and weight $w \leq n^\varepsilon$. In our HCC problem, we assign it a color that corresponds to the w-th vertex of the i-th leg. Now, observe that given a query $I = [I_1, I_2]$ to the sum-max problem, asking the same query on $\mathscr{G}$ will produce the answer to the sum-max problem: all the vertices on a leg have the

same color which is different from the color of all the other legs. Furthermore, at every leg we simply need to find its lowest vertex that is contained in the query range which is equivalent to finding the point of maximum weight in the same color class. □

Observe that the proof follows directly using the above claims. Note that at each step, we might blow up the query time and the space by a constant factor. Also, Claim 7.3.2 requires a constant number of predecessor queries. Depending on the assumptions on the coordinates of the points this can take a varying amount of time but it is dominated by the actual cost of answering the range counting queries in any reasonable model of computation. □

As a consequence of the above equivalence, we can get a number of conditional lower bounds for HCC queries on trees.

Corollary 7.1. *The barrier of $S(n)Q(n) = \Omega(n(\log n/\log\log n)^2)$ for weighted 2D range counting data structure also applies to unweighted HCC queries, even for graphs as simple as generalized caterpillar graphs. The query lower bound of $\Omega(\log n/\log\log n)$ also applies to the HCC problem. Here $S(n)$ and $Q(n)$ refer to the space and query complexities.*

Finally we remark that if the graph $\mathscr{G}$ is a path, then the HCC problem simply reduces to the range max queries which do have more efficient solutions (e.g., with $O(n)$ space and $O(1)$ query time [50] plus $O(1)$ predecessor queries). And thus, caterpillar graphs are among the simplest graphs on which the above reduction is possible.

7.4 General Hierarchical Color Counting Queries

Now we shift our attention to the problem where the underlying graph $\mathscr{G}$ is a directed acyclic graph (DAG). This variant is clearly more complicated than the tree variant. However, we observe a very curious behavior, namely, the preprocessing bound is much higher than the space complexity. We show a conditional lower bound on the preprocessing time using a reduction from the orthogonal vectors problem.

A Reduction from Orthogonal Vectors

We will reduce the Orthogonal Vectors problem to the 1D HCC problem.

Theorem 7.4. *Assuming the Orthogonal Vectors conjecture, any solution to the 1D hierarchical color counting problem on a DAG using $P(n)$ preprocessing time and $Q(n)$ query time, must obey $P(n) + nQ(n) \geq n^{2-o(1)}$.*

Note that in the HCC problem, a query time of $O(|\mathscr{G}|)$ is trivial by simple graph traversal methods. As a result, the above reduction shows that any non-trivial solution (besides $n^{o(1)}$ factor improvements) must have a large preprocessing time.

Proof. Let $\eta = \frac{n}{\log^c(n)}$ for a large enough constant c. We build an instance of HCC with η points, and a DAG $\mathscr{G}$ with $O(\eta)$ vertices but with $O(n)$ edges. We reduce the orthogonal vectors problem on η vectors of dimension $\log^c n$ to this instance of HCC, notice that $n^{2-o(1)} = \eta^{2-o(1)}$.

Given two sets A and B of η boolean vectors in $\{0,1\}^d$, we will construct the following DAG $\mathscr{G}$. $\mathscr{G}$ will have three layers. For each vector in A create a vertex (i.e., a category) in what we denote the first layer. Now for each of the d coordinates of the vectors create a vertex in the second layer. Lastly create a vertex in the third layer for each vector in B.

Create the following edges: For a vertex corresponding to vector a_i in the first layer create an outgoing edge to all coordinate vertices in the second layer in which a_i has a one at that corresponding coordinate. Then, for a coordinate vertex in the second layer create an outgoing edge to all vectors in the third layer where the corresponding vector in B has a one at that coordinate. This clearly takes $O(\eta d) = O(n)$ to construct and the construction is illustrated in Figure 7.1.

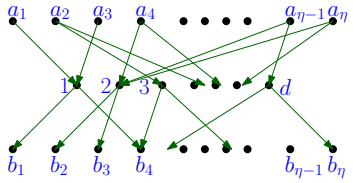

Figure 7.1: The underlying hierarchy DAG in the Orthogonal Vectors reduction

To figure out whether there exists a vector $a \in A$ and a vector $b \in B$ such that a and b are orthogonal, we do the following. Create a point in $\mathbb{R}$ for each of the vertices in the third layer; their locations do not matter as long as they are distinct. We use n queries by simply querying each point individually and thus each query interval has just one point inside it!

Claim 7.4.1. *Consider a HCC query that contains the point p_i that corresponds to a vector $b_i \in B$. There is no vector in A that is orthogonal to b_i, if and only if the output size is $|b_i|_1 + 1 + \eta$ where $|b_i|_1$ is the number of ones in vector b_i.*

Proof. First, let us consider the case when there is no vector in A that is orthogonal to b_i. Consider an arbitrary vector $a_j \in A$. Since a_j is not orthogonal to b_i, there exists a coordinate k where both b_i and a_j have a 1 at that coordinate. This implies that a_j is connected to b_i via the k-th vertex in the middle. As a result, all vectors in A are ancestors of b_i and since b_i is connected to $|b_i|_1$ vertices in the middle, the output of the HCC query will be as claimed.

The converse also follows by a similar argument. If a vector $a_j \in A$ does not share a 1 coordinate with b_i, then this corresponds to one of the vertices in the top layer not being counted by the query, hence the output is less than $|b_i|_1 + 1 + \eta$. □

Since one can easily store the values $|b_i|_1$ in $O(\eta)$ space, we can solve the Orthogonal Vectors problem using η queries on a solution for the HCC on the

aforementioned DAG. The DAG has size $O(\eta d) = O(n)$ and thus we obtain the lower bound $P(n) + nQ(n) \geq \eta^{2-o(1)} = n^{2-o(1)}$. □

A Data Structure for General DAGs for HCC

Despite the lower bound in the previous section, it is possible to give a non-trivial data structure for HCC queries on a general DAG, however, our goal is to to reduce the space complexity rather than the preprocessing time. Surprisingly, this is possible and in fact we can achieve a substantial improvement in space complexity.

Theorem 7.5. *It is possible to solve the HCC problem on $O(n)$ points in 1D on a DAG $\mathscr{G}$ of size n using $\tilde{O}(n^2)$ preprocessing time, $\tilde{O}(n^{3/2})$ space and $O(\log(n))$ query time.*

Proof. We start by remarking that we cannot hope to reduce the preprocessing time, as shown by our conditional lower bound, however and rather surprisingly, we show that the space can be reduced to $\tilde{O}(n^{3/2})$.

Assume the input coordianates have been reduced to rank space (i.e., between 1 and n). Let $I_q = (i, j)$ be the query interval. Observe that we can afford to store the answer explicitly when $|I_q| \leq \sqrt{n}$ (i.e., "short" queries) since the number of such queries is $O(n^{3/2})$. This allows us to answer such queries in constant time. The subtle challenge, however, is to do it within $\tilde{O}(n^2)$ preprocessing time as the obvious solution could take much longer.

We start by computing the transitive closure $\mathscr{G}^c$ of $\mathscr{G}$, which takes $\tilde{O}(n^2)$ time. For a point p_i denote by c_i its color in $\mathscr{G}$ and let d_i be the number of parents of p_i in the transitive closure $\mathscr{G}^c$. We create d_i copies of the point p_i at the same position as p_i and assign each a unique parent of p_i as color. The end result will be a set of $O(n^2)$ points such that every point has a unique color and such that computing the number of colors in an interval $I = [I_1, I_2]$ will yield the answer to the hierarchical query with the same interval. This essentially gives a "flat" representation of the hierarchical color structure, and consequently, using the existing solutions for CRC queries, we can compute the answer to all the short queries in $\tilde{O}(n^{3/2})$ time and store the results in a table. Thus, short queries can be answered in constant time, using $\tilde{O}(n^{3/2})$ space and $\tilde{O}(n^2)$ preprocessing time.

It remains to show how to deal with the long queries. Note that we can repeat the above process for all the queries to obtain a "flat representation" but doing so will yield a $\tilde{O}(n^2)$ space complexity. The key idea here is that we can "compress" the flat representation down to $O(n^{3/2})$ space, as follows. Keep in mind that the compressed representation only needs to deal with queries I_q such that $|I_q| \geq \sqrt{n}$. Partition the set of original n points into $2\sqrt{n}$ subsets of $\sqrt{n}/2$ consecutive points. For every subset P_i and every color class c (in the flat representation), delete all the points of color c in P_i except for the points in the smallest and the largest position. This will leave at most two points of color c in all subsets and thus there will be at most $O(\sqrt{n})$ points of color c in all subsets. Over n colors this yields $O(n^{3/2})$ points. We store them in a data structure for CRC queries and this will take $\tilde{O}(n^{3/2})$ space.

The claim is that the compressed representation answers queries correctly. Consider a query interval I_q of size at least $\sqrt{n}$ and assume to the contrary that there used to be a point p of color c inside I_q in a subset P_i but p got deleted during the compression step. However, we have kept the rightmost point, p_1, and the leftmost point, p_2, of color c inside P_i. Now, observe that since P_i contains at most $\sqrt{n}/2$ points, either its rightmost point or its leftmost point (or both) must be inside I_q. As a result, either p_1 or p_2 must be inside I_q, a contradiction. The preprocessing time here is also trivially $\tilde{O}(n^2)$. □

7.5 Sub-Category Range Counting and Equivalence

Here, we will focus on SCRC queries.

Equivalences

We show that when the catalog tree $\mathscr{G}$ is a path, then the SCRC problem in 1D is equivalent to a number of well-studied problems, as follows.

Theorem 7.6. *The following problems are equivalent: (i) SCRC when $\mathscr{G}$ is a single path on a one-dimensional input P, (ii) 3-sided distinct coordinate counting (for a planar point set P), (iii) 3-sided color counting (for a planar point set P), (iv) 3D dominance color counting (for a 3D point set P), and finally (v) 3D dominance counting (for a 3D point set P).*

To prove this, first in the appendix (Section 7.6), we show the following lemma.

Lemma 7.2. *When $\mathscr{G}$ is a path, the SCRC problem on a one-dimensional input P is equivalent to the 3-sided distinct coordinate counting problem.*

The above lemma shows that (i) and (ii) are equivalent. Next, we observe that (ii), (iii), and (iv) all reduces to (v): (iii) is a generalization of (ii), the reduction from (iii) to (iv) is standard by mapping a 2D input point (x_i, y_i) to the 3D point $(-x_i, y_i, x_i)$ and the 3-sided query range $\mathscr{R} = [q_\ell, q_r] \times (-\infty, q_t]$ to the 3D dominance range $(-\infty, -q_\ell] \times (-\infty, q_t] \times (-\infty, q_x]$. The reduction from (iv) to (v) was shown by Saladi [81]. Thus, the only remaining piece of the puzzle is to show a reduction from (v) to (ii). We do this next.

Theorem 7.7. *3D dominance counting can be solved by 3-sided distinct coordinate counting.*

Proof. Consider an instance of 3D dominance counting. First, we reduce the instance to rank space which means we can assume that query coordinates are integers and the input coordinates are *distinct* integers between 1 and n. For an input point $p_i = (x_i, y_i, z_i)$, we create four points $p_i^1 = (-x_i, z_i)$, $p_i^2 = (y_i, z_i)$, $p_i^3 = (-x_i, z_i)$ and $p_i^4 = (y_i, z_i + 0.5)$. Next, we create two 2D 3-sided distinct Y-coordinate counting structures. In the first structure we put p_i^1 and p_i^2, and in the second structure we put

p_i^3 and p_i^4. Given a query point $p = (x, y, z)$, we transform it into the 3-sided range $r_p = [x, y] \times (\infty, z + 0.5]$.

We claim the number of dominated points of p is the difference between the outputs of the two data structures on r_p. If p dominates p_i then $x_i \leq x$, $y_i \leq y$ and $z_i \leq z$, hence $-x \leq x_i$ and $z_i + 0.5 \leq z + 0.5$. This means that all four point are inside r_p. The first data structure counts 1, since the two points share the second coordinate and the second data structure counts 2 since they have distinct second coordinates.

The key observation is that if p does not dominate p_i at least one of $(-x_i, z_i)$ or (y_i, z_i) and $(y_i, z_i + 0.5)$ will not be in the range, which consequently implies hence the difference of the outputs will be zero (regarding p_i). The reduction is clearly linear in the size of the input set. □

The above equivalence, allows us to build a data structure for SCRC queries when $\mathscr{G}$ is a tree.

Corollary 7.2. *SCRC problem in 1D on category tree can be solved using $O(n \log^2 n / \log\log n)$ space and with the optimal query time of $O(\log n / \log\log n)$.*

Proof. We split the tree into its heavy path decomposition. Once again, we need to look deeper into the details of the heavy-path decomposition. Start from the root of $\mathscr{G}$ and follow a path π to a leaf of $\mathscr{G}$ where at every node u of $\mathscr{G}$, we always descend to a child of u that has the largest subtree. For every node $v \in \pi$, consider the subset $P_v \subset P$ that have a color from the set $\mathscr{G}_{\leq}(v)$. Build another instance of SCRC problem where the category graph is set to π, and a point $p \in P_v$ is assigned color v. In this instance of SCRC, the category graph is a path and by Theorem 7.6 it is equivalence to 3D dominance counting and thus it can be solved with $O(n \log n / \log\log n)$ space and with $O(\log n / \log\log n)$ query time [59]. Observe that if for the query pair (I, v_q), consisting of an interval I and a node $v_q \in \mathscr{G}$, we have $v_q \in \pi$, then the query can readily be answered. Otherwise, v_q must not be on the central path π. To handle such queries, we simply recurse on every tree that remains after removing π.

By the heavy path decomposition, the depth of the recursion is $O(\log n)$. Furthermore, the paths obtained at the depth i of the heavy path decomposition are independent and thus in total they contain $O(n)$ points. Over all the $O(\log n)$ levels, it blows up the space by a factor of $O(\log n)$. Note that to answer a query (I, v_q), we need to find the level i of the heavy path decomposition and a path π_i that contains v_q. However, this can simply be answered by placing a pointer from v_q to the appropriate data structure. Then, after finding π_i, the query can be answered using a single dominance range counting query in $O(\log n / \log\log n)$ time.

The query time of the above data structure is also optimal which follows from combining our reductions with previously known lower bounds [75].

□

A Conditional Lower Bound for SCRC

For SCRC where the underlying category graph is a DAG there is a trivial upper bound of $O(n^2)$ space and $O(\log n)$ query time: for each node v_i in the DAG store a 1D CRC structure on $\mathscr{G}_{\leq}(v_i)$. To answer a query (I, v_q), we simply query the CRC structure on v_q with the interval I. The conditional lowerbound on HCC can easily be extended to SCRC. See Section 7.6 in the appendix for details.

Corollary 7.3. *Assuming the Orthogonal Vectors conjecture, any solution to the 1D sub-category range counting problem on a DAG using $P(n)$ preprocessing time and $Q(n)$ query time, must obey $P(n) + nQ(n) \geq n^{2-o(1)}$.*

7.6 Appendix

HCC is at Least as Hard as Color Counting

Consider an instance of regular colored counting given by a point set P in $\mathbb{R}^d$ where each point is assigned a color from a set C. We build two hierarchical color counting data structures. In the first data structure, all colors in C are represented by leaf nodes in a balanced binary tree T_1; for simplicity we assume $|C|$ is a power of two; otherwise, we add dummy colors to C. In the second data structure, for every pair of sibling leaves c and c', we "collapse them" into their parent c_p, to obtain a second balanced binary tree T_2, meaning, any point that had color c or c' will receive c_p as its color. The resulting colored point set will be stored in the second hierarchical color counting data structure.

Given a query range $\mathscr{R}$ for the regular colored counting problem, we query both data structures with $\mathscr{R}$ and report their difference. We claim it will be the correct answer to the regular colored counting query.

To see this, we can consider two sibling leaf colors c and c' and their parent c_p in T_1. Case one is when none of them is in the query range $\mathscr{R}$. In this case, neither data structure will count anything and thus their difference will also not count either color. The second case is when exactly one of them, say c is in the query. In this case, the first data structure counts the leaf c and the path connecting c_p to the root of T_1 where as the second counts only the latter and thus the difference counts c exactly once. Lastly if both leaves are in the query, the first data structure counts two more leafs when compared to the second data structure (the two leaf nodes in the query) and we again get the correct output. The three cases are summarized in Figure 7.2.

Reducing the SCRC Problem to the 3-sided Distinct Coordinate Counting Problem

Here, we would like to show that when $\mathscr{G}$ is a path, the SCRC Problem reduces to the 3-sided distinct coordinate counting problem.

To do that, first consider a 1D input point set for the SCRC problem. Map the input point x_i with color $c_i \in \mathscr{G}$, to the point $(x_i, h(c_i))$, where $h(c_i)$ is the number of

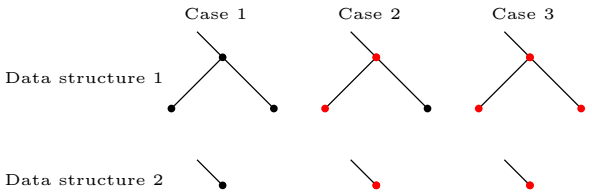

Figure 7.2: Each case, where the red dots corresponds to points counted

nodes below c_i on the path $\mathscr{G}$. Store the resulting point set in a data structure for the distinct coordinate counting queries. Then, given a query interval $[a,b]$ and query node c_q, we create the 3-sided query range $\mathscr{R} = [a,b] \times (-\infty, h(c_q)]$. Observe that if a point $(x_i, h(c_q))$ lies inside $\mathscr{R}$, it implies that c_i is below c_q and that $x_i \in [a,b]$. Thus, the number of distinct Y-coordinates inside $\mathscr{R}$ is precisely the answer to the SCRC problem.

To show the converse reduction, consider a 2D point set P for the 3-sided distinct coordinate counting. We create a node $v(y)$ for each distinct Y-coodinate y in the input set, meaning, $\mathscr{G}$ is a path that has as many vertices as the number of distinct Y-coordinates in P.

For each point $p_i = (x_i, y_i)$, we create a 1D point with x-coordinate x_i and color $v(y_i)$. Consider a query range $\mathscr{R} = [a,b] \times (-\infty, c]$ and let y be the predecessor of c among the Y-coordinates of P. We create the 1D range $[a,b]$ and query the node $v(y)$. It is straightforward to verify that the answer to the SCRC is exactly the number of distinct Y-coordinates inside $\mathscr{R}$.

Reducing the OV Problem to SCRC on a DAG

Corollary 7.3. *Assuming the Orthogonal Vectors conjecture, any solution to the 1D sub-category range counting problem on a DAG using $P(n)$ preprocessing time and $Q(n)$ query time, must obey $P(n) + nQ(n) \geq n^{2-o(1)}$.*

Proof. We pick the same underlying graph as in theorem 7.4 and the point set P of η points p_i such that the color of p_i is equivalent to node b_i. We query the SCRC η times, each time using a different a_i as our query node and an interval which spans all points of P. If for a query node a_i the output is less than η we know that a_i is orthogonal to some b_j, since they do not share a coordinate where they both have a one. □

Bibliography

[1] Amir Abboud and Søren Dahlgaard. Popular conjectures as a barrier for dynamic planar graph algorithms. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 477–486. IEEE, 2016. 11

[2] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 434–443. IEEE, 2014.

[3] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for lcs and other sequence similarity measures. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 59–78. IEEE, 2015.

[4] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 375–388, 2016.

[5] Amir Abboud, Arturs Backurs, Karl Bringmann, and Marvin Künnemann. Fine-grained complexity of analyzing compressed data: Quantifying improvements over decompress-and-solve. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 192–203. IEEE, 2017.

[6] Amir Abboud, Aviad Rubinstein, and Ryan Williams. Distributed pcp theorems for hardness of approximation in p. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 25–36. IEEE, 2017.

[7] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is valiant's parser. *SIAM Journal on Computing*, 47(6):2527–2555, 2018.

[8] Amir Abboud, Karl Bringmann, Holger Dell, and Jesper Nederlof. More consequences of falsifying seth and the orthogonal vectors conjecture. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 253–266, 2018.

[9] Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. *SIAM Journal on Computing*, 47(3):1098–1122, 2018. 11

[10] Peyman Afshani. Improved pointer machine and I/O lower bounds for simplex range reporting and related problems. In *Symposium on Computational Geometry (SoCG)*, pages 339–346, 2012. 11, 31, 72

[11] Peyman Afshani. A lower bound for dynamic fractional cascading. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2229–2248. SIAM, 2021. 12

[12] Peyman Afshani and Anne Driemel. On the complexity of range searching among curves. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 898–917, 2017. 68, 70, 79

[13] Peyman Afshani, Lars Arge, and Kasper Dalgaard Larsen. Orthogonal range reporting in three and higher dimensions. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 149–158, 2009. 71

[14] Peyman Afshani, Lars Arge, and Kasper Dalgaard Larsen. Orthogonal range reporting: query lower bounds, optimal structures in 3-d, and higher-dimensional improvements. In *Symposium on Computational Geometry (SoCG)*, pages 240–246, 2010. 68, 69, 70

[15] Peyman Afshani, Lars Arge, and Kasper Green Larsen. Higher-dimensional orthogonal range reporting and rectangle stabbing in the pointer machine model. In *Proceedings of the Twenty-Eighth Annual Symposium on Computational Geometry*, SoCG '12, page 323–332, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450312998. doi: 10.1145/2261250.2261299. URL `https://doi.org/10.1145/2261250.2261299`. 12

[16] Peyman Afshani, Lars Arge, and Kasper Green Larsen. Higher-dimensional orthogonal range reporting and rectangle stabbing in the pointer machine model. In *Symposium on Computational Geometry (SoCG)*, pages 323–332, 2012. doi: http://doi.acm.org/10.1145/2261250.2261299. 68, 69, 70, 71

[17] Pankaj K. Agarwal. Range searching. In J. E. Goodman, J. O'Rourke, and C. Toth, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, Inc., 2016. 69, 82

[18] Pankaj K. Agarwal and Jeff Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*. AMS Press, 1999. 69

[19] Udit Agarwal and Vijaya Ramachandran. Fine-grained complexity for sparse graphs. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 239–252, 2018. 11

[20] Divesh Aggarwal and Noah Stephens-Davidowitz. (gap/s) eth hardness of svp. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 228–238, 2018. 11

[21] Noga Alon and Joel H. Spencer. *The Probabilistic Method*. Wiley Publishing, 4th edition, 2016. ISBN 1119061954. 71

[22] 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. 28, 90

[23] Amihood Amir, Timothy M. Chan, Moshe Lewenstein, and Noa Lewenstein. On hardness of jumbled indexing. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 114–125, 2014. 18, 42, 44

[24] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless seth is false). In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 51–58, 2015. 11, 88

[25] Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 457–466. IEEE, 2016. 11

[26] Antal Balog and Endre Szemerédi. A statistical theorem of set addition. *Combinatorica*, 14(3):263–268, 1994. 47

[27] Sebastian Böcker. Simulating multiplexed SNP discovery rates using base-specific cleavage and mass spectrometry. *Bioinformatics*, 23(2):e5–e11, 01 2007. 43

[28] Michael Ben-Or. Lower bounds for algebraic computation trees. In *Proceedings of the fifteenth Annual ACM Symposium on Theory of Computing*, pages 80–86, 1983. 6

[29] Gary Benson. Composition alignment. In Gary Benson and Roderic D. M. Page, editors, *Algorithms in Bioinformatics*, pages 447–461, 2003. 16, 43

[30] Jon Louis Bentley. Multidimensional divide-and-conquer. *Communications of the ACM (CACM)*, 23(4):214–229, 1980. 6

[31] Sebastian Bocker and Zsuzsanna Lipták. A fast and simple algorithm for the money changing problem. *Algorithmica*, 48(4):413–432, 2007. 16

[32] Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless seth fails. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 661–670. IEEE, 2014. 11, 88

[33] Péter Burcsi, Ferdinando Cicalese, Gabriele Fici, and Zsuzsanna Lipták. On table arrangements, scrabble freaks, and jumbled pattern matching. In Paolo Boldi and Luisa Gargano, editors, *Fun with Algorithms*, pages 89–101, 2010. 16, 18, 43, 44

[34] Diptarka Chakraborty, Lior Kamma, and Kasper Green Larsen. Tight cell probe bounds for succinct boolean matrix-vector multiplication. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1297–1306, 2018. 13

[35] Timothy M. Chan. Persistent predecessor search and orthogonal point location in the word RAM. In *Proc. 22ndProceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2011. To appear. 28

[36] Timothy M. Chan. Orthogonal Range Searching in Moderate Dimensions: k-d Trees and Range Trees Strike Back. In Boris Aronov and Matthew J. Katz, editors, *33rd International Symposium on Computational Geometry (SoCG 2017)*, volume 77 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:15, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-038-5. doi: 10.4230/LIPIcs.SoCG.2017.27. URL `http://drops.dagstuhl.de/opus/volltexte/2017/7226`. 28, 70

[37] Timothy M. Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 31–40, 2015. 18, 43, 44, 45

[38] Timothy M. Chan, Yakov Nekrich, Saladi Rahul, and Konstantinos Tsakalidis. Orthogonal Point Location and Rectangle Stabbing Queries in 3-d. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107, pages 31:1–31:14, 2018. 69

[39] Bernard Chazelle. Filtering search: A new approach to query-answering. *SIAM Journal of Computing*, 15(3):703–724, 1986. 27, 69

[40] Bernard Chazelle. Functional approach to data structures and its use in multidimensional searching. 17(3):427–462, 1988. ISSN 0097-5397. doi: http://dx.doi.org/10.1137/0217026. 7, 85

[41] Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal of Computing*, 17(3):427–462, 1988. 89

[42] Bernard Chazelle. Lower bounds for orthogonal range searching: I. the reporting case. *Journal of the ACM (JACM)*, 37(2):200–212, 1990. 11, 22, 46, 68, 69, 70, 71, 72

[43] Bernard Chazelle and Herbert Edelsbrunner. Linear space data structures for two types of range search. *Discrete & Computational Geometry*, Volume 2(1): 113,126, 1987. doi: http://www.springerlink.com/c={nt/p6822t1876386152}. 28

[44] Bernard Chazelle and Burton Rosenberg. Simplex range reporting on a pointer machine. *Computational Geometry*, 5(5):237 – 247, 1996. ISSN 0925-7721. doi: 10.1016/0925-7721(95)00002-X. 11, 22, 46, 72

[45] K. L. Chung and G. A. Hunt. On the zeros of $\sum_{1}^{n} \pm 1$. *Annals of Mathematics*, 50 (2):385–400, 1949. 55

[46] Ferdinando Cicalese, Gabriele Fici, and Zsuzsanna Lipt´ ak. Searching for Jumbled Patterns in Strings. In *Proc. of the Prague Stringology Conference 2009*, pages 105–117, 2009. 18, 42, 44

[47] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3 edition, 2008. 69

[48] Lech Duraj, Marvin Künnemann, and Adam Polak. Tight conditional lower bounds for longest common increasing subsequence. *Algorithmica*, 81(10): 3968–3992, 2019. 11, 88

[49] P. Erdös and S. J. Taylor. Some problems concerning the structure of random walk paths. *Acta Mathematica Hungarica*, 11(1-2):137–162, 3 1960. 55

[50] Johannes Fischer. Optimal succinctness for range minimum queries. In *Proc. 9thLatin American Symposium on Theoretical Informatics (LATIN)*, pages 158–169, 2010. 92

[51] M. Fredman and M. Saks. The cell probe complexity of dynamic data structures. In *Proc. 21stProceedings of ACM Symposium on Theory of Computing (STOC)*, pages 345–354, 1989. 13

[52] G. A. Freiman. Foundations of a structual theory of set addition. *Translation of Math. Monographs*, 37, 1973. 47

[53] Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. Conditional lower bounds for space/time tradeoffs. In *Workshop on Algorithms and Data Structures*, pages 421–436. Springer, 2017. 13

[54] W.T. Gowers. A new proof of Szemerédi's theorem. *Geometric & Functional Analysis GAFA*, 11(3):465–588, 2001. 47

[55] P. Gupta, R. Janardan, and M. Smid. Further results on generalized intersection searching problems: Counting, reporting, and dynamization. *Journal of Algorithms*, 19(2):282–317, 1995. ISSN 0196-6774. 7, 84, 90

[56] P. Gupta, R. Janardan, R. Saladi, and M. Smid. Computational geometry: generalized (or colored) intersection searching. In *Handbook of Data Structures and Applications*, chapter 67, pages 1–17. Chapman & Hall/CRC, 2017. 84

[57] Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *siam Journal on Computing*, 13(2):338–355, 1984. 87

[58] Meng He and Serikzhan Kazi. Data Structures for Categorical Path Counting Queries. In *Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 191, pages 15:1–15:17, 2021. 34, 84

[59] Joseph JaJa, Christian W. Mortensen, and Qingmin Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *Proc. 15thInternational Symposium on Algorithms and Computation (ISAAC)*, pages 558–568, 2004. 35, 85, 96

[60] R. Janardan and M. Lopez. Generalized intersection searching problems. *International Journal of Computational Geometry & Applications*, 3:39–69, 1993. 83, 84

[61] Julio Juárez-Xochitemol and Edgar Chávez. Quantum jumbled pattern matching. *arXiv preprint arXiv:2203.00164*, 2022. 20

[62] Haim Kaplan, Natan Rubin, Micha Sharir, and Elad Verbin. Efficient colored orthogonal range counting. *SIAM Journal of Computing*, 38(3):982–1011, 2008. 7, 8, 85

[63] D. Knuth, J. Morris, Jr., and V. Pratt. Fast pattern matching in strings. *SIAM Journal of Computing*, 6(2):323–350, 1977. 43

[64] Tomasz Kociumaka, Jakub Radoszewski, and Wojciech Rytter. Efficient indexes for jumbled pattern matching with constant-sized alphabet. In Hans L. Bodlaender and Giuseppe F. Italiano, editors, *Algorithms – ESA 2013*, pages 625–636, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-40450-4. 18, 44

[65] Tsvi Kopelowitz and Virginia Vassilevska Williams. Towards optimal set-disjointness and set-intersection data structures. 2020. 11

[66] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1272–1287. SIAM, 2016. 11

[67] Kasper Green Larsen. The cell probe complexity of dynamic range counting. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 85–94, 2012. 13

[68] Kasper Green Larsen. Higher cell probe lower bounds for evaluating polynomials. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 293–301. IEEE, 2012. 13

[69] 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 265–277, 2013. 7, 35, 85, 91

[70] Tanaeem M. Moosa and M. Sohel Rahman. Indexing permutations for binary strings. *Information Processing Letters (IPL)*, 110(18):795 – 798, 2010. 18, 44

[71] Tanaeem M. Moosa and M. Sohel Rahman. Sub-quadratic time and linear space data structures for permutation matching in binary strings. *Journal of Discrete Algorithms*, 10:5–9, 2012. 18, 44

[72] Yakov Nekrich. Four-dimensional dominance range reporting in linear space. *arXiv preprint arXiv:2003.06742*, 2020. 28

[73] Laxmi Parida. Gapped permutation patterns for comparative genomics. In *International Workshop on Algorithms in Bioinformatics*, pages 376–387. Springer, 2006. 16

[74] Rohit J. Parikh. On context-free languages. *Journal of the ACM (JACM)*, 13(4): 570–581, 1966. 15, 16, 43

[75] Mihai Pătraşcu. Unifying the landscape of cell-probe lower bounds. In *Proc. 49thProceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 434–443, 2008. 13, 85, 86, 87, 96

[76] Mihai Pătraşcu and Erik D. Demaine. Logarithmic lower bounds in the cell-probe model. 35:932–963, April 2006.

[77] Mihai Pătraşcu and Mikkel Thorup. Don't rush into a union: Take time to find your roots. In *Proc. 43rdProceedings of ACM Symposium on Theory of Computing (STOC)*, 2011. To appear. See also arXiv:1102.1783. 13

[78] Saladi Rahul. *Improved Bounds for Orthogonal Point Enclosure Query and Point Location in Orthogonal Subdivisions in* $\mathbb{R}^3$, pages 200–211. 69

[79] Saladi Rahul. An (almost) optimal solution for orthogonal point enclosure query in $\mathbb{R}^3$. *Mathematics of Operations Research*, 45(1):369–383, 2020. doi: 10.1287/moor.2019.0994. 69

[80] Imre Rusza. Sumsets and structure. In *Combinatorial Number Theory and Additive Group Theory*. 2008. 47

[81] Rahul Saladi. Approximate range counting revisited. 12(1), 2021. 35, 86, 95

[82] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, July 1948. ISSN 0005-8580. doi: 10.1002/j.1538-7305.1948.tb01338.x. 15, 30, 72

[83] Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983. ISSN 0022-0000. 87

[84] Endre Szemerédi. On sets of integers containing no k elements in arithmetic progression. *Acta Arithmetica*, 27:299–345, 1975. 47

[85] 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. 11, 46, 69, 71

[86] Thomas M. Thompson. *From Error-Correcting Codes Through Sphere Packings to Simple Groups*, volume 21. Mathematical Association of America, 1 edition, 1983. ISBN 9780883850374. URL `http://www.jstor.org/stable/10.4169/j.ctt5hh9fv`. 30, 72

[87] Virginia Vassilevska Williams. Towards tight approximation bounds for graph diameter and eccentricities. 2018. 11

[88] Omri Weinstein and Huacheng Yu. Amortized dynamic cell-probe lower bounds from four-party communication. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 305–314. IEEE, 2016. 13

[89] R. Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM Journal of Computing*, 47(5):1965–1985, 2018. 44

[90] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357 – 365, 2005. ISSN 0304-3975. International Colloquium on Automata, Languages and Programming (ICALP). 10, 68, 70, 87