Models and Techniques for Proving Data Structure Lower Bounds

Kasper Green Larsen

PhD Dissertation

Department of Computer Science
Aarhus University
Denmark

Models and Techniques for Proving Data Structure Lower Bounds

A Dissertation
Presented to the Faculty of Science and Technology
of Aarhus University
in Partial Fulfilment of the Requirements for the
PhD Degree

by
Kasper Green Larsen
October 26, 2013

Abstract

In this dissertation, we present a number of new techniques and tools for proving lower bounds on the operational time of data structures. These techniques provide new lines of attack for proving lower bounds in both the cell probe model, the group model, the pointer machine model and the I/O-model. In all cases, we push the frontiers further by proving lower bounds higher than what could possibly be proved using previously known techniques.

For the cell probe model, our results have the following consequences:

- The first $\Omega(\lg n)$ query time lower bound for linear space static data structures. The highest previous lower bound for any static data structure problem peaked at $\Omega(\lg n / \lg \lg n)$.
- An $\Omega((\lg n / \lg \lg n)^2)$ lower bound on the maximum of the update time and the query time of dynamic data structures. This is almost a quadratic improvement over the highest previous lower bound of $\Omega(\lg n)$.

In the group model, we establish a number of intimate connections to the fields of combinatorial discrepancy and range reporting in the pointer machine model. These connections immediately allow us to translate decades of research in discrepancy and range reporting to very high lower bounds on the update time t_u and query time t_q of dynamic group model data structures. We have listed a few in the following:

- For d-dimensional halfspace range searching, we get a lower bound of $t_u t_q = \Omega(n^{1-1/d})$. This comes within a $\lg \lg n$ factor of the best known upper bound.
- For orthogonal range searching, we get a lower bound of $t_u t_q = \Omega(\lg^{d-1} n)$.
- For ball range searching, we get a lower bound of $t_u t_q = \Omega(n^{1-1/d})$.

The highest previous lower bound proved in the group model does not exceed $\Omega((\lg n / \lg \lg n)^2)$ on the maximum of t_u and t_q.

Finally, we present a new technique for proving lower bounds for range reporting problems in the pointer machine and the I/O-model. With this technique, we tighten the gap between the known upper bound and lower bound for the most fundamental range reporting problem, orthogonal range reporting.

Acknowledgments

First and foremost, I wish to thank my wife, Lise, for all her support and encouragement. I also wish to thank her for helping me raise two wonderful children during my studies. I feel incredibly grateful for having such a lovely family. Finally, I wish to thank her for moving with me to Princeton with two so small children, a travel which was much tougher than if we had stayed close to family and friends. I'll never forget she did that for me.

Secondly, I wish to thank my advisor, Lars Arge, for an amazing time. He has done a fantastic job at introducing me to the academic world. In the beginning of my studies, when I was completely unknown to the community, Lars did a lot of work in introducing me to important researchers in my area. I would not have gotten to work with so many talented people, had it not been for Lars believing enough in me to tell all his peers about me. I also wish to thank him for the amount of freedom he has given me in my research. Without that, I would probably never had studied lower bounds, a field completely outside Lars' own area of expertise. Finally, I wish to thank him also for creating a great environment in our research group. I had an immensely fun time and enjoyed the many beers we had together. Thank you, Lars.

Next, I want to thank all the people at MADALGO. Each one of them has contributed to making these past years a great experience. In particular, I wish to thank Thomas Mølhave, both for introducing me to the group when I was still an undergraduate and also for being a great friend. A big thanks also goes to Peyman Afshani for all the great collaboration we've had. I also wish to give a special thanks to all my former office mates: Thomas Mølhave, Pooya Davoodi, Freek van Walderveen and Bryan T. Wilkinson. A thank you also goes to Else Magård and Ellen Kjemstrup for always helping me with practical issues. Finally, a special thanks also goes to Gerth S. Brodal and Peter B. Miltersen for lots of guidance and many valuable discussions.

A big thanks also goes to Huy L. Nguyen at Princeton University, who did a great job at showing me around at Princeton and making me feel welcome.

I also wish to thank all my coauthors for the very inspiring work we've done together, thank you: Peyman Afshani, Manindra Agrawal, Pankaj K. Agarwal, Lars Arge, Jens Bennedsen, Karl Bringmann, Gerth S. Brodal, Joshua Brody, Michael E. Caspersen, Timothy M. Chan, Benjamin Doerr, Stephane Durocher, Mark Greve, Allan G. Jørgensen, Kurt Mehlhorn, Jason Morrison, Thomas Mølhave, Huy L. Nguyen, Rasmus Pagh, Mihai Pătraşcu, Jeff M. Phillips, Jakob Truelsen, Freek van Walderveen, Bryan T. Wilkinson and Carola Winzen. Here a special thanks goes to Allan G. Jørgensen for introducing me to the area of lower bounds and for sharing many great ideas.

Next I want to thank Stephen Alstrup for inviting me to Copenhagen University last summer. I really enjoyed the days I spend there, the cool problems we worked on and the beers we had in the evenings.

I also want to give a very big thanks to Mikkel Thorup, who has been a great friend and incredibly fun to hang out with. Especially our hiking and bird watching trip in Palm Springs, the great wine pairing we had at Craft, New York and the fun visit at the local bar in New Brunswick are great memories that stand out. Thank you.

Finally, I want to give the biggest thanks to a person who has meant very much to me during my studies, Mihai Pătraşcu. Very sadly, he passed away last year, only 29 years old. I feel so privileged to have had him as my friend, and no other person has had a bigger impact on my research and thus my every day work life. I still remember when Allan Jørgensen came to my office in the beginning of my studies and handed me one of Mihai's papers, saying something like: "You have to read this. The proofs are so elegant and the results are so cool. We have to do this too!". Prior to this, I had only focused on upper bounds, but I was completely sold by Mihai's incredible proofs and inspiring writing. That was the start of the journey leading to the results in this dissertation. Later I was so fortunate to meet Mihai on several occasions and soon he was not only my idol, but even more a very good friend. Mihai was always up for fun and lots of beer and wine, as I'm sure everyone who knew him agrees with. Very sadly our friendship lasted only a few years and I clearly remember how shocked I was when Mihai told me at SODA'11 that he had been diagnosed with brain cancer just a month earlier. Amazingly, it didn't seem to kill his fantastic spirit: Just months later, he had undergone brain surgery and I remember that he wrote an email just a couple of hours later regarding some technical details for the final version of our joint paper! Later I visited him in New York a couple of times, but sadly I could see with every visit how the cancer was getting to him. Especially my last visit, just two weeks before he passed, was very heart breaking. Nonetheless, I'm very happy I got to see him that last time. In that respect, I also wish to thank Mihai's wife, Mira, for inviting me to their home. My best wishes for the future goes to her. Finally, I want to say that I feel very privileged to have been an invited speaker at the FOCS'12 workshop in Mihai's memory. A big thanks to the organizers.

This dissertation is dedicated to the memory of Mihai Pătraşcu.

Kasper Green Larsen,
Aarhus, October 26, 2013.

In memory of Mihai Pătraşcu

Preface

My main research area has been data structures, with an emphasis on both range searching and lower bounds. For maximum coherence, I have chosen to base this dissertation only on (a subset of) my work in proving lower bounds. It would be a shame not to mention the many other interesting problems I have worked on during the past years. This section therefore aims to give a brief overview of all the results I obtained during my studies.

At the time of writing, I have published 18 papers and have another 3 manuscripts currently in submission. These papers are listed here:

[2] P. Afshani, P. K. Agarwal, L. Arge, K. G. Larsen, and J. M. Phillips. (approximate) uncertain skylines. In *Proc. 14th International Conference on Database Theory*, pages 186–196, 2011.

[3] P. Afshani, M. Agrawal, B. Doerr, K. Mehlhorn, K. G. Larsen, and C. Winzen. The query complexity of finding a hidden permutation. Manuscript.

[4] P. Afshani, L. Arge, and K. D. Larsen. Orthogonal range reporting in three and higher dimensions. In *Proc. 50th IEEE Symposium on Foundations of Computer Science*, pages 149–158, 2009.

[5] P. Afshani, L. Arge, and K. D. Larsen. Orthogonal range reporting: Query lower bounds, optimal structures in 3d, and higher dimensional improvements. In *Proc. 26th ACM Symposium on Computational Geometry*, pages 240–246, 2010.

[6] P. Afshani, L. Arge, and K. G. Larsen. Higher-dimensional orthogonal range reporting and rectangle stabbing in the pointer machine model. In *Proc. 28th ACM Symposium on Computational Geometry*, pages 323–332, 2012.

[13] L. Arge and K. G. Larsen. I/O-efficient spatial data structures for range queries. *SIGSPATIAL Special*, 4(2):2–7, July 2012.

[14] L. Arge, K. G. Larsen, T. Mølhave, and F. van Walderveen. Cleaning massive sonar point clouds. In *Proc. 18th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems*, pages 152–161, 2010.

[26] K. Bringmann and K. G. Larsen. Succinct sampling from discrete distributions. In *Proc. 45th ACM Symposium on Theory of Computation*, 2013. To appear.

[27] G. S. Brodal and K. G. Larsen. Optimal planar orthogonal skyline counting queries. Manuscript.

[28] J. Brody and K. G. Larsen. Adapt or die: Polynomial lower bounds for non-adaptive dynamic data structures. Manuscript.

[29] M. E. Caspersen, K. D. Larsen, and J. Bennedsen. Mental models and programming aptitude. In *Proc. 12th SIGCSE Conference on Innovation and Technology in Computer Science Education*, pages 206–210, 2007.

[31] T. M. Chan, S. Durocher, K. G. Larsen, J. Morrison, and B. T. Wilkinson. Linear-space data structures for range mode query in arrays. In *Proc. 29th Symposium on Theoretical Aspects of Computer Science*, pages 290–301, 2012.

[32] T. M. Chan, K. G. Larsen, and M. Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proc. 27th ACM Symposium on Computational Geometry*, pages 1–10, 2011.

[46] M. Greve, A. G. Jørgensen, K. D. Larsen, and J. Truelsen. Cell probe lower bounds and approximations for range mode. In *Proc. 37th International Colloquium on Automata, Languages, and Programming*, pages 605–616, 2010.

[49] A. G. Jørgensen and K. G. Larsen. Range selection and median: Tight cell probe lower bounds and adaptive data structures. In *Proc. 22nd ACM/SIAM Symposium on Discrete Algorithms*, pages 805–813, 2011.

[52] K. G. Larsen. On range searching in the group model and combinatorial discrepancy. In *Proc. 52nd IEEE Symposium on Foundations of Computer Science*, pages 542–549, 2011.

[53] K. G. Larsen. The cell probe complexity of dynamic range counting. In *Proc. 44th ACM Symposium on Theory of Computation*, pages 85–94, 2012.

[54] K. G. Larsen. Higher cell probe lower bounds for evaluating polynomials. In *Proc. 53rd IEEE Symposium on Foundations of Computer Science*, pages 293–301, 2012.

[55] K. G. Larsen and H. L. Nguyen. Improved range searching lower bounds. In *Proc. 28th ACM Symposium on Computational Geometry*, pages 171–178, 2012.

[56] K. G. Larsen and R. Pagh. I/O-efficient data structures for colored range and prefix reporting. In *Proc. 23rd ACM/SIAM Symposium on Discrete Algorithms*, pages 583–592, 2012.

[57] K. G. Larsen and F. van Walderveen. Near-optimal range reporting structures for categorical data. In *Proc. 24th ACM/SIAM Symposium on Discrete Algorithms*, 2013. To appear.

These papers can roughly be categorized as follows: Papers [4, 5, 32, 56, 6, 13, 57, 27] all study variants of the orthogonal range reporting problem, mainly from an upper bound perspective. In orthogonal range reporting, we are to maintain a set of points in d-dimensional space, while supporting efficient retrieval of all points inside an axis-aligned query rectangle. This is one of the most fundamental problems in computational geometry and spatial data bases. The most important results in these papers are: In [5], we present the first optimal data structure for 3-dimensional orthogonal range reporting in the pointer machine model. In contrast, the 2-dimensional variant has been completely solved for more than two decades. In [32], we present the current best data structures for orthogonal range reporting in the word-RAM in all dimensions $d \geq 2$. Similarly, the paper [4] gives the best known upper bound for orthogonal range reporting in the I/O-model for dimensions $d \geq 3$. Finally, the paper [6] presents the fastest pointer machine data structure in dimensions $d \geq 4$, as well as the highest query time lower bound proved for such data structures.

The papers [28, 46, 49, 53, 54] focus on proving lower bounds in the cell probe model. Of particular interest are the latter two papers [53, 54], which present new techniques for proving lower bounds. More specifically, the paper [54] presents a new technique for proving lower bounds for static data structures, yielding the highest static lower bounds to date. Similarly, [53] presents a new technique for proving dynamic lower bounds. The lower bounds obtained using the new technique are almost quadratically higher than what could be obtained using previous techniques. This paper received both the STOC'12 (44th ACM Symposium on Theory of Computing) Best Paper Award and the Best Student Paper Award (the Danny Lewin Award).

The two papers [52, 55] study range searching in the group model. Here all input objects to a range searching problem are assigned a weight from a commutative group and the goal is to compute the group sum of the weights assigned to those input objects intersecting a given query range. The two papers [52, 55] demonstrate tight connections between range searching in the group model and combinatorial discrepancy and range reporting in the pointer machine. These two connections allow us to translate decades of lower bound results in discrepancy theory and range reporting directly to group model lower bounds. The lower bounds we obtain are polynomial of magnitude, whereas the highest previous lower bound in the group model was only poly-logarithmic. The paper [52] received the FOCS'11 (52nd IEEE Symposium on Foundations of Computer Science) Best Student Paper Award (the Machtey Award).

The remaining papers [29, 14, 2, 31, 26, 3] study various algorithmic and data structural questions. Of these, I want to high-light the paper [26] in which we study the basic problem of sampling from a discrete probability distribution. Here we improve on the space usage of a long-standing classic solution, dating all the way back to 1974. Furthermore, our solution is extremely simple and practically efficient. Finally, we also present lower bounds showing that our new solution is optimal.

As mentioned, this dissertation focuses on lower bounds. I have not included all my lower bound results, but have instead chosen to include only results from the six papers [4, 6, 52, 53, 54, 55]. These papers all introduce new techniques, with which we prove lower bounds previously out of reach. I have focused on new techniques because I believe new techniques and tools have a much larger impact than a number of concrete lower bounds proved using previously known techniques. Chapter 2 studies the cell probe model and is based on [53, 54]. The papers [52, 55] form the basis for Chapter 3, which studies range searching in the group model. Finally, Chapter 4 studies the pointer machine and the I/O-model. The results therein are part of the two papers [4, 6].

Contents

Chapter 1

Introduction

The theory of data structures is concerned with representing data in main memory or on disk, such that queries about the data can be answered efficiently. We typically think of data as a set of input objects. Some common examples of queries include *"How many of the input objects satisfies requirement x?"*, *"Which of the input objects is smallest with respect to property y?"* and *"Give me all input objects satisfying requirement z"*. Data structure problems naturally divide into two categories, *static* problems, where the input data is given once and for all, and *dynamic* problems, where we must support updates to the input data. For static problems, efficiency is most commonly measured in terms of query time and space usage, i.e. the time it takes to answer a query and the amount of main memory or disk space used to maintain the data structure. For dynamic data structures, we are also interested in the update time. Efficient data structures is one of the bearing pillars of our modern information society. Classic data structures such as *hash tables*, *priority queues*, *red-black trees* and *linked lists* are part of any respectable standard library and are used at the core of almost all real-world applications. The use of hash tables and linked lists are even built into the syntax of the popular programming language Python. Another example where data structures plays a key role is in relational data bases. Such data bases are in effect just collections of data structures (indices) constructed on the different attributes of the stored records. Common data structures in this regime are known as *B-trees*, *kd-trees* and *range trees.*

After decades of research, we now have efficient solutions, called *upper bounds*, for most of the basic data structure problems. However, since data structures are used extensively everywhere, even a ten percent improvement in the performance of any of the key data structures mentioned above would have a huge impact. Thus researches still strive to improve the known solutions. But when does it end? Can we always improve the solutions we have? Or is there some limit to how efficiently a data structure problem can be solved? This is precisely the question addressed by *lower bounds.* Lower bounds are mathematical functions putting a limit on the performance of data structures. More concretely, a lower bound is a statement of the following flavor: *"Any data structure solving problem x using y bits of space, must use at least $f(y)$ CPU cycles to answer a query"*. Observe the crucial difference between upper bounds and lower bounds: Upper bounds show the *existence* of an efficient solution, while lower bounds must say something about *all* possible data structures, even those no one has thought of yet. In this light, it should not come as a surprise that proving *some* non-trivial lower bound is significantly harder than obtaining *some* non-trivial upper bound. The natural goal when proving lower bounds is of course to show that the data structure upper bounds we know are optimal, i.e. there cannot possibly exist a faster data structure than what we already have.

So how are lower bounds proved? For this, one needs a *model* of what a data structure can do. However, there is a tradeoff to be made: The more realistic the model, the harder it is to prove lower bounds. As a consequence, numerous models of computation have emerged over the years, in one way or another trying to balance the meaningfulness of lower bounds and the burden of proving them. The most prominent models include the semi-group and group model, the pointer machine model, the I/O-model and the cell probe model. The amount of success the community has had in proving lower bounds in these models vary widely. In the more restrictive pointer machine model and semi-group model, very high and near-tight lower bounds have been known for decades. This is certainly not the case for the group model and the cell probe model.

In this dissertation, we present new techniques and frameworks for proving data structure lower bounds in the pointer machine model, the I/O-model, the group model and the cell probe model. In all cases, we push the frontiers further by obtaining lower bounds higher than what could be proved using the previously known techniques. In the following section, we formally define the above models and present the concrete results we obtain.

1.1 Models of Computation

As mentioned, models of computation are typically designed with two conflicting goals in mind: Predictive power for the actual performance when implemented on a real computer, and secondly, with the aim of alleviating the task of proving lower bounds. The computational model most geared towards designing new upper bounds is called the unit cost word-RAM. This model resembles very closely what can be implemented in modern imperative programming languages such as C++ and Java. It allows various natural operations on machine words in constant time, including for instance integer addition, multiplication, division, etc. We give the details of this model in Section 1.2. When proving lower bounds for word-RAM data structures, we typically do not want the lower bounds to be dependent on the concrete set of machine instructions available. Therefore, we use to the cell probe model when proving lower bounds. This model abstracts away the instruction set and allows for *arbitrary* instructions on machine words in constant time. This model is the least restrictive of all the lower bound models proposed in the literature and consequently the lower bounds proved in this model are the most generally applicable we have. Unfortunately this generality has come at a high cost: Prior to our work, the highest lower bound proved for any data structure problem is just logarithmic. This is far from the conjectured lower bound for many natural data structure problems. We describe the cell probe model and our results in this model in Section 1.3.

Given the rather limited success researches have had in proving cell probe lower bounds, more restrictive models have been designed with the hope of obtaining higher and still meaningful lower bounds. The models we consider in this dissertation have been designed with a particular class of problems in mind, namely *range searching problems*. Range searching is one of the most fundamental and well-studied topics in the fields of computational geometry and spatial databases. The input to a range searching problem consists of a set of n geometric objects, most typically points in d-dimensional space, and the goal is to preprocess the input into a data structure, such that given a query range, one can efficiently aggregate information about the input objects intersecting the query range. Some of the most typical types of query ranges are axis-aligned rectangles, halfspaces, simplices and balls. The first class of range searching problems we consider is known as range searching in the semi-group and the group model. In the (semi-)group model, each input object to a range searching problem is assigned a weight from a (semi-)group and the goal is to compute the (semi-)group sum of the weights assigned to the input objects intersecting a query range. In Section 1.4 we further motivate the semi-group and group-model and finally present our results in these models.

The second class of range searching problems that we consider is *range reporting*. Here the goal is to report the set of input objects intersecting a query range. Lower bounds for reporting problems have typically been proved in the pointer machine model. Essentially, the pointer machine model is a restriction of the word-RAM in which navigation is only allowed by following pointers, i.e. we disallow random accesses. While this may seem like a severe limitation, it turns out that most known range reporting data structures are in fact pointer-based. Thus no significant separation has been shown between the two models from an upper bound perspective. We give a more formal definition of the pointer machine model in Section 1.5, where we also present our results in that model.

Finally, we study the case where the input data set is too large to fit in the main memory of the machine. For such data sets, the performance bottleneck is no longer the number of CPU instructions executed, but instead the number of disk accesses needed to answer a query. Thus the performance of a data structure when analysed in the classic models, such as the word-RAM and the pointer machine, is no longer a good predictor for the actual performance of the data structure when used on a huge data set on a real machine. To make more accurate performance predictions when dealing with large data sets, Aggarwal and Vitter [8] designed the I/O-model in which data structures are analysed in terms of the number of disk accesses needed to answer queries, rather than the number of CPU instructions. At a high level, this model considers a machine with a bounded memory of M words and an infinite sized disk partitioned into blocks of B consecutive words each. Computation can only take place on data residing in main memory, and data is moved to and from disk in blocks. The goal is to minimize the number of block moves, called I/Os. We further discuss this model and the results we obtain therein in Section 1.6.

1.2 The Word-RAM

In the unit cost word-RAM model, a data structure is represented in a random access memory, partitioned into words of w bits each. The memory words have integer addresses amongst $[2^w] = \{0, \ldots, 2^w - 1\}$, i.e. we assume a word has enough bits to store the address of any other memory word (it can store a pointer). Retrieving a memory word is assumed to take constant time. Also, any word operation that can be found in modern programming languages such as C++ or Java is supported in constant time. This includes e.g. integer addition, subtraction, multiplication and division, bitwise operations such as AND, OR and XOR on two words and left/right shifts. Standard comparisons on words are also supported in constant time, e.g. checking for equality, greater than or smaller than (when interpreted as integers), etc. Typically, we assume $w = \Omega(\lg n)$ where n is the size of the input. This allows for storing an index into the data structure in a single word.

The query cost is simply defined as the number of instructions needed to answer a query and the space is defined as the largest address used, i.e. the space usage is S words if only addresses amongst $[S] = \{0, \ldots, S - 1\}$ are used.

Dynamic data structures are also required to support updates. Here the update time is defined as the number of instructions needed to apply the desired changes to the data structure.

1.3 The Cell Probe Model

The cell probe model is a less restrictive version of the word-RAM. Again a static data structure consists of a memory of cells, each containing w bits that may represent arbitrary information about the input. The memory cells all have an integer address amongst $[2^w]$ and we say that the data structure uses S cells of space if only cells of addresses $[S]$ are used. Here we also make the common assumption that a cell has enough bits to address the input, i.e. we assume $w = \Omega(\lg n)$, where n is the input size.

When presented with a query, a static data structure reads (probes) a number of cells from the memory, and at the end must announce the answer to the query. The cell probed at each step may be *any deterministic function* of the query and the contents of the previously probed cells, thus all computations on read data are free of charge (i.e. we allow arbitrary instructions). The *query cost* is defined as the number of cells probed when answering a query. Clearly lower bounds in this model also apply to data structures developed in the word-RAM (we count only memory accesses and allow arbitrary instructions).

A dynamic data structure in the cell probe model must also support updates. When presented with an update, a dynamic data structure may both read and write to memory cells. We refer to reading or writing a cell jointly as probing the cell. Again, the cells probed and the contents written to cells during an update may be arbitrary functions of the previously probed cells and the update itself. The query cost is defined as for static data structures. The *update cost* is defined as the number of cells probed when performing an update.

Randomization. In this dissertation, we also consider data structures that are randomized. When answering queries, a randomized data structure is given access to a stream of uniform random bits. The cells probed when answering queries are allowed to depend also on this random stream. The *expected query cost* is defined as the maximum over all pairs of a query q and an input I (update sequence U), of the expected number of cells probed when answering q on input I (after processing the updates U). Furthermore, we allow randomized data structures to return an incorrect result when answering queries. We define the *error probability* of a randomized data structure as the maximum over all pairs of an input I (update sequence U) and a query q, of the probability of returning an incorrect result when answering q on input I (after processing the updates U).

By a standard reduction, any randomized data structure with a constant probability of error $\delta < 1/2$ and expected query cost t, can be transformed into a randomized data structure with any other constant error probability $\delta' > 0$ and worst case query cost $O(t)$, see Section 2.1.2. Hence, when stating query cost lower bounds for randomized data structures with a constant error probability $\delta < 1/2$, we omit the concrete error probability and whether the lower bound is for the expected query cost or the worst case query cost.

1.3.1 Previous Results

In the following, we first give a brief overview of the previous techniques and highest lower bounds obtained in the field of static and dynamic cell probe lower bounds.

Static Data Structures. Many outstanding results and techniques were proposed in the years following Yao's introduction of the cell probe model [82]. One of the most notable papers from that time was the paper by Miltersen et al. [66], relating asymmetric communication complexity and static data structures. During the next decade, a large number of results followed from their techniques, mainly related to predecessor search, partial match and other nearest neighbor type problems [9, 64, 18, 19, 58, 12, 78]. Unfortunately these techniques could not distinguish the performance of near-linear space data structures from data structures using polynomial space. Thus for problems where the number of queries is polynomial in the input size, all these results gave no query cost lower bounds beyond $\Omega(1)$, even for linear space data structures (there are at most $n^{O(1)}$ queries, which is trivially solved in polynomial space and constant query time by complete tabulation). This barrier was not overcome until the milestone papers of Pătraşcu and Thorup [73, 74]. The technique they introduced has since then evolved (see e.g. [70]) into an elegant refinement of Miltersen et al.'s reduction from static data structures to asymmetric communication complexity, and it has triggered a renewed focus on static lower bounds, see e.g. [69, 70, 79, 46, 49]. Their results pushed the highest achieved query lower bound to $\Omega(\lg d/\lg\lg d)$ for data structures using $n \lg^{O(1)} d$ cells of space, where d is the number of different queries to the data structure problem. This lower bound was proved also for randomized data structures with any constant error probability $\delta < 1/2$. Their technique thus provided the first non-trivial lower bounds when $d = n^{O(1)}$, and their lower bounds remained the highest achieved prior to our work (Miltersen [63] showed by counting arguments that there must *exist* problems that need either query time $\Omega(n/w)$ or space $\Omega(d)$. Finding an explicit such problem is the main challenge). We note that the technique of Pătraşcu and Thorup cannot be used to prove lower bounds beyond $\Omega(\lg d/\lg\lg d)$, even for linear space deterministic data structures, see Section 2.1.

Recently, Panigrahy et al. [68] presented another technique for proving static cell probe lower bounds. Their technique is based on sampling cells of the data structure instead of relying on communication complexity. Using this technique, they reproved the bounds of Pătraşcu and Thorup [74] for various nearest neighbor search problems. We note that the idea of sampling cells has appeared before in the world of succinct data structures, see e.g. the papers by Gál and Miltersen [44] and Golynski [45].

Dynamic Data Structures. Lower bounds for dynamic data structures have almost exclusively been proved by appealing to the seminal chronogram technique of Fredman and Saks [42]. The basic idea is to divide a sequence of n updates into *epochs* of exponentially decreasing size. From these epochs, one partitions the cells of a data structure into subsets, one for each epoch i. The subset associated to an epoch i contains the cells that where last updated when processing the updates of epoch i. Lower bounds now follow by arguing that to answer a query after the n updates, one has to probe $\Omega(1)$ cells associated to each epoch. For technical reasons (see Section 2.2.1), the epoch sizes have to decrease by a factor of at least wt_u, where t_u is the worst case update cost of the data structure. Thus one obtains lower bounds no higher than $t_q = \Omega(\lg n/\lg(wt_u))$, where t_q is the expected query cost of the data structure. This bound peaks at $\max\{t_u, t_q\} = \Omega(\lg n/\lg\lg n)$ for any poly-logarithmic cell size. We note that by minor modifications of these ideas, the same bound can be achieved when t_u is the amortized update cost. We also mention one of the most notable applications of the chronogram technique, due to Alstrup et al. [11]. In their paper, they proved a lower bound of $t_q = \Omega(\lg n/\lg(wt_u))$ for the *marked ancestor* problem. Here the input is a rooted tree and an update either marks or unmarks a node of the tree. A query asks whether a given node has a marked ancestor or not. Many natural data structure problems with a range searching flavor to them easily reduce to the marked ancestor problem and hence the lower bound also applies to those problems.

The bounds of Fredman and Saks remained the highest achieved until the breakthrough results of Pătraşcu and Demaine [72]. In their paper, they extended upon the ideas of Fredman and Saks to give a tight lower bound for the *partial sums* problem. In the partial sums problem, the input consists of an array of n entries, each storing an integer. An update changes the value stored at an entry and a query asks to return the sum of the integers in the subarray between two given indices i and j. Their lower bound states that $t_q \lg(t_u/t_q) = \Omega(\lg n)$ and $t_u \lg(t_q/t_u) = \Omega(\lg n)$ when the integers have $\Omega(w)$

bits, which in particular implies $\max\{t_q, t_u\} = \Omega(\lg n)$. We note that they also obtain tight lower bounds in the regime of smaller integers. The bounds hold even when allowed amortization and randomization. For the most natural cell size of $w = \Theta(\lg n)$, this was the highest achieved lower bound before our work.

The two above techniques both lead to smooth tradeoff curves between update time and query time. While this behaviour is correct for the partial sums problem, there are many examples where this is certainly not the case. Pătraşcu and Thorup [75] recently presented a new extension of the chronogram technique, which can prove strong threshold lower bounds. In particular they showed that if a data structure for maintaining the connectivity of a graph under edge insertions and deletions has amortized update time just $o(\lg n)$, then the query time explodes to $n^{1-o(1)}$.

In the search for super-logarithmic lower bounds, Pătraşcu introduced a dynamic set-disjointness problem named the multiphase problem [71]. Based on a widely believed conjecture about the hardness of 3-SUM, Pătraşcu first reduced 3-SUM to the multiphase problem and then gave a series of reductions to different dynamic data structure problems, implying polynomial lower bounds under the 3-SUM conjecture. Proving an unconditional lower bound for the multiphase problem seems to be the most promising direction for obtaining lower bounds of polynomial magnitude. In this context, the paper [33] presents many interesting results related to Pătraşcu's multiphase problem.

Finally, we mention that Pătraşcu [69] presented a technique capable of proving a lower bound of $\max\{t_q, t_u\} = \Omega((\lg n/ \lg\lg n)^2)$ for *dynamic weighted orthogonal range counting*, but only when the weights are $\lg^{2+\varepsilon} n$-bit integers where $\varepsilon > 0$ is an arbitrarily small constant. In this problem, we are to support insertions of two-dimensional points with coordinates on the grid $[n] \times [n]$, where each point is assigned an integer weight. The goal is to return the sum of the weights assigned to the points lying inside an axis-aligned query rectangle. For range counting with δ-bit weights, it is most natural to assume that the cells have enough bits to store the weights, since otherwise one immediately obtains an update time lower bound of δ/w just for writing down the change. Hence Pătraşcu's proof is meaningful only in the case of $w = \lg^{2+\varepsilon} n$ (as he also notes). Thus the magnitude of the lower bound compared to the number of bits, δ, needed to describe an update operation (or a query), remains below $\Omega(\delta)$. This bound holds when t_u is the worst case update time and t_q the expected query time of a data structure. Pătraşcu mentioned that it was an important open problem to prove a similar bound for range counting without weights.

To summarize, the highest lower bound obtained so far on $\max\{t_q, t_u\}$ remains just $\Omega(\lg n)$ in the most natural setting of cell size $w = \Theta(\lg n)$. If we allow cell size $w = \lg^{2+\varepsilon} n$, then a lower bound of $\max\{t_q, t_u\} = \Omega((\lg n/ \lg\lg n)^2)$ is the highest proved.

1.3.2 Our Contributions

In Chapter 2, we present new techniques for proving cell probe lower bounds for both static and dynamic data structures. Our new techniques in both cases yield the highest lower bounds to date and hence opens a new range of problems for which we may hope to prove tight lower bounds.

In the static case, we further investigate the cell sampling technique proposed by Panigrahy et al. [68]. Surprisingly, we show that with a small modification, the technique is more powerful than the communication complexity framework of Pătraşcu and Thorup [74]. More specifically, we apply the technique to the static *polynomial evaluation* problem. In this problem, we are given an n-degree polynomial with coefficients from a finite field $\mathbf{F}$. The goal is to evaluate the polynomial at a given query element $x \in \mathbf{F}$. For this problem, we obtain a query cost lower bound of $\Omega(\lg |\mathbf{F}| / \lg(Sw/n \lg |\mathbf{F}|))$ when $|\mathbf{F}|$ is at least $n^{1+\varepsilon}$ for an arbitrarily small constant $\varepsilon > 0$. This lower bound holds for randomized data structure with any constant error probability $\delta < 1/2$. For linear space data structures (i.e. $S = O(n \lg |\mathbf{F}|/w)$), this bound simplifies to $\Omega(\lg |\mathbf{F}|)$. This is the highest static cell probe lower bound to date, and is a $\lg \lg |\mathbf{F}|$ factor larger than what can possibly be achieved using the communication framework. We discuss the previous work on the concrete problem of polynomial evaluation in Section 2.1.2.

For dynamic data structures, we first consider a dynamic variant of the polynomial evaluation problem (see Section 2.2.2). We have chosen to demonstrate our new technique for this (perhaps slightly artificial) problem because it allows for a very clean proof. The lower bound we obtain is of the form $t_q = \Omega(\lg |\mathbf{F}| \lg n / \lg(wt_u / \lg |\mathbf{F}|) \lg(wt_u))$ when the field has size at least $\Omega(n^2)$. Here t_u is the worst case update time and t_q is the query cost of any randomized data structure with a constant error probability $\delta < 1/2$. In the natural case of $|\mathbf{F}| = n^{2+O(1)}$, $w = \Theta(\lg n)$ and poly-logarithmic update time t_u, the lower bound simplifies to $t_q = \Omega((\lg n / \lg \lg n)^2)$. This is almost a quadratic improvement over the highest previous lower bound of Pătraşcu and Demaine [72]. Secondly, we also apply our technique to a more natural problem, namely dynamic weighted orthogonal range counting in two-dimensional space, where the weights are $\Theta(\lg n)$-bit integers. For this problem, we obtain a lower bound of $t_q = \Omega((\lg n / \lg(wt_u))^2)$. This gives a partial answer to the open problem posed by Pătraşcu by reducing the requirement of the magnitude of weights from $\lg^{2+\varepsilon} n$ to just logarithmic. Finally, the lower bound is also tight for any update time that is at least $\lg^{2+\varepsilon} n$, hence deepening our understanding of one of the most fundamental range searching problems.

1.4 The Group Model

In the group model, each input object to a range searching problem is assigned a weight from a commutative group, and the goal is to preprocess the input into a data structure, consisting of precomputed group elements and auxiliary data, such that given a query range, one can efficiently compute the group sum of the weights assigned to the input objects intersecting the query range. The data structure answers queries by adding and subtracting a subset of the precomputed group elements (in a group, each element has an inverse element, thus we have subtraction) to yield the answer to the query. In addition to answering queries, we require that a data structure supports updating the weights of the input objects. The related semi-group model is defined identically, except that weights come from a semi-group and hence data structures cannot subtract (in a semi-group, we are not guaranteed the existence of an inverse element).

Motivation. The true power of the group model and semi-group model lies in the abstraction. Having designed an efficient (semi-)group model data structure (which does not exploit properties of the particular (semi-)group in question), one immediately has a data structure for many other natural range searching problems. For instance, a range counting data structure can be obtained from a group model data structure by plugging in the group $(\mathbb{Z}, +)$, i.e. integers with standard addition and subtraction, and then assigning each input object a weight of 1. Emptiness queries, i.e. returning whether a query range is empty, can be solved by choosing the semi-group $(\{0, 1\}, \vee)$ and assigning each input object the weight 1. Similarly, finding the right-most point inside a query region could be done using the semi-group $(\mathbb{R}, \max)$ and assigning each point a weight corresponding to its x-coordinate. As a technical remark, note that any semi-group data structure is also a group model data structure, but the converse is not true. Hence proving lower bounds in the group model is at least as hard as in the semi-group model (and seemingly much harder as we shall see).

Since the group model was first introduced there has been two slightly different definitions of data structures, one a bit less restrictive than the other. The most restrictive type of data structure is known in the literature as *oblivious*, while the other type has not received a formal name. To avoid confusion, we have chosen to name this other type of data structure *weakly oblivious*. In the following we review both definitions, starting with the least restrictive:

Weakly Oblivious Data Structures. A *weakly oblivious* data structure in the group model, is a dynamic data structure with no understanding of the particular group in question, i.e. it can only access and manipulate weights through black-box addition and subtraction [42]. Thus from the data structure's point of view, each precomputed group element is just a linear combination over the weights (and possibly previous weights) assigned to the input objects. When answering a query, such a data structure adds and subtracts a subset of these linear combinations to finally yield the linear combination summing exactly the weights currently assigned to the input objects intersecting the query range. Deciding which precomputed elements to add an subtract is determined from the auxiliary data. When given an update request, the data structure may change the auxiliary data, delete some of the stored group elements, and also create new group elements to store by adding and subtracting both previously stored group elements and the newly assigned weight.

The query time of such a data structure is defined as the number of precomputed group elements used when answering a query, and the update time is defined as the number of created and deleted group elements on an update request. Thus all access to the auxiliary data is considered free of charge, and hence deciding which precomputed group elements to add, subtract, create and delete is also for free.

We note that if we did not require the data structure to have no knowledge of the group in question, then range searching over any finite group would be trivial: The data structure could start by storing each element in the group and then encode the weights assigned to the input points in the auxiliary data. Thus when given a query, it can compute the answer by examining the auxiliary data and then returning the corresponding stored group element using no group operations at all. Similarly, updates are for free since we only need to change to auxiliary data. The group model is thus incomparable to the cell probe model.

Oblivious Data Structures. The second, and slightly more restrictive definition of data structures, was given by Fredman [43]. Again data structures are considered to have no knowledge of the group, and queries are still answered by adding and subtracting precomputed linear combinations over the weights assigned to the input points. The update operations are however more constrained: an update of the weight assigned to an input object p, is supported simply by re-evaluating every precomputed group element for which the weight of p occurs with non-zero coefficient in the corresponding linear combination. Every stored group element thus corresponds to a linear combination over the currently assigned weights, and may not include previous weights.

The query time of an oblivious data structure is defined as the number of group elements used when answering a query, and the update time is defined as the number of linear combinations that need to be re-evaluated when updating the weight of an input object. We note that lower bounds proved for weakly oblivious data structures also apply to oblivious data structures. For a more mathematical definition of an oblivious data structure, we refer the reader to Section 3.3.

Given that data structures in the group model have no understanding of the particular group in question, the additional freedom allowed for weakly oblivious data structures might seem artificial. Thus we mention that the main motivating factors for studying weakly oblivious data structures, is that they allow for amortization in the update algorithm and secondly, previous lower bounds proved for weakly oblivious data structures were (somewhat) easily translated to cell probe lower bounds.

1.4.1 Previous Results

In the following, we first review the previous results on lower bounds for range searching problems in the (semi-group and) group model, and then present the best known upper bounds for the two most fundamental range searching problems: orthogonal range searching and halfspace range searching.

In the semi-group model, researchers have been very successful in proving lower bounds. Since data structures cannot subtract, range searching lower bound proofs tend to have a very geometric flavor: If a data structure stores a precomputed semi-group element involving the weight of an input object, then the query algorithm can only use that precomputed semi-group element when answering query ranges that intersects that input object (its weight cannot be cancelled out). Thus semi-group lower bound proofs boils down to arguing that it is hard to “cover” all query ranges with a small collection of subsets of input objects.

Unfortunately we have no such property when allowing subtraction (i.e. the group model). The difficulties encountered when moving from the semi-group to the group model have been recognized as major obstacles for decades, and we believe the following quote by Pătraşcu captures the essence of these difficulties:

“Philosophically speaking, the difference in the type of reasoning behind semi-group lower bounds and group/cell probe lower bounds is parallel to the difference between *understanding geometry* and *understanding computation*. Since we have been vastly more successful at the former, it should not come as a surprise that progress outside the semi-group model has been extremely slow [69].”

In 1982, Fredman [43] gave the definition of an oblivious data structure in the group model. He then managed to prove an $\Omega(\lg n)$ lower bound on the average cost per operation in a sequence of n updates and n queries to the *partial sums* problem. In the group model variant of the partial sums problem, the input is an array of n entries, each storing an element from a commutative group, and the goal is to support weight updates and range queries of the form: “What is the group sum of the elements in the subarray from index i through j?”.

The next result on group model lower bounds was due to Fredman and Saks [42], who introduced the celebrated chronogram technique (see Section 2.2.1 for a description of this technique when used in the cell probe model). Using this technique, they again proved lower bounds for the partial sums problem, stating that any dynamic data structure must have an average cost per operation of $\Omega(\lg n/\lg\lg n)$ over a sequence of n updates and n queries [42]. While the lower bound is weaker than the earlier lower bound of Fredman, it holds also for weakly oblivious data structures.

Chazelle [35, 36] later proved lower bounds for *offline* range searching in the group model. He first proved a lower bound of $\Omega(n \lg n)$ for offline two-dimensional *halfspace range searching* [36]. The input to the offline halfspace range searching problem is a set of n query halfspaces and n input points, each assigned a weight from a commutative group, and the goal is to compute for every query halfspace, the group sum of the weights assigned to the points contained therein. In [35] he considered offline two-dimensional orthogonal range searching and proved a lower bound of $\Omega(n \lg\lg n)$. Here the input is n axis-aligned query rectangles and n points, each assigned a weight. Again, the goal is to answer all the queries on the given input points. Both these lower bounds were established using a general theorem for proving lower bounds for offline range searching in the group model: Letting A denote the *incidence matrix* corresponding to the input set of points and queries (i.e. the matrix with one row for each query R_i and one column for each point p_j, such that entry $a_{i,j}$ is 1 if $p_j \in R_i$ and it is 0 otherwise), Chazelle showed that for any $1 \leq k \leq n$, if λ_k denotes the k'th largest eigenvalue of A, then the offline problem requires $\Omega(k \cdot \lg(\lambda_k))$ group operations [36]. Thus proving offline group model lower bounds was reduced to constructing input and query sets where the corresponding incidence matrix has large eigenvalues.

The next big result was due to Pătraşcu and Demaine [72], who showed an $\Omega(\lg n)$ lower bound on the average cost per operation over a sequence of n updates and n queries to the partial sums problem. While matching the early results of Fredman, this bound also applies to weakly oblivious data structures.

Finally, Pătraşcu [69] proved an $\Omega(\lg n/\lg(\lg n + S/n))$ lower bound for the query time of *static* data structures for two-dimensional orthogonal range searching. Here S is the space used by the data structure in number of precomputed group sums. Using an elegant extension of the chronogram technique, this provided the highest lower bound to date for any dynamic range searching problem in the group model, namely $t_q = \Omega((\lg n/\lg(\lg n + t_u))^2)$, where t_q is the query time and t_u is the update time. This lower bound applies to weakly oblivious data structures for two-dimensional orthogonal range searching.

Given that it has now been three decades since the model was defined, we believe it is fair to say that progress indeed has been extremely slow outside the semi-group model, with the highest lower bound to date not exceeding $\Omega((\lg n/\lg\lg n)^2)$ per operation.

On the upper bound side, there is no separation between what has been achieved for oblivious and weakly oblivious data structures. Thus, all the bounds we mention in the following hold for both types of data structures.

The best results for d-dimensional orthogonal range searching in the group model is achieved through the classic data structures known as *range trees* [24]. These data structures provide a solution with $t_q = t_u = O(\lg^d n)$. From the above lower bounds, these data structures are seen to be optimal in 1-d, and to have a query time within a $\lg^{O(1)}\lg n$ factor from optimal in 2-d. Unfortunately it is not even known from a lower bound perspective whether the query and update time must grow with dimension.

For halfspace range searching, one can use Chan's results on *partition trees* to give data structures with $t_u = O(\lg\lg n)$ and $t_q = O(n^{1-1/d})$ [30], and with some extra work, one can extend the results in [61] to achieve a tradeoff between query time and update time of $t_q = \tilde{O}(n^{1-1/d}/t_u^{1/d})$, for any $t_u = \Omega(\lg n)$. Here $\tilde{O}(\cdot)$ hides poly-logarithmic factors. Thus the highest known lower bound for any explicit problem is exponentially far from the best known upper for halfspace range searching.

1.4.2 Our Contributions

In Chapter 3, we present two new techniques for proving lower bounds for oblivious data structures. The first technique establishes an intimate connection between dynamic range searching in the group model and *combinatorial discrepancy*. The second technique demonstrates a connection to *range reporting* in the pointer machine model. These two connections allow us to reuse decades of research in combinatorial discrepancy and range reporting to immediately obtain a whole range of exceptionally high and near-tight lower bounds for all of the basic range searching problems. We have stated a few of the obtained lower bounds in the following, where t_u is the worst case update time and t_q the worst case query time of an oblivious data structure:

- For d-dimensional halfspace range searching, we get a lower bound of $t_u t_q = \Omega(n^{1-1/d})$. This comes within a $\lg\lg n$ factor of Chan's recent upper bound [30].
- For orthogonal range searching, we get a lower bound of $t_u t_q = \Omega(\lg^{d-1} n)$.
- For ball range searching, we get a lower bound of $t_u t_q = \Omega(n^{1-1/d})$.

We note that these lower bounds need a requirement of *bounded coefficients* which was not needed for the previous lower bounds. This requirement is however satisfied by all known upper bounds. We discuss this requirement further in Chapter 3, where we also present all the other lower bounds that follow from our techniques.

The previous highest lower bound for any explicit problem stated that $t_q = \Omega((\lg n/\lg(\lg n + t_u))^2)$, thus our lower bounds are exponentially higher than what has been achieved before. Our result also has implications for the field of combinatorial discrepancy. Using textbook range searching solutions, we improve on the best known discrepancy upper bound for axis-aligned rectangles in all dimensions $d \geq 3$, see Chapter 3 for details.

1.5 The Pointer Machine Model

The pointer machine model was introduced in 1979 by Tarjan [81]. In this model, a range reporting data structure is represented by a directed graph. Each node of the graph may store either an input object or some auxiliary data. The nodes have constant out-degrees and one node is designated as the root. When answering a query, the data structure starts by reading the root node. The data structure then examines the contents of that node and if the node stores an input object, the data structure may choose to report that object if it intersects the query range. Following that, it either terminates or selects an edge leaving the root and retrieves the node pointed to by that edge. This process continues, where at each step, the data structure selects an edge leaving one of the previously seen nodes and retrieves the node pointed to by that edge. When the process terminates, we require that all input objects that intersects the query range have been reported, i.e. each reported input object must be stored in at least one of the explored nodes. Thus a data structure in the pointer machine model is a data structure where all memory accesses are through pointers and random accesses are disallowed. Also, input objects have to be stored *indivisibly* in order for them to be reported and hence standard compression techniques used in the word-RAM cannot be applied.

The space of a pointer machine data structure is defined as the number of nodes in the corresponding graph, and the query time is the number of nodes explored when answering a query. Note that when proving lower bounds, we make no restriction on how the data structure determines which nodes to explore; We simply assume it knows the entire graph and thus non-deterministically can choose the best subgraph to explore.

While the pointer machine model is somewhat constrained compared to the popular word-RAM model (see Section 1.2), there are several motivating factors for studying the complexity of range reporting in this model. First and foremost, we can prove polynomially high and often very tight lower bounds in this model. This stands in sharp contrast to the highest query time lower bound of $\Omega(\lg n)$ for any static data structure problem in the word-RAM (or cell probe model). Additionally, most word-RAM range reporting upper bounds are really pointer-based or can easily be implemented without random accesses with a small overhead, typically at most an $O(\lg n)$ multiplicative cost in the query time and/or space. Thus pointer machine lower bounds indeed shed much light on the complexity of range reporting problems.

1.5.1 Previous Results

With only a few exceptions (see [1, 6]), lower bounds for range reporting in the pointer machine model have all been proved by appealing to a theorem initially due to Chazelle [34] and later refined by Chazelle and Rosenberg [40]. First, let P be a set of input objects to a range searching problem and let $\mathcal{R}$ be a set of query ranges. Then we say that $\mathcal{R}$ is (t, h)-*favorable* if

1. $|R \cap P| \geq t$ for all $R \in \mathcal{R}$.
2. $|R_1 \cap R_2 \cap \cdots \cap R_h \cap P| = O(1)$ for all sets of h different queries $R_1, \ldots, R_h \in \mathcal{R}$.

Letting k denote the output size of a query, Chazelle and Rosenberg proved that

Theorem 1.1 (Chazelle and Rosenberg [40]). *Let P be a set of n input objects to a range searching problem and $\mathcal{R}$ a set of m query ranges. If $\mathcal{R}$ is $(\Omega(t_q), h)$-favorable, then any pointer machine data structure for P with query time $t_q + O(k)$ must use space $\Omega(mt_q/h)$.*

This theorem completely reduces the task of proving pointer machine range reporting lower bounds to a geometric problem of constructing a hard input and query set. As we shall see, it is possible to prove lower bounds of polynomial magnitude using this technique. Thus, in contrast to the cell probe and group model setting, the main objective for research in pointer machine lower bounds is not so much to develop techniques that can prove higher lower bounds, but instead to prove tight lower bounds for the fundamental problems. Hence we have summarized the highest previous lower bounds for the three most fundamental range reporting problems in the following:

Orthogonal Range Reporting. In this problem, the input consists of n points in d-dimensional space and the goal is to report all points contained in an axis-aligned query rectangle. Already in 1990, Chazelle [34] proved an $\Omega(n(\lg n/\lg t_q)^{d-1})$ space lower bound for this problem. Here $t_q + O(k)$ is the query time of the data structure, where k is the number of points reported. The lower bound is known to be tight for $t_q = \Omega(\lg n(\lg n/\lg\lg n)^{d-3})$ [5].

While the lower bound of Chazelle focuses on the space usage, together with Afshani and Arge [5], we recently proved a lower bound focusing on the query time. Our lower bound states that $t_q = \Omega((\lg n/\lg(S/n))^{\lfloor d/2 \rfloor - 1})$ where S denotes the space usage. While this is known not to be tight (we improve on it in Chapter 4), it was the first lower bound to show that the query time has to increase beyond $\Omega(\lg n + k)$ for constant d and near-linear space usage ($n \lg^{O(1)} n$ space). We note that the fastest query time obtained for $d \geq 4$ and $n \lg^{O(1)} n$ space usage is $O(\lg n (\lg n/\lg\lg n)^{d-4+1/(d-2)} + k)$ [6], thus the true complexity of the problem remains a mystery.

Simplex Range Reporting. In simplex range reporting, the input again consists of a set of n points in d-dimensional space and query regions are simplices (the generalization of triangles to higher dimensions). This problem is fundamental since range reporting with any polytope as query range can be solved by first triangulating the polytope and then querying a simplex range reporting data structure with each simplex in the triangulation. The first lower bound for this problem was due to Chazelle and Rosenberg [40] who showed that any simplex range reporting data structure must use space $\Omega((n/t_q)^{d-\varepsilon})$ when the query time is $t_q + O(k)$. This comes fairly close to the known upper bounds having space usage $O((n/t_q)^d \lg^{O(1)} n)$ for any $t_q = \Omega(\lg^{d+1} n)$ [62].

Using the new technique we present in Chapter 4, Afshani [1] recently presented a tighter lower bound of $S = \Omega((n/t_q)^d / 2^{O(\sqrt{\lg t_q})})$. In particular for the regime of $t_q = \lg^{O(1)} n$, this lower bound is within poly-logarithmic factors from the best upper bound.

Halfspace Range Reporting. Unfortunately the status for the halfspace range reporting problem is less satisfactory than for orthogonal and simplex range reporting. The best known lower bounds for this problem are obtained by reduction from range reporting in a *slab* in roughly $\sqrt{d}$ dimensions. Not going into details, we merely note that this gives lower bounds of roughly $S = (n/t_q)^{\Omega(\sqrt{d})}$ [1] where the query time is $t_q + O(k)$, whereas the best known upper bounds have space usage $O((n/t_q)^{\lfloor d/2 \rfloor} \lg^{O(1)} n)$ for any $t_q = \Omega(\lg^c n)$ for a sufficiently large constant $c > 0$ depending on the dimension [7].

1.5.2 Our Contributions

In Chapter 4 we present a new technique for proving pointer machine lower bounds. With this technique, we shrink the gap between the upper and lower bound for orthogonal range reporting. More specifically, we obtain a lower bound stating that data structures with query time $t_q + t_k k$ and space usage S must satisfy $t_q = \Omega(\lg n \cdot \lg_h^{\lfloor d/2 \rfloor - 2} n)$ for $d \geq 4$. Here $h = \max\{S/n, 2^{t_k}\}$. This is a $\lg h$ factor higher than our previous bound from Afshani et al. [5]. Perhaps even more interesting, we use the same hard input instance as we used in [5] in combination with the theorem of Chazelle and Rosenberg (Theorem 1.1). Since the lower bound obtained there was a $\lg h$ factor weaker than our new bound, our new technique is more powerful than the technique of Chazelle and Rosenberg (for some problems and inputs at least).

It remains an intriguing open problem whether the true complexity of orthogonal range reporting has a query time that increases with every dimension or only every other dimension.

Finally, as we noted above, Afshani [1] recently used our new technique to improve on the long-standing best lower bound for simplex range reporting. This also lead to the current best lower bounds for halfspace range reporting (still only growing as roughly $\sqrt{d}$ in the exponent).

1.6 The I/O-Model

The I/O-model of Aggarwal and Vitter [8] was designed to more accurately predict the performance of data structures and algorithms when the size of the input data exceeds the main memory size. When this happens, most of the input data resides on a slow secondary storage device (e.g. a magnetic hard disk). The difference in access time between main memory and disk is typically several orders of magnitude. This means that the time spend performing CPU instructions becomes negligible to the time spend accessing data stored on disk and hence a standard algorithmic analysis in the word-RAM or pointer machine model is no longer a good predictor for performance.

The crucial property of most types of secondary storage devices is that, while access time is slow, reading (and writing) large chunks of consecutive memory locations is very fast. Therefore, data is moved between main memory and disk in large chunks of consecutive memory locations. The hope is that the next many memory locations needed by the algorithm or data structure all belong to the same chunk and hence the large transfer time can be amortized away. This property is exactly what one tries to exploit when designing data structures in the I/O-model.

In the I/O-model, a machine consists of a CPU, a main memory of limited size M words and an infinitely sized disk. The disk is partitioned into blocks of B consecutive words each and data is transferred between main memory and disk in blocks. The movement of one block is called an I/O. The CPU can only perform instructions on data stored in main memory and hence a data structure has to decide how to organize data on disk and which blocks to transfer to main memory when answering queries. The performance of a data structure is measured in space usage (number of words) and the number of I/Os needed to answer a query. Note that all computation on data in main memory is free of charge.

1.6.1 Previous Results

There is an enormous body of work on algorithms and data structures in the I/O-model, thus we have chosen to present only the previous work most related to our results, namely work on orthogonal range reporting. For range reporting in the I/O-model, we assume input objects are *indivisible* and that one input objects fits in a machine word. Hence main memory can store M input objects and a disk block can store B input objects. It is interesting that, in contrast to the pointer machine vs. word-RAM setting, no separation has been shown between orthogonal range reporting in the I/O-model with and without the indivisibility requirement. In fact, it is generally believed that most problems (including sorting) cannot be solved more efficiently in the I/O-model when abandoning the indivisibility requirement (see [48] for more discussion and one of the only convincing exceptions).

When designing orthogonal range reporting data structures in the I/O-model, we typically aim at a query cost of the form $O(\lg_B^c n + k/B)$ where k denotes the number of reported objects and $c \geq 1$ is some small constant. Note that in the I/O-model, the optimal term involving k is $O(k/B)$, and not $O(k)$, since B output objects can be written in one I/O. For natural values of B, the difference between $O(k)$ and $O(k/B)$ is so large that an $O(k)$ reporting cost is completely uninteresting. Also, the optimal search cost is $\lg_B n$ (corresponding to the height of a B-ary search tree), at least when the coordinates of the input objects can only be compared. For natural values of B, $\lg_B n$ is rarely more than 3 or 4, thus a $\lg_B n$ search cost significantly improves over the performance of a binary search.

The main tool for proving lower bounds for range reporting data structures in the I/O-model was introduced by Hellerstein et al. [47] and slightly refined by Arge et al. [15]. Hellerstein et al. [47] defined what they called the *indexability model*, which can be thought of as a variant of the I/O-model targeted at proving lower bounds for range reporting problems. Again, input objects are considered *indivisible* and hence to report an input object, the query algorithm must read a block storing that object. The details of the model are as follows:

The Indexability Model [47]. In the indexability model [47] an indexing problem is described by a *workload* $W = (I, Q)$, where I is a set of input objects and Q is a set of subsets of I; the elements of Q are called *queries*. Given a workload W and a block size B, an *indexing scheme* is defined on I by a block assignment function, $\mathbb{B}$, which is a set of B-sized subsets of I. Intuitively, all the input objects in a set $b \in \mathbb{B}$ are stored in one block.

The quality of an indexing scheme is quantified by two parameters: *space* and *query time*. The space is defined as $B|\mathbb{B}|$ (the number of objects stored). If any query in Q is covered by at most $t_q + t_k\lceil |q|/B \rceil$ blocks of $\mathbb{B}$, then the query time is defined as the (t_q, t_k) tuple [15] (this a slight variation on the original definition of access overhead [47]). For any range reporting data structure in the I/O-model (storing input objects indivisibly), an indexing scheme is naturally defined by just looking at the input objects stored in the blocks of the storage medium, hence lower bounds proved in the indexability model immediately gives lower bounds in the I/O-model.

The following *redundancy theorem* relates space and query cost and is the main tool for proving range reporting lower bounds in the I/O-model [15, 47]:

Theorem 1.2 (Refined Redundancy Theorem [15])**.** *For a workload $W = (I, Q)$ where $Q = \{q_1, q_2, \dots, q_m\}$, let $(I, \mathbb{B})$ be an indexing scheme for W with query cost (t_q, t_k) with $t_k \leq \sqrt{B}/8$ such that for any $1 \leq i, j \leq m, i \neq j : |q_i| \geq Bt_q$ and $|q_i \cap q_j| \leq B/(64t_k^2)$. Then the space of $(I, \mathbb{B})$ is at least $\frac{1}{12}\sum_{i=1}^{m} |q_i|$.*

The similarity to the theorem of Chazelle and Rosenberg (Theorem 1.1) is striking: To prove a lower bound, one constructs a set of input objects and queries, such that any query contains sufficiently many input objects, and secondly, any two queries have a sufficiently small intersection. Except for a recent result by Afshani [1], simplex and halfspace range reporting has not received any attention from a lower bound perspective in the I/O-model, hence we only review the results for orthogonal range reporting in the following.

The two-dimensional version of the problem has been completely solved for some time now. In [15], Arge et al. presented a data structure answering queries in optimal $O(\lg_B n + k/B)$ I/Os using $O(n \lg n / \lg \lg_B n)$ space, where k is the number of reported points. They also used the Refined Redundancy Theorem to prove that this space bound is optimal for any query time of the form $\lg_B^{O(1)} n \cdot (1 + k/B)$. However, in three and higher dimensions, the problem remains open. From a lower bound side, Hellerstein et al. [47] proved that any data structure with query time $t_q + t_k k$ must use space $\Omega(n(\lg B / \lg(t_k t_q))^{d-1})$ for $d \geq 3$. Note the $\lg B$ in the numerator and not $\lg n$ as in the two-dimensional case. From an upper bound side, the current best tradeoffs were presented in our paper Afshani et al. [4]. Here a data structure using $O(n(\lg n / \lg \lg_B n)^{d-1})$ space and answering queries in $O(\lg_B n(\lg n / \lg \lg_B n)^{d-2} + k/B)$ I/Os was presented. Alternatively, a data structure using $O(n(\lg n / \lg \lg_B n)^{d})$ space and answering queries in $O(\lg_B n(\lg n / \lg \lg_B n)^{d-3} + k/B)$ was also presented. For $d = 3$, it is believed that the optimal bound is $O(\lg_B n + k/B)$ query cost and $O(n(\lg n / \lg \lg_B n)^2)$ space, i.e. the best query time of the two solutions and the best space of the two. Thus both the lower bound and the upper bounds are believed to be off for all dimensions $d \geq 3$. Also, the query time of the best upper bounds for $d \geq 4$ are not of the form $\lg_B^{O(1)} n + O(k/B)$, but instead pay almost a $\lg n$ penalty per dimension, thus not exploiting the large block size.

1.6.2 Our Contributions

In Chapter 4 we make two steps towards tightening the lower bounds for orthogonal range reporting. First, we show that our new technique for proving lower bounds in the pointer machine (see the discussion in Section 1.5) can be extended to also prove I/O-model lower bounds. This yields a new lower bound for orthogonal range reporting, stating that data structures with query time $t_q + t_k k/B$ and space S, must satisfy $t_q = \Omega(\lg_h^{\lfloor d/2 \rfloor - 1} n)$ for $d \geq 4$. Here $h = \max\{S/n, t_k\}$. The most surprising conclusion from this lower bound is that it is not possible to exploit the large block size to obtain query times of $\lg_B^{O(1)} n + O(k/B)$ in dimensions $d \geq 4$, at least not without paying an expensive B^ε factor in space for a constant $\varepsilon > 0$. Also note that the lower bound decreases as a polynomial in $\lg t_k$ and not as a polynomial in t_k which was the case in the pointer machine (see Section 1.5). We also mention that Afshani's [1] recent lower bounds for simplex range reporting were based on our new technique. Finally, we use the Refined Redundancy Theorem and construct a hard workload to obtain a space lower bound of $\Omega(n(\lg n / \lg(t_q t_k))^{d-1})$ for data structures with query cost $t_q + t_k k/B$ when $d \geq 2$. Thus we replace the $\lg B$ term in the bound of Hellerstein et al. [47] with a seemingly more correct $\lg n$ term.

Chapter 2

The Cell Probe Model

In this chapter, we present our results in the cell probe model. As mentioned in Chapter 1, the cell probe model has been designed such that lower bounds proved in this model applies to data structures developed in the standard word-RAM model. In that light, the cell probe model is really the most natural and appealing model for proving lower bounds. Unfortunately, as we saw in Section 1.3, the highest previous lower bounds are just logarithmic, even for dynamic data structures. This severely restricts the list of natural data structure problems for which we may hope to prove tight lower bounds. Therefore, the main objective for current research in the cell probe model, is to design new techniques that allow us to prove higher lower bounds and thereby expand the list of problems that we may hope to settle the complexity of. This is precisely the focus of this chapter, in which we introduce two new techniques for proving cell probe lower bounds, one for static data structures and one for dynamic data structures.

In Section 2.1, we first present the previous techniques for proving lower bounds for static data structures. We also discuss the limitations of these techniques and finally present our new technique and compare it to the previous approaches. This new technique allow us to obtain static query time lower bounds peaking at $\Omega(\lg d)$, where d is the number of queries to the data structure problem. We conclude the section on static data structures by demonstrating our technique on the concrete problem of polynomial evaluation.

We continue in Section 2.2 by presenting a number of previous techniques for proving dynamic lower bounds. We have chosen to focus on the techniques most closely related to our new technique. We discuss the limitations of these techniques and finally present the key ideas of our new technique, leading to lower bounds of $\Omega((\lg n/\lg\lg n)^2)$ on the maximum of the update and query time. After having presented the techniques we move on to demonstrate our new technique for the dynamic polynomial evaluation problem and the dynamic weighted range counting problem.

2.1 Static Data Structures

In this section, we focus on static data structures. In the static setup, the input data is given once and for all and we must preprocess it into a data structure to support answering queries. The two performance metrics are space and query time (in particular, we allow arbitrary preprocessing time). We start by reviewing previous techniques for proving lower bounds for static data structures. Following that, we introduce our new technique and demonstrate it on the polynomial evaluation problem.

2.1.1 Techniques

In the following, we first review the techniques of Miltersen et al. [66], Pătraşcu and Thorup [73, 74], and of Panigrahy et al. [68]. Finally we introduce our new approach and compare it to the previous ones.

Miltersen et al. [66]. This technique relies on a reduction to an asymmetric communication game. More formally, to prove a lower bound for a data structure problem, we consider a communication game between two players Alice and Bob. Alice is given a query to the data structure problem and Bob an input set. The goal for Alice and Bob is to compute the answer to Alice's query on Bob's input using as little communication as possible. The "asymmetry" lies in Alice having a much smaller input than Bob.

Now assume a data structure exists using S cells of space and with query cost t. This data structure gives rise to a t-round protocol for the communication game: Bob first construct the data structure on his input. Alice then simulates the query algorithm, and for each cell probe, she sends to Bob the address of the desired cell. Bob then replies with the contents of the cell, until all t probes have been simulated, and Alice knows the answer. The data structure solution thus provides a protocol in which Alice sends $t \lg S$ bits and Bob sends tw bits. Cell probe lower bounds now follow by bounding the communication needed by Alice and Bob using tools from communication complexity, see e.g. the excellent book by Kushilevitz and Nisan [51].

Let d be the number of different queries to the data structure problem. Since the communication game is trivial once either of the two players has send his/her entire input to the other player, this technique cannot prove lower bounds beyond $t \lg S = \Omega(\lg d)$, i.e. at most $t = \Omega(\lg d / \lg S)$, which, as mentioned in Section 1.3, degenerates to $\Omega(1)$ in the setting of $d = n^{O(1)}$ where n is the minimum space usage in bits (i.e. 2^n is the number of inputs). The $\lg S$ term clearly shows that the technique does not distinguish between near-linear and polynomial space data structures.

Pătraşcu and Thorup [74]. This technique is essentially a refinement of the above communication game: Instead of Alice receiving one query as input, she receives a set of k queries. The goal is to compute the answer to all k queries on Bob's one input.

Again assume a data structure exists using S cells of space and with query cost t. Bob constructs the data structure on his input, but this time Alice simulates the query algorithm for all k queries in *parallel*, i.e. in each of the t rounds, she sends to Bob the addresses of all the cell probes requested, for a total of $\lg \binom{S}{k} \approx k \lg(S/k)$ bits per round. Bob replies with the requested cells, including addresses and contents for a total of $k(w + \lg S) \leq 2kw$ bits per round (since $\lg S \leq w$). The key point is that $\lg(S/k)$ is much smaller than $\lg S$ for k close to n, i.e. the average cost of specifying a cell is much smaller. Finally, the cell probe lower bounds are obtained by using communication complexity tools [51] to lower bound the communication between Alice and Bob.

Let d be as above, and furthermore let 2^n be the number of different input sets to the data structure problem. By the same argument as earlier, we cannot hope to prove lower bounds beyond $2ktw = \Omega(n)$. This naturally constrains $k = O(n/wt)$. Similarly, we cannot prove anything beyond $kt \lg(S/k) = \Omega(k \lg d)$. This maximizes at $t = \Omega(\lg d / \lg(S/k))$, which by the bound on k cannot exceed $t = \Omega(\lg d / \lg(Swt/n))$. Since S must be at least n/w, this peaks at $t = \Omega(\lg d / \lg \lg d)$ for $S = n \lg^{O(1)} d$. For problems with polynomially many queries, i.e. $d = n^{O(1)}$, the bound becomes $t = \Omega(\lg n / \lg \lg n)$ for space $n \lg^{O(1)} n$.

Panigrahy et al. [68]. This is the technique which we extend upon. In contrast to the two previous, this technique is not based on communication complexity. Instead, lower bounds are proved by showing that a small set of memory cells "solves" many queries, leading to a contradiction. More specifically, consider a data structure problem with 2^n inputs and d different queries and assume for simplicity that d is at least n^2. To obtain a lower bound, we assume a data structure solution exists for this problem, having query time t. Using this data structure, we show how to represent each of the 2^n inputs uniquely in less than n bits when t is too small, i.e. we *encode* the inputs in less than n bits. Obviously this is not possible since we have 2^n distinct inputs and hence t has to be large.

In greater detail, we map each input I to a string of less than n bits by first implementing the claimed data structure on it. This produces a memory representation consisting of S cells of w bits each. For t iterations, $i = 1, \dots, t$, we now carefully pick a set C_i of $n/4tw$ cells in the data structure. We represent I by writing down the cell sets $C_1, \dots, C_t$, including addresses and contents, for a total of $t(w + \lg S)n/4tw \leq n/2 < n$ bits. What remains is to argue that I can be uniquely recovered from this representation. For this step, we consider each of the d possible queries. For a specific query q, we simulate the query algorithm, and for the i'th cell probe, we check whether the requested cell is included in C_i. If so, we have the contents and can continue the simulation. If not, we discard q and continue with the next query. If we simply choose the sets C_i as uniform random sets of $n/4tw$ cells, then the probability that the i'th cell probed when answering q on input I is included in C_i is precisely $n/4Stw$. It follows that the probability that q is not discarded is $(n/4Stw)^t$. By linearity of expectation, we expect to have $d \cdot (n/4Stw)^t$ queries that are not discarded. If $t = o(\lg d / \lg(Swt/n))$, this is $d^{1-o(1)} = n^{1+\Omega(1)}$. The lower bound finally follows by arguing that I can be uniquely recovered from the answer to any $n^{1+\Omega(1)}$ queries, leading to a contradiction and hence implying $t = \Omega(\lg d/\lg(Swt/n))$. Clearly the answer to any set of $n^{1+\Omega(1)}$ queries is not guaranteed to uniquely determine I, but luckily there is some freedom in the choice of the C_i's and a uniform random choice did not suffice in [68]. The intuition however remains the same, i.e. a small set of cells solves a large number of queries.

As with the previous technique, the highest achievable lower bound is $t = \Omega(\lg d/\lg(Swt/n))$, since for higher values of t, the number of queries not discarded reduces to $d \cdot (n/4Stw)^{\omega(\lg d/\lg(Swt/n))} = o(1)$. Note that for linear space, i.e. $S = O(n/w)$, this becomes $t = \Omega(\lg d/\lg t)$. As with the technique of Pătraşcu and Thorup, this is bounded by $t = \Omega(\lg d/\lg\lg d)$.

Our Technique. The key idea leading to our improved lower bound is in fact a simple twist to the technique of Panigrahy et al. [68]. In their approach, the cell sets C_i must have size less than n/wt since we have to write down t such sets and arrive at less than n bits. Thus the t factor is inevitable and hence lower bounds proved using this technique cannot exceed $\Omega(\lg d/\lg t)$, even for linear space ($S = O(n/w)$).

To avoid the t factor, we pick *one* set of cells C, having size $n/4w$. Encoding this set of cells, including addresses and contents, takes less than n bits. We then discard all queries for which *some* probe is outside C. The probability that all cell probes are inside C when C is chosen uniformly at random is $(n/4Sw)^t$. For $t = o(\lg d/\lg(Sw/n))$ this becomes $d^{-o(1)}$, i.e. we expect to have $d^{1-o(1)}$ queries that are not discarded. We can thus prove lower bounds of $t = \Omega(\lg d/\lg(Sw/n))$ which for linear space ($S = O(n/w)$) becomes $t = \Omega(\lg d)$. In some sense, we have parallelized the sampling for all t probes of the queries which leads to the improvement.

2.1.2 Static Polynomial Evaluation

In this section, we prove a cell probe lower bound for static polynomial evaluation. Before descending into the proof, we recall the problem definition. The input to polynomial evaluation is an n-degree polynomial with coefficients drawn from a finite field $\mathbf{F}$, where we assume $|\mathbf{F}| = n^{1+\Omega(1)}$. A query is specified by an element $x \in \mathbf{F}$ and the goal is to evaluate the polynomial at x. We first review the previous work for this problem.

Previous Results. The polynomial evaluation problem has seen a rather large amount of attention, in particular from a lower bound perspective. Miltersen [65] was the first to prove cell probe lower bounds for polynomial evaluation over a finite field $\mathbf{F}$. His lower bound states that $t = \Omega(\lg|\mathbf{F}|/\lg S)$ whenever $|\mathbf{F}|$ is at least $n^{1+\varepsilon}$ for an arbitrarily small constant $\varepsilon > 0$. Here t is the worst case query cost. This lower bound unfortunately degenerates to $t = \Omega(1)$ for $|\mathbf{F}| = n^{O(1)}$. In [44], Gál and Miltersen considered *succinct* data structures for polynomial evaluation. Succinct data structures are data structures that use space very close to the information theoretic minimum required for storing the input (in this case $(n+1)\lg|\mathbf{F}|$ bits). In this setting, they showed that any data structure for polynomial evaluation must satisfy $tr = \Omega(n)$ when $|\mathbf{F}| \geq (1+\varepsilon)n$ for any constant $\varepsilon > 0$. Here t is the worst case query cost and r is the *redundancy*, i.e. the *additive* number of extra bits of space used by the data structure compared to the information theoretic minimum. If data structures are allowed *non-determinism* (i.e. they can guess the right cells to probe), then Yin [83] proved a lower bound matching that of Miltersen [65].

On the upper bound side, Kedlaya and Umans [50] showed that there exists a static word-RAM data structure (and hence cell probe data structure) for polynomial evaluation, having space usage $n^{1+\varepsilon} \lg^{1+o(1)} |\mathbf{F}|$ and worst case query cost $\lg^{O(1)} n \lg^{1+o(1)} |\mathbf{F}|$ for any constant $\varepsilon > 0$.

The Lower Bound Proof.

Our goal is to prove a lower bound of $t = \Omega(\lg |\mathbf{F}| / \lg(Sw/n \lg |\mathbf{F}|))$, where S is the space in number of cells, w the cell size in bits and t is the query time of a randomized data structures with error probability δ, where δ is an arbitrary constant less than 1/2. Note that for linear space, i.e. $S = O(n \lg |\mathbf{F}|/w)$, the lower bound simplifies to $t = \Omega(\lg |\mathbf{F}|)$.

For the proof, we assume the availability of such a randomized data structure. To ease calculations, we first modify this data structure in the following standard way: When presented with a query, we repeat the query algorithm a sufficiently large constant number of times and then return the majority answer. By this procedure, we have effectively obtained a randomized data structure with error probability at most 1/19, while maintaining the same asymptotic expected query cost. Secondly, we modify the data structure by letting it return an arbitrary answer whenever the query algorithm does not terminate within a number of steps that is bounded by a sufficiently large constant times the expected query cost. By Markov's inequality, this yields a randomized data structure with error probability at most 1/18 and worst case query cost $O(t)$.

Finally, by fixing the random bits, this implies the existence of a deterministic data structure with error probability at most 1/18 and worst case query cost $O(t)$, where the error probability of the deterministic data structure is defined as the probability that it returns an incorrect result when answering a uniform random query on a uniform random input polynomial.

We show that such a deterministic data structure must have worst case query cost $\Omega(\lg |\mathbf{F}| / \lg(Sw/n \lg |\mathbf{F}|))$, which completes the proof.

Notation. We let $\mathbf{P}$ denote a random variable giving a uniform random n-degree polynomial with coefficients in the field $\mathbf{F}$, i.e. each of the $n+1$ coefficients of $\mathbf{P}$ is a uniform random element from $\mathbf{F}$. Clearly $H(\mathbf{P}) = (n+1) \lg |\mathbf{F}|$, where $H(\cdot)$ denotes binary entropy.

An Encoding Proof. Assume for contradiction that a deterministic data structure solution for polynomial evaluation over $\mathbf{F}$, using S cells of space, with worst case query cost $t = o(\lg |\mathbf{F}| / \lg(Sw/n \lg |\mathbf{F}|))$ and error probability 1/18 exists. Assume furthermore $|\mathbf{F}| = n^{1+\Omega(1)}$. Under this assumption, we show how to encode $\mathbf{P}$ using less than $H(\mathbf{P}) = (n+1) \lg |\mathbf{F}|$ bits in expectation, a contradiction. Following Panigrahy et al. [68], the high level idea of the encoding procedure is to implement the claimed data structure on $\mathbf{P}$. Letting $D(\mathbf{P})$ denote the set of cells stored by the data structure on input $\mathbf{P}$, we then find a subset of cells that *resolves* a large number of queries which do not err, and hence the cell set reveals much information about $\mathbf{P}$. Here we say that a set of cells $C \subseteq D(\mathbf{P})$ resolves a query q, if the query algorithm probes only cells in C when answering q on input $\mathbf{P}$. If the found set of cells can be described in fewer bits than the resolved queries reveal about $\mathbf{P}$, this gives the contradiction. The following is our main technical result:

Lemma 2.1. *With probability at least* $1/2$ *over the choice of* $\boldsymbol{P}$*, there exists a set of cells* $\boldsymbol{C} \subseteq D(\boldsymbol{P})$ *and an integer* $\boldsymbol{t}^*$*, where* $1 \leq \boldsymbol{t}^* \leq t$*, which satisfy all of the following properties:*

1. $|\boldsymbol{C}| = n \lg |\boldsymbol{F}|/5w$.
2. *Let* $G_{\boldsymbol{t}^*}^{\boldsymbol{C}}(\boldsymbol{P})$ *be the set of queries that succeed on input* $\boldsymbol{P}$ *(*good *queries), where furthermore each query* q *in* $G_{\boldsymbol{t}^*}^{\boldsymbol{C}}(\boldsymbol{P})$ *is resolved by* $\boldsymbol{C}$ *on input* $\boldsymbol{P}$ *and the query algorithm probes exactly* $\boldsymbol{t}^*$ *cells when answering* q *on input* $\boldsymbol{P}$*. Then* $|G_{\boldsymbol{t}^*}^{\boldsymbol{C}}(\boldsymbol{P})| = |\boldsymbol{F}|^{1-o(1)} = n^{1+\Omega(1)}$.
3. *Similarly, let* $B_{\boldsymbol{t}^*}^{\boldsymbol{C}}(\boldsymbol{P})$ *be the set of queries that err on input* $\boldsymbol{P}$ *(*bad *queries), where furthermore each query* q *in* $B_{\boldsymbol{t}^*}^{\boldsymbol{C}}(\boldsymbol{P})$ *is resolved by* $\boldsymbol{C}$ *and the query algorithm probes exactly* $\boldsymbol{t}^*$ *cells when answering* q *on input* $\boldsymbol{P}$*. Then* $|B_{\boldsymbol{t}^*}^{\boldsymbol{C}}(\boldsymbol{P})| \leq |G_{\boldsymbol{t}^*}^{\boldsymbol{C}}(\boldsymbol{P})|/2$.

Before giving the proof of Lemma 2.1, we show how we use it in the encoding and decoding procedures. We first give a high-level interpretation of Lemma 2.1: Examining the lemma, we see that for half of all possible input polynomials, the claimed (too fast) data structure must contain a set of cells $\mathbf{C}$, such that $\mathbf{C}$ can be described in less than $H(\mathbf{P})$ bits, and at the same time, many queries can be answered solely from the contents of the cells in $\mathbf{C}$. Now observe that knowing the answer to an evaluation query provides a point on the polynomial $\mathbf{P}$. Hence from $\mathbf{C}$, we can recover $|G_{\mathbf{t}^*}^{\mathbf{C}}(\mathbf{P})| = n^{1+\Omega(1)}$ points on the polynomial $\mathbf{P}$. Since an n-degree polynomial over a field is uniquely determined from any $n+1$ distinct points on the polynomial, it follows that (ignoring the queries that err for now) $\mathbf{P}$ is uniquely determined from $\mathbf{C}$, which gives a contradiction since $\mathbf{C}$ can be described in fewer than $H(\mathbf{P})$ bits. The remaining parts of the lemma ensure that we can recover $\mathbf{P}$ even when facing a number of queries that err. The details of this will become apparent in the encoding and decoding procedures below.

We note that to prove Lemma 2.1, we have to extend on the ideas in Panigrahy et al. [68] since their cell sampling technique would leave us with a set $\mathbf{C}$ of size a factor t larger than what we obtain. Readjusting parameters, this would loose a factor $\lg \lg |\mathbf{F}|$ in the lower bound and bring us back to what can be achieved using the communication approach. We discuss this further when we give the proof of Lemma 2.1.

Encoding Algorithm. The algorithm for encoding $\mathbf{P}$ does the following:

1. First we construct the claimed data structure on $\mathbf{P}$ and obtain the set of cells $D(\mathbf{P})$. If for every integer $1 \leq \mathbf{t}^* \leq t$, $D(\mathbf{P})$ does not contain a set of cells $\mathbf{C}$ satisfying the properties of Lemma 2.1, we simply encode $\mathbf{P}$ as a 0-bit followed by a naive encoding of $\mathbf{P}$, taking a total of $1+\lceil (n+1) \lg |\mathbf{F}| \rceil \leq 2 + H(\mathbf{P})$ bits.

2. If the integer $1 \leq \mathbf{t}^* \leq t$ and the cell set $\mathbf{C}$ does exist, we first write a 1-bit. We then encode both $\mathbf{t}^*$ and $\mathbf{C}$, including addresses and contents of the cells in $\mathbf{C}$, for a total of $1 + \lg t + |\mathbf{C}|(w + \lg S) \leq 3|\mathbf{C}|w \leq 3/5 \cdot H(\mathbf{P})$ bits.

This completes the encoding procedure. Next we show how to recover $\mathbf{P}$ from the encoding:

Decoding Algorithm. To recover the polynomial $\mathbf{P}$ from the above encoding, we do the following: We start by examining the first bit. If this is a 0, we immediately recover $\mathbf{P}$ from the remaining part of the encoding. If the first bit is 1, we obtain the set of cells $\mathbf{C}$ and the integer $\mathbf{t}^*$ from the encoding. We now simulate the query algorithm for each of the $|\mathbf{F}|$ possible queries. For each such query q, if the query algorithm requests a cell outside $\mathbf{C}$ we simply discard q. Otherwise we recover the contents of the requested cell from the encoding and continue the simulation until we either discard the query or obtain the answer to it (possibly incorrect answer). Once this procedure is done we are left with all the queries that are resolved by $\mathbf{C}$. We now prune this set by deleting all queries where the query algorithm did not probe exactly $\mathbf{t}^*$ cells. We are then left with the set $G^{\mathbf{C}}_{\mathbf{t}^*}(\mathbf{P}) \cup B^{\mathbf{C}}_{\mathbf{t}^*}(\mathbf{P})$, including the correct answer to each query in $G^{\mathbf{C}}_{\mathbf{t}^*}(\mathbf{P})$ and an incorrect answer to each query in $B^{\mathbf{C}}_{\mathbf{t}^*}(\mathbf{P})$ (but we do not know which queries belong to each set). We finally iterate through all possible input polynomials and return as our candidate for $\mathbf{P}$, the polynomial which agrees with the most of the answers we have obtained for queries in $G^{\mathbf{C}}_{\mathbf{t}^*}(\mathbf{P}) \cup B^{\mathbf{C}}_{\mathbf{t}^*}(\mathbf{P})$.

To see that the returned polynomial in fact is $\mathbf{P}$, first recall the standard fact that any n-degree polynomial over a field is uniquely determined from $n+1$ distinct points on the polynomial. This implies that any two distinct polynomials over the input field $\mathbf{F}$ can agree on the answer to at most n evaluation queries. Thus any polynomial different from $\mathbf{P}$ can agree with at most n of the answers obtained for queries in $G^{\mathbf{C}}_{\mathbf{t}^*}(\mathbf{P})$ and possibly all answers obtained for queries in $B^{\mathbf{C}}_{\mathbf{t}^*}(\mathbf{P})$. By Lemma 2.1, this is bounded by $n + |B^{\mathbf{C}}_{\mathbf{t}^*}(\mathbf{P})| \leq n + |G^{\mathbf{C}}_{\mathbf{t}^*}(\mathbf{P})| - |G^{\mathbf{C}}_{\mathbf{t}^*}(\mathbf{P})|/2 = |G^{\mathbf{C}}_{\mathbf{t}^*}(\mathbf{P})| + n - n^{1+\Omega(1)} < |G^{\mathbf{C}}_{\mathbf{t}^*}(\mathbf{P})|$ query answers. But $\mathbf{P}$ agrees on all answers in $G^{\mathbf{C}}_{\mathbf{t}^*}(\mathbf{P})$ and hence it follows that the returned polynomial indeed is $\mathbf{P}$.

Analysis. Invoking Lemma 2.1, we get that the encoding uses at most

$$1/2 \cdot (2 + H(\mathbf{P})) + 1/2 \cdot 3/5 \cdot H(\mathbf{P}) < 9/10 \cdot H(\mathbf{P})$$

bits in expectation, i.e. a contradiction.

Proof of Lemma 2.1. In this paragraph, we prove the main technical result, namely Lemma 2.1. As noted, our approach differs slightly from that of Panigrahy et al. [68]. In their paper, they find a set of cells resolving many queries by picking t random samples of cells, one for each cell probe by the data structure. Our key idea is to pick one sample for all t probes simultaneously. This small difference is crucial to obtain the improved lower bounds when the space is less than a factor t from linear.

First, by Markov's inequality, we get that with probability at least $1/2$, there are at most $2 \cdot |\mathbf{F}|/18 = |\mathbf{F}|/9$ queries that err on input $\mathbf{P}$. When this happens, we show that there exists $\mathbf{t}^*$ and $\mathbf{C}$ satisfying all the properties of Lemma 2.1. This boils down to counting arguments:

We first choose $\mathbf{t}^*$. For this, initialize a candidate set $T = \{1, \dots, t\}$ of values for $\mathbf{t}^*$. Now define $G(\mathbf{P})$ as the set of queries that succeed on input $\mathbf{P}$ and similarly define $B(\mathbf{P})$ as the set of queries that err on input $\mathbf{P}$. By assumption we have $|B(\mathbf{P})| \leq |\mathbf{F}|/9$ and hence $|G(\mathbf{P})| \geq 8/9 \cdot |\mathbf{F}| \geq 8|B(\mathbf{P})|$. Examine each $i \in T$ in turn and collect for each choice the set $G_i(\mathbf{P})$, consisting of all queries in $G(\mathbf{P})$ that probe exactly i cells on input $\mathbf{P}$. For each i where $|G_i(\mathbf{P})| < |G(\mathbf{P})|/2t$, we remove i from T, i.e. we set $T \leftarrow T \setminus \{i\}$. After this step, we have $\sum_{i \in T} |G_i(\mathbf{P})| \geq |G(\mathbf{P})|/2$ and $|G_i(\mathbf{P})| \geq |G(\mathbf{P})|/2t$ for each $i \in T$.

Next, we examine each remaining $i \in T$ and remove all such i where $|B_i(\mathbf{P})| > |G_i(\mathbf{P})|/4$. Here $B_i(\mathbf{P})$ is the set of queries in $B(\mathbf{P})$ that probe exactly i cells on input $\mathbf{P}$. Since

$$\sum_{i \in T} |G_i(\mathbf{P})| \geq |G(\mathbf{P})|/2 \geq 4|B(\mathbf{P})| \geq 4 \sum_{i \in T} |B_i(\mathbf{P})|,$$

it follows that T is non-empty after this pruning step. We let $\mathbf{t}^*$ equal an arbitrary remaining value in T, thus we have $|G_{\mathbf{t}^*}(\mathbf{P})| \geq 4|B_{\mathbf{t}^*}(\mathbf{P})|$ and $|G_{\mathbf{t}^*}(\mathbf{P})| \geq |G(\mathbf{P})|/2t$.

We find $\mathbf{C}$ in a similar fashion. For ease of notation, define $\Delta = n \lg |\mathbf{F}|/5w$. First initialize a candidate set Y, containing all Δ-sized subsets of cells in $D(\mathbf{P})$. We thus have $|Y| = \binom{S}{\Delta}$. For a set $C' \in Y$, we define $G_{\mathbf{t}^*}^{C'}(\mathbf{P})$ ($B_{\mathbf{t}^*}^{C'}(\mathbf{P})$) as the subset of queries in $G_{\mathbf{t}^*}(\mathbf{P})$ ($B_{\mathbf{t}^*}(\mathbf{P})$) that are resolved by C', i.e. they probe only cells in C' on input $\mathbf{P}$. Observe that each query in $G_{\mathbf{t}^*}(\mathbf{P})$ and $B_{\mathbf{t}^*}(\mathbf{P})$ is resolved by precisely $\binom{S-\mathbf{t}^*}{\Delta-\mathbf{t}^*}$ sets in Y, hence $\sum_{C' \in Y} |G_{\mathbf{t}^*}^{C'}(\mathbf{P})| = |G_{\mathbf{t}^*}(\mathbf{P})|\binom{S-\mathbf{t}^*}{\Delta-\mathbf{t}^*}$ and $\sum_{C' \in Y} |B_{\mathbf{t}^*}^{C'}(\mathbf{P})| = |B_{\mathbf{t}^*}(\mathbf{P})|\binom{S-\mathbf{t}^*}{\Delta-\mathbf{t}^*}$.

We now prune Y by deleting all sets $C' \in Y$ for which

$$|G_{\mathbf{t}^*}^{C'}(\mathbf{P})| < |G_{\mathbf{t}^*}(\mathbf{P})|\binom{S-\mathbf{t}^*}{\Delta-\mathbf{t}^*}/2\binom{S}{\Delta}.$$

We then have both

- $\sum_{C' \in Y} |G_{\mathbf{t}^*}^{C'}(\mathbf{P})| \geq |G_{\mathbf{t}^*}(\mathbf{P})|\binom{S-\mathbf{t}^*}{\Delta-\mathbf{t}^*}/2$.

- $|G_{\mathbf{t}^*}^{C'}(\mathbf{P})| \geq |G_{\mathbf{t}^*}(\mathbf{P})|\binom{S-\mathbf{t}^*}{\Delta-\mathbf{t}^*}/2\binom{S}{\Delta}$.

for all remaining $C' \in Y$. As the last step, we remove all $C' \in Y$ for which $|B^{C'}_{\mathbf{t}^*}(\mathbf{P})| > |G^{C'}_{\mathbf{t}^*}(\mathbf{P})|/2$. Again, since

$$\begin{aligned}
\sum_{C' \in Y} |G^{C'}_{\mathbf{t}^*}(\mathbf{P})| &\geq |G_{\mathbf{t}^*}(\mathbf{P})| \binom{S - \mathbf{t}^*}{\Delta - \mathbf{t}^*}/2 \\
&\geq 2|B_{\mathbf{t}^*}(\mathbf{P})| \binom{S - \mathbf{t}^*}{\Delta - \mathbf{t}^*} \\
&\geq 2 \sum_{C' \in Y} |B^{C'}_{\mathbf{t}^*}(\mathbf{P})|,
\end{aligned}$$

we conclude that Y is non-empty after this step, and we let $\mathbf{C}$ equal an arbitrary remaining set. We have thus obtained $\mathbf{t}^*$ and $\mathbf{C}$ satisfying both

- $|G^{\mathbf{C}}_{\mathbf{t}^*}(\mathbf{P})| \geq |G_{\mathbf{t}^*}(\mathbf{P})|\binom{S-\mathbf{t}^*}{\Delta-\mathbf{t}^*}/2\binom{S}{\Delta} \geq |G(\mathbf{P})|\binom{S-\mathbf{t}^*}{\Delta-\mathbf{t}^*}/4t\binom{S}{\Delta}$.
- $|B^{\mathbf{C}}_{\mathbf{t}^*}(\mathbf{P})| \leq |G^{\mathbf{C}}_{\mathbf{t}^*}(\mathbf{P})|/2$.

Lemma 2.1 now follows since

$$\begin{aligned}
|G(\mathbf{P})| \frac{\binom{S-\mathbf{t}^*}{\Delta-\mathbf{t}^*}}{4t\binom{S}{\Delta}} &\geq 2|\mathbf{F}| \frac{\binom{S-\mathbf{t}^*}{\Delta-\mathbf{t}^*}}{9t\binom{S}{\Delta}} \\
&= |\mathbf{F}| \frac{2(S-\mathbf{t}^*)!\Delta!}{9tS!(\Delta-\mathbf{t}^*)!} \\
&\geq |\mathbf{F}| \frac{2(\Delta-\mathbf{t}^*)^{\mathbf{t}^*}}{9tS^{\mathbf{t}^*}} \\
&\geq |\mathbf{F}| \left(\frac{n \lg |\mathbf{F}|}{6Sw}\right)^t = |\mathbf{F}|^{1-o(1)},
\end{aligned}$$

where the last step followed from making the contradictory assumption that $t = o(\lg |\mathbf{F}| / \lg(Sw/n \lg |\mathbf{F}|))$. We finally conclude:

Theorem 2.1. *Any static cell probe data structure for evaluating an n-degree polynomial over a finite field $\boldsymbol{F}$ must have query cost $t = \Omega(\lg |\boldsymbol{F}| / \lg(Sw/n \lg |\boldsymbol{F}|))$ if $\boldsymbol{F}$ has size at least $n^{1+\Omega(1)}$. Here S is the space usage in number of cells and w is the cell size. This lower bound holds for randomized data structures with any constant error probability $\delta < 1/2$. For linear space (i.e. $S = O(n \lg |\boldsymbol{F}|/w)$), this lower bound simplifies to $t = \Omega(\lg |\boldsymbol{F}|)$.*

2.2 Dynamic Data Structures

In this section, we present a new technique for proving lower bounds for dynamic data structures. For dynamic data structures, the two performance metrics of interest are query time and update time. Observe however that the space usage is tightly coupled to the update time: If the update time is t_u, then in n update operations, the *largest* data structure that can be constructed has space usage nt_u.

We start by reviewing some of the previous techniques for proving dynamic lower bounds. Following that, we introduce our new technique and apply it to the two problems of dynamic polynomial evaluation and dynamic weighted orthogonal range counting.

2.2.1 Techniques

In this section, we first review the two previous techniques most important to this work, and then present our new technique.

Fredman and Saks [42]. This technique is known as the chronogram technique. The basic idea is to consider batches, or *epochs*, of updates to a data structure problem. More formally, one defines an epoch i for each $i = 1, \dots, \lg_\beta n$, where $\beta > 1$ is a parameter. The i'th epoch consists of performing β^i randomly chosen updates. The epochs occur in time from largest to smallest epoch, and at the end of epoch 1, every cell of the constructed data structure is associated to the epoch in which it was last updated. The goal is to argue that to answer a query after epoch 1, the query algorithm has to probe one cell associated to each epoch. Since a cell is only associated to one epoch, this gives a total query time lower bound of $\Omega(\lg_\beta n)$.

Arguing that the query algorithm must probe one cell associated to each epoch is done by setting β somewhat larger than the worst case update time t_u and the cell size w. Since cells associated to an epoch j cannot contain useful information about an epoch $i < j$ (the updates of epoch j were performed before knowing what the updates of epoch i was), one can ignore cells associated to such epochs when analysing the probes to an epoch i. Similarly, since all epochs following epoch i (future updates) writes a total of $O(\beta^{i-1}t_u) = o(\beta^i)$ cells, these cells do not contain enough information about the β^i updates of epoch i to be of any use (recall the updates are random, thus there is still much randomness left in epoch i after seeing the cells written in epochs $j < i$). Thus if the answer to a query depends on an update of epoch i, then the query algorithm must probe a cell associated to epoch i to answer the query.

We note that Fredman and Saks also defined the notion of epochs over a sequence of intermixed updates and queries. Here the epochs are defined relative to each query, and from this approach they obtain their amortized bounds.

Pătraşcu [69]. This technique uses the same setup as the chronogram technique, i.e. one considers epochs $i = 1, \dots, \lg_\beta n$ of updates, followed by one query. The idea is to use a static $\Omega(\lg_\beta n)$ lower bound proof to argue that the query algorithm must probe $\Omega(\lg_\beta n)$ cells from each epoch if the update time is $o((\lg n/\lg\lg n)^2)$. Summing over all epochs, this gives a lower bound of $\Omega(\lg^2_\beta n)$. In the following, we give a coarse overview of the general framework for doing so.

One first proves a lower bound on the amount of communication in the following (static) communication game (for every epoch i): Bob receives all epochs of updates to the dynamic data structure problem and Alice receives a set of queries and all updates of the epochs preceding epoch i. The goal for them is to compute the answer to Alice's queries after all the epochs of updates.

When such a lower bound has been established, one considers each epoch i in turn and uses the dynamic data structure to obtain an efficient protocol for the above communication game between Alice and Bob. The key idea is to let Alice simulate the query algorithm of the dynamic data structure on each of her queries, and whenever a cell associated to epoch i is requested, she asks Bob for the contents. Bob replies and she continues the simulation. Clearly the amount of communication is proportional to the number of probes to cells associated to epoch i, and thus a lower bound follows from the communication game lower bound. The main difficulty in implementing this protocol is that Alice must somehow recover the contents of the cells not associated to epoch i without asking Bob for it. This is accomplished by first letting Bob send all cells associated to epochs $j < i$ to Alice. For sufficiently large β, this does not break the communication lower bound. To let Alice know which cells that belong to epoch i, Bob also sends a Bloom filter specifying the addresses of the cells associated to epoch i. A Bloom filter is a membership data structure with a false positive probability. By setting the false positive probability to $1/\lg^c n$ for a large enough constant $c > 0$, the Bloom filter can be send using $O(\lg\lg n)$ bits per cell associated to epoch i. If $t_u = o((\lg n/\lg\lg n)^2)$, this totals $o(\beta^i \lg^2 n/\lg\lg n)$ bits.

Now Alice can execute the updates of the epochs preceding epoch i (epochs $j > i$) herself, and she knows the cells (contents and addresses) associated to epochs $j < i$. She also has a Bloom filter specifying the addresses of the cells associated to epoch i. Thus to answer her queries, she starts simulating the query algorithm. Each time a cell is requested, she first checks if it is associated to epochs $j < i$. If so, she has the contents herself and can continue the simulation. If not, she checks the Bloom filter to determine whether it belongs to epoch i. If the Bloom filter says no, the contents of the cell was not updated during epochs $j \leq i$ and thus she has the contents from the updates she executed initially. Finally, if the Bloom filter says yes, she asks Bob for the contents. Clearly the amount of communication is proportional to the number of probes to cells associated to epoch i plus some additional communication due to the $t_q/\lg^c n$ false positives.

To get any lower bound out of this protocol, sending the Bloom filter must cost less bits than it takes to describe the updates of epoch i (Bob's input). This is precisely why the lower bound of Pătraşcu requires large weights assigned to the input points.

Our Technique. Our new technique elegantly circumvents the limitations of Pătraşcu's technique by exploiting the cell sampling idea from Section 2.1.2. The basic setup is the same, i.e. we consider epochs $i = 1, \dots, \lg_\beta n$, where the i'th epoch consists of β^i updates. As with the two previous techniques, we associate a cell to the epoch in which it was last updates. Lower bounds now follow by showing that any data structure must probe $\Omega(\lg_\beta n)$ cells associated to each epoch i when answering a query at the end of epoch 1. Summing over all $\lg_\beta n$ epochs, this gives us a lower bound of $\Omega(\lg_\beta^2 n)$.

To show that $\Omega(\lg_\beta n)$ probes to cells associated to an epoch i are required, we assume for contradiction that a data structure probing $o(\lg_\beta n)$ cells associated to epoch i exists. Using this data structure, we then consider a game between an encoder and a decoder. The encoder receives as input the updates of all epochs, and must from this send a message to the decoder. The decoder then sees this message and all updates preceding epoch i and must from this uniquely recover the updates of epoch i. If the message is smaller than the entropy of the updates of epoch i (conditioned on preceding epochs), this gives an information theoretic contradiction. The trick is to find a way for the encoder to exploit the small number of probed cells to send a short message.

As mentioned, we use the cell sampling idea from to exploit the small number of probes. In Section 2.1.2 we saw that if C is a set of cells, and if the query algorithm of a data structure probes $o(\lg_\beta n)$ cells from C on average over all queries (for large enough β), then there is a subset of cells $C' \subseteq C$ which *resolves* a large number of queries. Similarly to Section 2.1.2, we say that a subset of cells $C' \subseteq C$ resolves a query, if the query algorithm probes no cells in $C \setminus C'$ when answering that query. What this observation gives us compared to the approach of Pătraşcu, is that we can find a large set of queries that are all resolved by the same small subset of cells associated to an epoch i. Thus we no longer have to specify all cells associated to epoch i, but only a small fraction.

With this observation in mind, the encoder proceeds as follows: First he executes all the updates of all epochs on the claimed data structure. He then sends all cells associated to epochs $j < i$. For large enough β, this message is smaller than the entropy of the β^i updates of epoch i. Letting C_i denote the cells associated to epoch i, the encoder then finds a subset of cells $C_i' \subseteq C_i$, such that a large number of queries are resolved by C_i'. He then sends a description of those cells and proceeds by finding a subset Q of the queries resolved by C_i', such that knowing the answer to all queries in Q reduces the entropy of the updates of epoch i by more than the number of bits needed to describe C_i', Q and the cells associated to epochs $j < i$. He then sends a description of Q followed by an encoding of the updates of epoch i, conditioned on the answers to queries in Q. Since the entropy of the updates of epoch i is reduced by more bits than was already send, this gives our contradiction (if the decoder can recover the updates from the above messages).

To recover the updates of epoch i, the decoder first executes the updates preceding epoch i. His goal is to simulate the query algorithm for every query in Q to recover all the answers. He achieves this in the following way: For each cell c requested when answering a query $q \in Q$, he examines the cells associated to epochs $j < i$ (those cells were send by the encoder), and if c is contained in one of those he immediately recovers the contents. If not, he proceeds by examining the set C_i'. If c is included in this set, he has again recovered the contents and can continue the simulation. Finally, if c is not in C_i', then c must be associated to an epoch preceding epoch i (since queries in Q probe no cells in $C_i \setminus C_i'$), thus the decoder recovers the contents of c from the updates that he executed initially. In this manner, the decoder can recover the answer to every query in Q, and from the last part of the message he recovers the updates of epoch i.

The main technical challenge in using our technique lies in arguing that if $o(\lg_\beta n)$ cells are probed amongst the cells associated to epoch i, then the claimed cell set C_i' and query set Q exists.

In Section 2.2.2 we first use our technique to prove a lower bound for dynamic polynomial evaluation. This problem is tailored towards giving as clean an introduction of our technique as possible. In Section 2.2.3 we then prove our lower bound for weighted range counting.

2.2.2 Dynamic Polynomial Evaluation

In this section, we prove a lower bound for dynamic polynomial evaluation. In this problem, we are given an input polynomial P over a finite field $\mathbf{F}$, where we assume $|\mathbf{F}| = \Omega(n^2)$. We think of P as being represented by its n roots $r_1, \ldots, r_n$, i.e. $P(x) = (x - r_1)(x - r_2) \cdots (x - r_n)$[1]. The goal is to support updating the roots, i.e. given an element $y_0 \in \mathbf{F}$ and an index $i \in \{1, \ldots, n\}$, we must support updating r_i to y_0. Initially, we have $r_1 = \cdots = r_n = 0$. A query to this problem asks to evaluate $P(x_0)$ for a query element $x_0 \in \mathbf{F}$. The lower bound we prove for this problem holds for randomized data structures with a constant probability of error $\delta < 1/2$.

While the problem may seem slightly artificial, we note that the problem has been chosen to give the cleanest possible introduction of our new technique.

The Lower Bound Proof.

Our aim is to prove a lower bound of $t_q = \Omega(\lg |\mathbf{F}| \lg n / \lg(wt_u / \lg |\mathbf{F}|) \lg(wt_u))$ for randomized data structures with a constant probability of error $\delta < 1/2$. At first sight, this bound looks rather complicated, thus we note that it simplifies to $t_q = \Omega((\lg n / \lg \lg n)^2)$ in that natural case of $w = \lg^{O(1)} n, t_u = \lg^{O(1)} n$ and $|\mathbf{F}| = n^{O(1)}$.

In the proof, we give a lower bound on the expected query cost of a deterministic data structure, and *not* on the worst case query cost using a reduction as in Section 2.1.2. We also assume there is some arbitrary, but fixed ordering on the elements of $\mathbf{F}$.

We prove the lower bound by giving a hard distribution over sequences of updates followed by one random query. We then bound the expected query cost of any deterministic data structure with error probability at most 1/18 over the distribution. To be precise, the expectation in the query cost is measured over a uniform random choice of query and a sequence of updates drawn from the hard distribution. Similarly, the error probability is defined as the probability that the data structure returns an incorrect answer on a uniform random query and a sequence of updates drawn from the hard distribution. By arguments similar to those in Section 2.1.2, this translates into an equivalent lower bound on the expected query cost for randomized data structures with any constant error probability $\delta < 1/2$.

The first step of the proof is thus to design a hard distribution over updates, followed by a uniform random query.

[1]Note that if the field is not algebraically closed, it is not possible to represent all polynomials over the field in this form. Since we are considering lower bounds, this is not an issue.

Hard Distribution. The hard distribution is simple: We first execute n updates, where the ith update sets the ith root of the maintained polynomial to a uniform random element from $\mathbf{F}$. Following the n updates, we execute an evaluation query at a uniform random element in $\mathbf{F}$. This concludes the hard distribution.

We use $\mathbf{x}_i$ to denote the random variable giving the uniform random element from $\mathbf{F}$ that is used in the ith update operation. We let $\mathbf{X} = \mathbf{x}_1 \cdots \mathbf{x}_n$ be the random variable giving all the n updates. Finally we let $\mathbf{q}$ denote the uniform random element in $\mathbf{F}$ that is used in the query.

High-Level Proof. For the remainder of this section, we assume the availability of a deterministic data structure for dynamic polynomial evaluation, having worst case update time t_u and error probability 1/18 over the hard distribution. Our goal is to lower bound the expected query cost of this data structure.

For this, conceptually divide the updates $\mathbf{X} = \mathbf{x}_1 \cdots \mathbf{x}_n$ into $\lg_\beta n$ *epochs* of size β^i for $i = 0, \dots, \lg_\beta n - 1$, where $\beta = (wt_u)^2$. Epoch 0 consists of the last update $\mathbf{x}_n$, and generally epoch i consists of updates $\mathbf{x}_{n_i+1}, \dots, \mathbf{x}_{n_i+\beta^i}$ where $n_i = n - \sum_{j=0}^{i} \beta^j$.

We let $\mathbf{X}_i = \mathbf{x}_{n_i+1} \cdots \mathbf{x}_{n_i+\beta^i}$ denote the random variable giving the updates of epoch i. For a sequence of updates $\mathbf{X}$, we define $D(\mathbf{X})$ as the set of cells stored by the available data structure after the sequence of updates $\mathbf{X}$. We additionally partition $D(\mathbf{X})$ into sets $D_i(\mathbf{X})$ for $i = 0, \dots, \lg_\beta n - 1$, where $D_i(\mathbf{X})$ consists of the subset of cells in $D(\mathbf{X})$ that was updated in epoch i, but not in epochs $j < i$, i.e. $D_i(\mathbf{X})$ is the set of cells last updated in epoch i. Finally, we define $t_i(\mathbf{X}, \mathbf{q})$ as the number of cells in $D_i(\mathbf{X})$ that is probed by the query algorithm when answering $\mathbf{q}$ after the updates $\mathbf{X}$. With this notation, our goal is to show

Lemma 2.2. *If $\beta = (wt_u)^2$, then $\mathbb{E}[t_i(\boldsymbol{X}, \boldsymbol{q})] = \Omega(\lg |\boldsymbol{F}| / \lg(wt_u / \lg |\boldsymbol{F}|))$ for all epochs $i \geq 1$.*

Before giving the proof of Lemma 2.2, we show that it immediately gives the claimed lower bound. Since the cells sets $D_i(\mathbf{X})$ are disjoint, we get that the number of cells probed when answering $\mathbf{q}$ is at least $\sum_i t_i(\mathbf{X}, \mathbf{q})$ (due to rounding, there are some updates happening before epoch $\lg_\beta n - 1$, therefore at least and not exactly). It now follows from linearity of expectation that the expected number of cells probed when answering $\mathbf{q}$ is $\Omega(\lg |\mathbf{F}| / \lg(wt_u / \lg |\mathbf{F}|) \cdot \lg_\beta n) = \Omega(\lg |\mathbf{F}| / \lg(wt_u / \lg |\mathbf{F}|) \cdot \lg n / \lg(wt_u))$.

What remains is thus to prove Lemma 2.2, which will be the focus of the remainder of this section.

Bounding the Probes to Epoch i (Proof of Lemma 2.2).

To prove Lemma 2.2 we assume for contradiction that the available data structure satisfies $\mathbb{E}[t_{i^*}(\mathbf{X}, \mathbf{q})] = o(\lg |\mathbf{F}| / \lg(wt_u / \lg |\mathbf{F}|))$ for some epoch $i^* \geq 1$. Now observe that $H(\mathbf{X} \mid \mathbf{x}_1 \cdots \mathbf{x}_{n_{i^*}}) = H(\mathbf{X}) - n_{i^*} \lg |\mathbf{F}| = (n - n_{i^*}) \lg |\mathbf{F}| \geq \beta^{i^*} \lg |\mathbf{F}|$. We finish the proof by showing that, conditioned on $\mathbf{x}_1 \cdots \mathbf{x}_{n_{i^*}}$, we can use the claimed data structure to encode $\mathbf{X}$ in less than $H(\mathbf{X} \mid \mathbf{x}_1 \cdots \mathbf{x}_{n_{i^*}})$ bits in expectation, i.e. a contradiction. As a technical detail, note that in contrast to Section 2.1.2, we are encoding a sequence of updates and not just a polynomial, thus it is important that the encoding not only recovers the polynomial corresponding to the updates $\mathbf{X}$, but it must also recover the *ordering* of the updates.

Before giving the encoding and decoding procedures, we present the main technical lemma. This lemma shows exactly what happens if the data structure probes too few cells from epoch i^*.

Lemma 2.3. *Let $i^* \geq 1$ be the epoch where $\mathbb{E}[t_{i^*}(\boldsymbol{X}, \boldsymbol{q})] = o(\lg |\boldsymbol{F}| / \lg(wt_u / \lg |\boldsymbol{F}|))$. Partition the field $\boldsymbol{F}$ into $|\boldsymbol{F}|^{1/4}$ consecutive groups of $|\boldsymbol{F}|^{3/4}$ elements, denoted $\boldsymbol{F}_1, \dots, \boldsymbol{F}_{|\boldsymbol{F}|^{1/4}}$ (based on the ordering on $\boldsymbol{F}$). Then there exists a choice of index $1 \leq j^* \leq |\boldsymbol{F}|^{1/4}$ such that with probability at least $1/2$ over the choice of $\boldsymbol{X}$, there exists a subset of cells $\boldsymbol{C}_{i^*} \subseteq D_{i^*}(\boldsymbol{X})$ such that the following holds*

- *All updates $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n$ are unique, i.e. $\boldsymbol{x}_i \neq \boldsymbol{x}_j$ for $i \neq j$.*
- $|\boldsymbol{C}_{i^*}| \leq 1/25 \cdot \beta^{i^*} \lg |\boldsymbol{F}| / w$.
- *Let $G_{j^*}^{\boldsymbol{C}_{i^*}}(\boldsymbol{X})$ denote the subset of queries from $\boldsymbol{F}_{j^*}$ that do not err on input $\boldsymbol{X}$, and where furthermore the query algorithm probes no cells in $D_{i^*}(\boldsymbol{X}) \setminus \boldsymbol{C}_{i^*}$ when answering a query in $G_{j^*}^{\boldsymbol{C}_{i^*}}(\boldsymbol{X})$ after the updates $\boldsymbol{X}$. Then $|G_{j^*}^{\boldsymbol{C}_{i^*}}(\boldsymbol{X})| = |\boldsymbol{F}|^{74/100 - o(1)} \geq n + 1$.*

As with Lemma 2.1, this lemma shows that if a data structure is too efficient, then there must exist a small subset of cells that solves many queries. However, in the dynamic setting, we need several additional properties to obtain a contradiction. As we explained when giving the high-level overview of our new technique (Section 2.2.1), we can reach a contradiction by encoding a subset of queries for which to simulate the query algorithm and obtain the corresponding answers. Since the answer to an evaluation query reveals at most $\lg |\mathbf{F}|$ bits, encoding these queries must take less than $\lg |\mathbf{F}|$ bits per query to obtain a contradiction. This is the reason why we focus on one particular subset of queries $\mathbf{F}_{j^*}$: Since $|\mathbf{F}_{j^*}| = |\mathbf{F}|^{3/4}$, encoding queries from $\mathbf{F}_{j^*}$ can be done in $\lg |\mathbf{F}|^{3/4}$ bits. This buys us $\lg |\mathbf{F}| - \lg |\mathbf{F}|^{3/4} = 1/4 \lg |\mathbf{F}|$ bits per query, which is enough to reach the contradiction.

We defer the proof of Lemma 2.3 to the end of the section, and instead move on to show how we use Lemma 2.3 in the encoding and decoding procedures.

Encoding. Given the sequence of updates $\mathbf{X}$, we first construct the claimed data structure on $\mathbf{X}$ and obtain the cell set $D(\mathbf{X})$ as well as the partitions into subsets $D_{\lg_\beta n - 1}(\mathbf{X}), \dots, D_0(\mathbf{X})$. We now encode $\mathbf{X}$ using the following simple procedure:

1. We first examine $D(\mathbf{X})$ to determine whether a cell set $\mathbf{C}_{i^*} \subseteq D_{i^*}(\mathbf{X})$ and a query set $G_{j^*}^{\mathbf{C}_{i^*}}(\mathbf{X})$ exists, satisfying all the properties of Lemma 2.3. This is done simply by trying all choices for $\mathbf{C}_{i^*}$ and $G_{j^*}^{\mathbf{C}_{i^*}}(\mathbf{X})$ (note that

j^* is the same for all choices of $\mathbf{X}$, thus it is assumed known both in the encoding and decoding procedure). If such sets do not exist, or if $\mathbf{x}_1, \dots, \mathbf{x}_n$ are not all distinct, we first write a 0-bit, followed by a naive encoding of updates $\mathbf{x}_{n_{i^*}+1}, \dots, \mathbf{x}_n$, taking a total of $1+\lceil (n-n_{i^*}) \lg |\mathbf{F}| \rceil \leq 2 + H(\mathbf{X} \mid \mathbf{x}_1 \cdots \mathbf{x}_{n_{i^*}})$ bits.

2. If the claimed sets $\mathbf{C}_{i^*}$ and $G_{j^*}^{\mathbf{C}_{i^*}}(\mathbf{X})$ exists and $\mathbf{x}_1, \dots, \mathbf{x}_n$ are all distinct, we instead start by writing a 1-bit. We then examine $G_{j^*}^{\mathbf{C}_{i^*}}(\mathbf{X})$ and find a subset, $\mathbf{Q}_{i^*} \subseteq G_{j^*}^{\mathbf{C}_{i^*}}(\mathbf{X})$, of $\beta^{i^*}+1$ queries, such that none of the queries in $\mathbf{Q}_{i^*}$ evaluate the polynomial at one of the roots that was set during the updates of epochs $j \neq i^*$, i.e. no query in $\mathbf{Q}_{i^*}$ is amongst $\mathbf{x}_1, \dots, \mathbf{x}_{n_{i^*}}, \mathbf{x}_{n_{i^*-1}+1}, \dots, \mathbf{x}_n$. Since $G_{j^*}^{\mathbf{C}_{i^*}}(\mathbf{X})$ contains at least $n+1$ elements, we can always find such a set of queries. We now write down a description of $\mathbf{C}_{i^*}$ and $\mathbf{Q}_{i^*}$, including addresses and contents of the cells in $\mathbf{C}_{i^*}$. Accounting for also writing down $|\mathbf{C}_{i^*}|$ this takes a total of $1 + w + |\mathbf{C}_{i^*}| 2w + \lg \binom{|\mathbf{F}|^{3/4}}{|\mathbf{Q}_{i^*}|} \leq 3/25 \cdot \beta^{i^*} \lg |\mathbf{F}| + \lg \binom{|\mathbf{F}|^{3/4}}{\beta^{i^*}+1}$ bits (here we exploit that $G_{j^*}^{\mathbf{C}_{i^*}}(\mathbf{X}) \subseteq \mathbf{F}_{j^*}$ to get the exponent 3/4).

3. Next we write down all updates following epoch i^*, $\mathbf{x}_{n_{i^*-1}+1}, \dots, \mathbf{x}_n$, and all cell sets $D_{i^*-1}(\mathbf{X}), \dots, D_0(\mathbf{X})$. This takes another $\sum_{j=0}^{i^*-1} O(|D_j(\mathbf{X})|w + \beta^j \lg |\mathbf{F}|) = O(\beta^{i^*-1}(\lg |\mathbf{F}| + wt_u))$ bits.

4. In the last step, we consider the sorted sequence of the updates in epoch i^*, i.e. sorted by the ordering in $\mathbf{F}$ of the values they assign the roots, and not by the time of executing the updates. From this sequence, we write down the permutation that brings the sequence back into sorted order of execution time, taking a total of $\lceil \lg(\beta^{i^*}!) \rceil$ bits.

Decoding. In the following we show how to recover $\mathbf{X}$ from the encoding. Recall that we are recovering $\mathbf{X}$ conditioned on the updates preceding epoch i^*, i.e. we are given access to $\mathbf{x}_1 \cdots \mathbf{x}_{n_{i^*}}$ when recovering $\mathbf{X}$. The decoding procedure does the following:

1. We start by examining the first bit of the encoding. If this is a 0-bit, we immediately recover $\mathbf{X}$ from the remainder of the encoding and the given updates $\mathbf{x}_1 \cdots \mathbf{x}_{n_{i^*}}$.

2. If the first bit of the encoding is a 1, we start by executing updates $\mathbf{x}_1 \cdots \mathbf{x}_{n_{i^*}}$ on the claimed data structure. From this, we construct the cell sets $D^*_{\lg_\beta n-1}(\mathbf{X}), \dots, D^*_{i^*+1}(\mathbf{X})$, where $D^*_i(\mathbf{X})$ denotes the set of cells that were updated in epoch i but not during epochs $i-1, \dots, i^*+1$. Note that $D_i(\mathbf{X}) \subseteq D^*_i(\mathbf{X})$ for $i = \lg_\beta n - 1, \dots, i^*+1$. From the encoding, we furthermore recover $D_{i^*-1}(\mathbf{X}), \dots, D_0(\mathbf{X})$ as well as updates $\mathbf{x}_{n_{i^*-1}+1}, \dots, \mathbf{x}_n$. Finally, we recover $\mathbf{C}_{i^*}$ and $\mathbf{Q}_{i^*}$ from the encoding.

3. The next step is to recover the answer to each query in $\mathbf{Q}_{i^*}$ as if the query was executed after all updates $\mathbf{X}$. For this, we examine each query

in $\mathbf{Q}_{i^*}$ in turn. For a query $q \in \mathbf{Q}_{i^*}$, we execute the query algorithm of the claimed data structure. For each cell c that is requested by the query algorithm, we first examine cell sets $D_{i^*-1}(\mathbf{X}), \dots, D_0(\mathbf{X})$ and if c is contained in any of them, we have immediately recovered the contents of c as it is in $D(\mathbf{X})$. If c is not in $D_{i^*-1}(\mathbf{X}), \dots, D_0(\mathbf{X})$ we continue by examining $\mathbf{C}_{i^*}$. If c is contained therein, we have again recovered the contents of c in $D(\mathbf{X})$ and we continue executing the query algorithm. If c is also not in $\mathbf{C}_{i^*}$, then since $\mathbf{Q}_{i^*} \subseteq G_{i^*}^{\mathbf{C}_{i^*}}(\mathbf{X})$ we know by Lemma 2.3 that c is not in $D_{i^*}(\mathbf{X}) \setminus \mathbf{C}_{i^*}$. Therefore, the contents of c has not been updated during epochs $i^*, \dots, 0$ and therefore we recover the contents of c in $D(\mathbf{X})$ from $D^*_{\lg_\beta n-1}(\mathbf{X}), \dots, D^*_{i^*+1}(\mathbf{X})$. Thus regardless of which cell the query algorithm requests, we can recover the contents as it is in $D(\mathbf{X})$. It follows that the query algorithm terminates with the answer to q after updates $\mathbf{X}$. Finally, since no queries in $\mathbf{Q}_{i^*}$ err on input $\mathbf{X}$, we conclude that the recovered answers are also correct.

4. Recovering $\mathbf{X}$ is now straightforward. From updates $\mathbf{x}_1, \dots, \mathbf{x}_{n_{i^*}}$ and $\mathbf{x}_{n_{i^*-1}+1}, \dots, \mathbf{x}_n$ we know $n - \beta^{i^*}$ points on the polynomial $P(\mathbf{X})$ corresponding to $\mathbf{X}$ since all $\mathbf{x}_i$'s are unique. Since the queries in $\mathbf{Q}_{i^*}$ do not evaluate $P(\mathbf{X})$ at $\mathbf{x}_1, \dots, \mathbf{x}_{n_{i^*}}, \mathbf{x}_{n_{i^*-1}+1}, \dots, \mathbf{x}_n$ and $|\mathbf{Q}_{i^*}| = \beta^{i^*} + 1$, the answers to queries in $\mathbf{Q}_{i^*}$ give us another $\beta^{i^*} + 1$ points on $P(\mathbf{X})$. Hence $P(\mathbf{X})$ is uniquely determined from the encoding. From $P(\mathbf{X})$, we find the β^{i^*} roots that are not amongst $\mathbf{x}_1, \dots, \mathbf{x}_{n_{i^*}}, \mathbf{x}_{n_{i^*-1}+1}, \dots, \mathbf{x}_n$, i.e. we find the β^{i^*} unique elements of $\mathbf{F}$ corresponding to $\mathbf{x}_{n_{i^*}+1}, \dots, \mathbf{x}_{n_{i^*-1}}$, but we do not know how they are permuted in $\mathbf{X}$. We finally recover $\mathbf{X}$ from the encoding of how to permute these elements.

Analysis. In the following, we bound the expected size of the encoding and finally reach a contradiction. We start by analysing the size of the encoding when the conditions of Lemma 2.3 are satisfied. In this case, we write down a total of

$$3/25 \cdot \beta^{i^*} \lg |\mathbf{F}| + \lg \binom{|\mathbf{F}|^{3/4}}{\beta^{i^*}+1} + O(\beta^{i^*-1}(\lg |\mathbf{F}| + wt_u)) + \lg(\beta^{i^*}!)$$

bits. Since $\beta = (wt_u)^2$, this is bounded by

$$3/25 \cdot \beta^{i^*} \lg |\mathbf{F}| + (\beta^{i^*}+1)\lg(|\mathbf{F}|^{3/4}/\beta^{i^*}) + O(\beta^{i^*} \lg |\mathbf{F}|/wt_u) + \beta^{i^*} \lg(\beta^{i^*}) + O(1).$$

This is again upper bounded by

$$\beta^{i^*}(\lg |\mathbf{F}|^{3/4} + 3/25 \lg |\mathbf{F}| + o(\lg |\mathbf{F}|))$$

bits, which finally gives

$$(87/100 + o(1))\beta^{i^*} \lg |\mathbf{F}| \le 9/10 H(\mathbf{X} \mid \mathbf{x}_1 \cdots \mathbf{x}_{n_{i^*}})$$

bits. From Lemma 2.3, we get that this is the amount of bits spend with probability at least 1/2, hence the expected size of the encoding is at most

$$1/2 \cdot (2 + H(\mathbf{X} \mid \mathbf{x}_1 \cdots \mathbf{x}_{n_{i^*}}) + 9/10 H(\mathbf{X} \mid \mathbf{x}_1 \cdots \mathbf{x}_{n_{i^*}})) < H(\mathbf{X} \mid \mathbf{x}_1 \cdots \mathbf{x}_{n_{i^*}}),$$

i.e. a contradiction.

Proof of Lemma 2.3. To prove Lemma 2.3, let $i^* \geq 1$ be an epoch in which $\mathbb{E}[t_{i^*}(\mathbf{X}, \mathbf{q})] = o(\lg |\mathbf{F}|/\lg(wt_u/\lg|\mathbf{F}|))$. We first partition $\mathbf{F}$ into the $|\mathbf{F}|^{1/4}$ consecutive groups $\mathbf{F}_1, \dots, \mathbf{F}_{|\mathbf{F}|^{1/4}}$ of $|\mathbf{F}|^{3/4}$ elements each (i.e. based on the ordering on $\mathbf{F}$). We choose j^* in the following way:

Let δ_j denote the error probability of the data structure when restricted to the queries in $\mathbf{F}_j$, i.e. δ_j is the probability of returning an incorrect result when answering a query drawn uniformly from $\mathbf{F}_j$ after a sequence of updates drawn from the hard distribution. Clearly $\sum_j \delta_j/|\mathbf{F}|^{1/4} \leq 1/18$ since the error probability over all queries is at most $1/18$. Also let $t^j_{i^*}(\mathbf{X})$ denote the average number of cells probed from $D_{i^*}(\mathbf{X})$ by the queries in $\mathbf{F}_j$ on input $\mathbf{X}$, i.e. $t^j_{i^*}(\mathbf{X}) = \sum_{q \in \mathbf{F}_j} t_{i^*}(\mathbf{X}, q)/|\mathbf{F}_j|$. We similarly have $\sum_j \mathbb{E}[t^j_{i^*}(\mathbf{X})]/|\mathbf{F}|^{1/4} = \mathbb{E}[t_{i^*}(\mathbf{X}, \mathbf{q})] = o(\lg|\mathbf{F}|/\lg(wt_u/\lg|\mathbf{F}|))$. It follows immediately from Markov's inequality and a union bound that there must exist a choice of j^* such that both $\delta_{j^*} \leq 4/18$ and $\mathbb{E}[t^{j^*}_{i^*}(\mathbf{X})] \leq 4\mathbb{E}[t_{i^*}(\mathbf{X}, \mathbf{q})] = o(\lg|\mathbf{F}|/\lg(wt_u/\lg|\mathbf{F}|))$. We let j^* equal an arbitrary such choice of index.

The last step is to show that the cell set $\mathbf{C}_{i^*}$ exists with probability at least $1/2$ over the choice of $\mathbf{X}$. For this, let $G_{j^*}(\mathbf{X})$ denote the subset of queries in $\mathbf{F}_{j^*}$ that succeed on input $\mathbf{X}$. Now using a union bound and Markov's inequality, we get that with probability at least $1/2$, there are both at most $|\mathbf{F}_{j^*}|17/18$ queries in $\mathbf{F}_{j^*}$ that err on input $\mathbf{X}$, i.e. $|G_{j^*}(\mathbf{X})| \geq 1/18|\mathbf{F}_{j^*}|$, and at the same time, we have $t^{j^*}_{i^*}(\mathbf{X}) \leq 100\mathbb{E}[t^{j^*}_{i^*}(\mathbf{X})] = o(\lg|\mathbf{F}|/\lg(wt_u/\lg|\mathbf{F}|))$ and finally all $\mathbf{x}_i$'s are unique (we have $|\mathbf{F}| = \Omega(n^2)$, thus all queries are unique with a very good constant probability when the constant in $\Omega(n^2)$ is large enough). When this happens, we show that the cell set $\mathbf{C}_{i^*}$ exists. First observe that since $|G_{j^*}(\mathbf{X})| = \Omega(|\mathbf{F}_{j^*}|)$ it follows that the average number of cells from $D_{i^*}(\mathbf{X})$ probed when answering a query in $G_{j^*}(\mathbf{X})$ is $O(t^{j^*}_{i^*}(\mathbf{X})) = o(\lg|\mathbf{F}|/\lg(wt_u/\lg|\mathbf{F}|))$. Hence there are $\Omega(|G_{j^*}(\mathbf{X})|) = \Omega(|\mathbf{F}_{j^*}|)$ queries in $G_{j^*}(\mathbf{X})$ that probe $o(\lg|\mathbf{F}|/\lg(wt_u/\lg|\mathbf{F}|))$ cells from $D_{i^*}(\mathbf{X})$. We thus prune $G_{j^*}(\mathbf{X})$ by deleting all queries that probe at least $1/100(\lg|\mathbf{F}|/\lg(wt_u/\lg|\mathbf{F}|))$ cells from $D_{i^*}(\mathbf{X})$ and we still have $|G_{j^*}(\mathbf{X})| = \Omega(|\mathbf{F}_{j^*}|) = \Omega(|\mathbf{F}|^{3/4})$.

Now consider all subsets of $\Delta = 1/25 \cdot \beta^{i^*} \lg|\mathbf{F}|/w$ cells in $D_{i^*}(\mathbf{X})$. Since any remaining query in $G_{j^*}(\mathbf{X})$ probes at most $\mu = 1/100(\lg|\mathbf{F}|/\lg(wt_u/\lg|\mathbf{F}|))$ cells from $D_{i^*}(\mathbf{X})$, we get that there must exist a subset $\mathbf{C}' \subseteq D_{i^*}(\mathbf{X})$ of Δ cells, for which at least $|G_{j^*}(\mathbf{X})|\binom{|D_{i^*}(\mathbf{X})|-\mu}{\Delta-\mu}/\binom{|D_{i^*}(\mathbf{X})|}{\Delta}$ queries in $G_{j^*}(\mathbf{X})$ probes no cells in $D_{i^*}(\mathbf{X}) \setminus \mathbf{C}'$. Since

$$\begin{aligned}
|G_{j^*}(\mathbf{X})|\frac{\binom{|D_{i^*}(\mathbf{X})|-\mu}{\Delta-\mu}}{\binom{|D_{i^*}(\mathbf{X})|}{\Delta}} &= \Omega\left(|\mathbf{F}|^{3/4}\frac{(|D_{i^*}(\mathbf{X})|-\mu)!\Delta!}{|D_{i^*}(\mathbf{X})|!(\Delta-\mu)!}\right) \\
&= \Omega\left(|\mathbf{F}|^{3/4}\frac{(\Delta-\mu)^{\mu}}{|D_{i^*}(\mathbf{X})|^{\mu}}\right) \\
&= |\mathbf{F}|^{3/4}\left(\Omega\left(\frac{\lg|\mathbf{F}|}{wt_u}\right)\right)^{\mu} \\
&= |\mathbf{F}|^{74/100-o(1)},
\end{aligned}$$

we conclude that the claimed set $\mathbf{C}_{i^*}$ exists. We finally get

Theorem 2.2. *Any dynamic cell probe data structure for evaluating an n-degree polynomial over a finite field $\boldsymbol{F}$ must have query cost, t_q, satisfying $t_q = \Omega(\lg|\boldsymbol{F}| \lg n/\lg(wt_u/\lg|\boldsymbol{F}|)\lg(wt_u))$ if $\boldsymbol{F}$ has size $\Omega(n^2)$. Here t_u is the worst case update time and w is the cell size. This lower bound holds for randomized data structures with any constant error probability $\delta < 1/2$.*

2.2.3 Dynamic Weighted Orthogonal Range Counting

In this section we prove our lower bound for dynamic weighted orthogonal range counting. An update to this problem consists of the insertion of a point on the two-dimensional grid $[n] \times [n]$. Each inserted point is assigned a δ-bit integer weight. A query is specified by a point $(x, y) \in [n] \times [n]$ and the goal is to return the sum of the weights assigned to the points *dominated* by (x, y). Here we say that a point (x', y') is dominated by (x, y) iff $x' \leq x$ and $y' \leq y$.

The lower bound we aim to prove states that any data structure for dynamic weighted orthogonal range counting must satisfy $t_q = \Omega((\lg n/\lg(wt_u))^2)$, where t_q is the expected query time, t_u the worst case update time and w the cell size in bits. This lower bound holds when the weights of the inserted points are $\Theta(\lg n)$-bit integers.

As in Section 2.2.2, we prove the lower bound by devising a hard distribution over updates, followed by one uniform random query. We then lower bound the expected cost (over the distribution) of answering the query for any *deterministic* data structure with worst case update time t_u. In the proof we assume the weights are $4\lg n$-bit integers and note that the lower bound applies to any $\varepsilon \lg n$-bit weights, where $\varepsilon > 0$ is an arbitrarily small constant, simply because a data structure for $\varepsilon \lg n$-bit integer weights can be used to solve the problem for any $O(\lg n)$-bit integer weights with a constant factor overhead by dividing the bits of the weights into $\lceil \delta/(\varepsilon \lg n) \rceil = O(1)$ chunks and maintaining a data structure for each chunk. We begin the proof by presenting the hard distribution over updates and queries.

Hard Distribution. Again, updates arrive in epochs of exponentially decreasing size. For $i = 1, \ldots, \lg_\beta n$ we define epoch i as a sequence of β^i updates, for a parameter $\beta > 1$ to be fixed later. The epochs occur in time from biggest to smallest epoch, and at the end of epoch 1 we execute a uniform random query in $[n] \times [n]$.

What remains is to specify which updates are performed in each epoch i. The updates of epoch i are chosen to mimic the hard input distribution for static orthogonal range counting on a set of β^i points (see e.g. [69]). We first define the following point set known as the Fibonacci lattice:

Definition 2.1 (Matoušek [60])**.** *The Fibonacci lattice F_m is the set of m two-dimensional points defined by $F_m = \{(i, if_{k-1} \bmod m) \mid i = 0, \ldots, m-1\}$, where $m = f_k$ is the k'th Fibonacci number.*

The β^i updates of epoch i now consists of inserting each point of the Fibonacci lattice F_{β^i}, but scaled to fit the input region $[n] \times [n]$, i.e. the j'th update of epoch i inserts the point with coordinates $(n/\beta^i \cdot j, n/\beta^i \cdot (jf_{k_i - 1}$

mod β^i)), for $j = 0, \ldots, \beta^i$. The weight of each inserted point is a uniform random integer amongst $[\Delta]$, where Δ is the largest prime number smaller than $2^{4 \lg n} = n^4$. This concludes the description of the hard distribution.

The Fibonacci lattice has the desirable property that it is very uniform. This plays an important role in the lower bound proof, and we have formulated this property in the following lemma:

Lemma 2.4 (Fiat and Shamir [41]). *For the Fibonacci lattice F_{β^i}, where the coordinates of each point have been multiplied by n/β^i, and for $\alpha > 0$, any axis-aligned rectangle in $[0 : n - n/\beta^i] \times [0 : n - n/\beta^i]$ with area $\alpha n^2/\beta^i$ contains between $\lfloor \alpha/a_1 \rfloor$ and $\lceil \alpha/a_2 \rceil$ points, where $a_1 \approx 1.9$ and $a_2 \approx 0.45$.*

Note that we assume each β^i to be a Fibonacci number (denoted f_{k_i}), and that each β^i divides n. These assumptions can easily be removed by fiddling with the constants, but this would only clutter the exposition.

For the remainder of this chapter, we let $\mathbf{U}_i$ denote the random variable giving the sequence of updates in epoch i, and we let $\mathbf{U} = \mathbf{U}_{\lg_\beta n} \cdots \mathbf{U}_1$ denote the random variable giving all updates of all $\lg_\beta n$ epochs. Finally, we let $\mathbf{q}$ be the random variable giving the query.

A Chronogram. Having defined the hard distribution over updates and queries, we proceed as in Section 2.2.2. Assume a deterministic data structure solution exists with worst case update time t_u. From this data structure and a sequence of updates $\mathbf{U} = \mathbf{U}_{\lg_\beta n}, \ldots, \mathbf{U}_1$, we define $D(\mathbf{U})$ to be the set of cells stored in the data structure after executing the updates $\mathbf{U}$. Associate each cell in $D(\mathbf{U})$ to the last epoch in which its contents were updated, and let $D_i(\mathbf{U})$ denote the subset of $D(\mathbf{U})$ associated to epoch i for $i = 1, \ldots, \lg_\beta n$. Also let $t_i(\mathbf{U}, q)$ denote the number of cells in $D_i(\mathbf{U})$ probed by the query algorithm of the data structure when answering the query $q \in [n] \times [n]$ after the sequence of updates $\mathbf{U}$. Finally, let $t_i(\mathbf{U})$ denote the average cost of answering a query $q \in [n] \times [n]$ after the sequence of updates $\mathbf{U}$, i.e. let $t_i(\mathbf{U}) = \sum_{q \in [n] \times [n]} t_i(\mathbf{U}, q)/n^2$. Then the following holds:

Lemma 2.5. *If $\beta = (wt_u)^9$, then $\mathbb{E}[t_i(\boldsymbol{U}, \boldsymbol{q})] = \Omega(\lg_\beta n)$ for all $i \geq \frac{15}{16} \lg_\beta n$.*

Since the cell sets $D_{\lg_\beta n}(\mathbf{U}), \ldots, D_1(\mathbf{U})$ are disjoint, we get by linearity of expectation that the expected query time must be at least $\Omega((\lg n / \lg(wt_u))^2)$. Thus what remains is to prove Lemma 2.5. This is the focus in the following:

Bounding the Probes to Epoch i (Proof of Lemma 2.5).

The proof of Lemma 2.5 is again based on an encoding argument. The framework is identical to Section 2.2.2, but arguing that a "good" set of queries to simulate exists is significantly more difficult.

Assume for contradiction that there exists a data structure solution such that under the hard distribution, with $\beta = (wt_u)^9$, there exists an epoch $i^* \geq \frac{15}{16} \lg_\beta n$, such that the claimed data structure satisfies $\mathbb{E}[t_{i^*}(\mathbf{U}, \mathbf{q})] = o(\lg_\beta n)$.

First observe that $\mathbf{U}_{i^*}$ is independent of $\mathbf{U}_{\lg_\beta n} \cdots \mathbf{U}_{i^*+1}$, i.e. $H(\mathbf{U}_{i^*} \mid \mathbf{U}_{\lg_\beta n} \cdots \mathbf{U}_{i^*+1}) = H(\mathbf{U}_{i^*})$. Furthermore, we have $H(\mathbf{U}_{i^*}) = \beta^{i^*} \lg \Delta$, since the updates of epoch i^* consists of inserting β^{i^*} fixed points, each with a uniform random weight amongst the integers $[\Delta]$. Our goal is to show that, conditioned on $\mathbf{U}_{\lg_\beta n} \cdots \mathbf{U}_{i^*+1}$, we can use the claimed data structure solution to encode $\mathbf{U}_{i^*}$ in less than $H(\mathbf{U}_{i^*})$ bits in expectation, which provides the contradiction.

Before presenting the encoding and decoding procedures, we show what happens if a data structure probes too few cells from epoch i^*. For this, we first introduce some terminology. For a query point $q = (x, y) \in [n] \times [n]$, we define for each epoch $i = 1, \ldots, \lg_\beta n$ the *incidence vector* $\chi_i(q)$, as a $\{0,1\}$-vector in $[\Delta]^{\beta^i}$. The j'th coordinate of $\chi_i(q)$ is 1 if the j'th point inserted in epoch i is dominated by q, and 0 otherwise. More formally, for a query $q = (x, y)$, the j'th coordinate $\chi_i(q)_j$ is given by:

$$\chi_i(q)_j = \begin{cases} 1 & \text{if } jn/\beta^i \leq x \wedge (jf_{k_i-1} \bmod \beta^i)n/\beta^i \leq y \\ 0 & \text{otherwise} \end{cases}$$

Similarly, we define for a sequence of updates $\mathbf{U}_i$, the β^i-dimensional vector $\mathbf{u}_i$ for which the j'th coordinate equals the weight assigned to the j'th inserted point in $\mathbf{U}_i$. We note that $\mathbf{U}_i$ and $\mathbf{u}_i$ uniquely specify each other, since $\mathbf{U}_i$ always inserts the same fixed points, only the weights vary.

Finally observe that the answer to a query q after a sequence of updates $\mathbf{U}_{\lg_\beta n}, \ldots, \mathbf{U}_1$ is $\sum_{i=1}^{\lg_\beta n} \langle \chi_i(q), \mathbf{u}_i \rangle$. With these definitions, we now present the main result forcing a data structure to probe many cells from each epoch:

Lemma 2.6. *Let $i \geq \frac{15}{16} \lg_\beta n$ be an epoch. If $t_i(\boldsymbol{U}) = o(\lg_\beta n)$, then there exists a subset of cells $C_i(\boldsymbol{U}) \subseteq D_i(\boldsymbol{U})$ and a set of query points $Q(\boldsymbol{U}) \subseteq [n] \times [n]$ such that:*

1. $|C_i(\boldsymbol{U})| = O(\beta^{i-1} w)$.
2. $|Q(\boldsymbol{U})| = \Omega(\beta^{i-3/4})$.
3. *The set of incidence vectors $\chi_i(Q(\boldsymbol{U})) = \{\chi_i(q) \mid q \in Q(\boldsymbol{U})\}$ is a linearly independent set of vectors in $[\Delta]^{\beta^i}$.*
4. *The query algorithm of the data structure solution probes no cells in $D_i(\boldsymbol{U}) \setminus C_i(\boldsymbol{U})$ when answering a query $q \in Q(\boldsymbol{U})$ after the sequence of updates $\boldsymbol{U}$.*

Comparing to Lemma 2.3, it is not surprising that this lemma gives the lower bound. We note that Lemma 2.6 essentially is a generalization of the results proved in the static range counting papers [69, 49], simply phrased in terms of cell subsets answering many queries instead of communication complexity. Since the proof contains only few new ideas, we have deferred it a bit and instead move on to the encoding and decoding procedures.

Encoding. Let $i^* \geq \frac{15}{16} \lg_\beta n$ be the epoch for which $\mathbb{E}[t_{i^*}(\mathbf{U}, \mathbf{q})] = o(\lg_\beta n)$. The encoding procedure follows the same overall approach as in Section 2.2.2:

1. First the encoder executes the sequence of updates $\mathbf{U}$ on the claimed data structure, and from this obtains the sets $D_{\lg_\beta n}(\mathbf{U}), \ldots, D_1(\mathbf{U})$. He then simulates the query algorithm on the data structure for every query $q \in [n] \times [n]$. From this, the encoder computes $t_{i^*}(\mathbf{U})$.

2. If $t_{i^*}(\mathbf{U}) > 2\mathbb{E}[t_{i^*}(\mathbf{U}, \mathbf{q})]$, then the encoder writes a 1-bit, followed by $\lceil \beta^{i^*} \lg \Delta \rceil = H(\mathbf{U}_{i^*}) + O(1)$ bits, simply specifying each weight assigned to a point in $\mathbf{U}_{i^*}$. This is the complete message send to the decoder when $t_{i^*}(\mathbf{U}) > 2\mathbb{E}[t_{i^*}(\mathbf{U}, \mathbf{q})]$.

3. If $t_{i^*}(\mathbf{U}) \leq 2\mathbb{E}[t_{i^*}(\mathbf{U}, \mathbf{q})]$, then the encoder first writes a 0-bit. Now since $t_{i^*}(\mathbf{U}) \leq 2\mathbb{E}[t_{i^*}(\mathbf{U}, \mathbf{q})] = o(\lg_\beta n)$, we get from Lemma 2.6 that there must exist a set of cells $C_{i^*}(\mathbf{U}) \subseteq D_{i^*}(\mathbf{U})$ and a set of queries $Q(\mathbf{U}) \subseteq [n] \times [n]$ satisfying 1-4 in Lemma 2.6. The encoder finds such sets $C_{i^*}(\mathbf{U})$ and $Q(\mathbf{U})$ simply by trying all possible sets in some arbitrary but fixed order. The encoder now writes down these two sets, including addresses and contents of the cells in $C_{i^*}(\mathbf{U})$, for a total of at most $O(w) + 2|C_{i^*}(\mathbf{U})|w + \lg \binom{n^2}{|Q(\mathbf{U})|}$ bits (the $O(w)$ bits specifies $|C_{i^*}(\mathbf{U})|$ and $|Q(\mathbf{U})|$).

4. The encoder now constructs a set X, such that $X = \chi_{i^*}(Q(\mathbf{U})) = \{\chi_{i^*}(q) \mid q \in Q(\mathbf{U})\}$ initially. Then he iterates through all vectors in $[\Delta]^{\beta^{i^*}}$, in some arbitrary but fixed order, and for each such vector x, checks whether x is in $\text{span}(X)$. If not, the encoder adds x to X. This process continues until $\dim(\text{span}(X)) = \beta^{i^*}$, at which point the encoder computes and writes down $(\langle x, \mathbf{u}_{i^*} \rangle \mod \Delta)$ for each x that was added to X. Since $\dim(\text{span}(\chi_{i^*}(Q(\mathbf{U})))) = |Q(\mathbf{U})|$ (by point 3 in Lemma 2.6), this adds a total of $\lceil (\beta^{i^*} - |Q(\mathbf{U})|) \lg \Delta \rceil$ bits to the message.

5. Finally, the encoder writes down all of the cell sets $D_{i^*-1}(\mathbf{U}), \ldots, D_1(\mathbf{U})$, including addresses and contents, plus all of the vectors $\mathbf{u}_{i^*-1}, \ldots, \mathbf{u}_1$. This takes at most $\sum_{j=1}^{i^*-1}(2|D_j(\mathbf{U})|w + \beta^j \lg \Delta + O(w))$ bits. When this is done, the encoder sends the constructed message to the decoder.

Next we present the decoding procedure:

Decoding. The decoder receives as input the updates $\mathbf{U}_{\lg_\beta n}, \ldots, \mathbf{U}_{i^*+1}$ and the message from the encoder. The decoder now recovers $\mathbf{U}_{i^*}$ by the following procedure:

1. The decoder examines the first bit of the message. If this bit is 1, then the decoder immediately recovers $\mathbf{U}_{i^*}$ from the encoding (step 2 in the encoding procedure). If not, the decoder instead executes the updates $\mathbf{U}_{\lg_\beta n} \cdots \mathbf{U}_{i^*+1}$ on the claimed data structure solution and obtains the cells sets $D^*_{\lg_\beta n}(\mathbf{U}), \ldots, D^*_{i^*+1}(\mathbf{U})$, where $D^*_j(\mathbf{U})$ contains the cells that were last updated during epoch j when executing only the updates $\mathbf{U}_{\lg_\beta n}, \ldots, \mathbf{U}_{i^*+1}$. We have $D_j(\mathbf{U}) \subseteq D^*_j(\mathbf{U})$ for all $j \geq i^* + 1$.

2. The decoder now recovers the sets $Q(\mathbf{U}), C_{i^*}(\mathbf{U}), D_{i^*-1}(\mathbf{U}), \ldots, D_1(\mathbf{U})$ and $\mathbf{u}_{i^*-1}, \ldots, \mathbf{u}_1$ from the encoding. For each query $q \in Q(\mathbf{U})$, the decoder then computes the answer to q as if all updates $\mathbf{U}_{\lg_\beta n}, \ldots, \mathbf{U}_1$ had been performed. The decoder accomplishes this by simulating the query algorithm on q, and for each cell requested, the decoder recovers the contents of that cell as it would have been if all updates $\mathbf{U}_{\lg_\beta n}, \ldots, \mathbf{U}_1$ had been performed. This is done as follows: When the query algorithm requests a cell c, the decoder first determines whether c is in one of the sets $D_{i^*-1}(\mathbf{U}), \ldots, D_1(\mathbf{U})$. If so, the correct contents of c is directly recovered. If c is not amongst these cells, the decoder checks whether c is in $C_{i^*}(\mathbf{U})$. If so, the decoder has again recovered the contents. Finally, if c is not in $C_{i^*}(\mathbf{U})$, then from point 4 of Lemma 2.6, we get that c is not in $D_{i^*}(\mathbf{U})$. Since c is not in any of $D_{i^*}(\mathbf{U}), \ldots, D_1(\mathbf{U})$, this means that the contents of c has not changed during the updates $\mathbf{U}_{i^*}, \ldots, \mathbf{U}_1$, and thus the decoder finally recovers the contents of c from $D^*_{\lg_\beta n}(\mathbf{U}), \ldots, D^*_{i^*+1}(\mathbf{U})$. The decoder can therefore recover the answer to each query q in $Q(\mathbf{U})$ if it had been executed after the sequence of updates $\mathbf{U}$, i.e. for all $q \in Q(\mathbf{U})$, he knows $\sum_{i=1}^{\lg_\beta n} \langle \chi_i(q), \mathbf{u}_i \rangle$.

3. The next decoding step consists of computing for each query q in $Q(\mathbf{U})$, the value $\langle \chi_{i^*}(q), \mathbf{u}_{i^*} \rangle$. For each $q \in Q(\mathbf{U})$, the decoder already knows the value $\sum_{i=1}^{\lg_\beta n} \langle \chi_i(q), \mathbf{u}_i \rangle$ from the above. From the encoding of $\mathbf{u}_{i^*-1}, \ldots, \mathbf{u}_1$, the decoder can compute the value $\sum_{i=1}^{i^*-1} \langle \chi_i(q), \mathbf{u}_i \rangle$ and finally from $\mathbf{U}_{\lg_\beta n}, \ldots, \mathbf{U}_{i^*+1}$ the decoder computes $\sum_{i=i^*+1}^{\lg_\beta n} \langle \chi_i(q), \mathbf{u}_i \rangle$. The decoder can now recover the value $\langle \chi_{i^*}(q), \mathbf{u}_{i^*} \rangle$ simply by observing that:
$$\langle \chi_{i^*}(q), \mathbf{u}_{i^*} \rangle = \sum_{i=1}^{\lg_\beta n} \langle \chi_i(q), \mathbf{u}_i \rangle - \sum_{i \neq i^*} \langle \chi_i(q), \mathbf{u}_i \rangle.$$

4. Now from the query set $Q(\mathbf{U})$, the decoder construct the set of vectors $X = \chi_{i^*}(Q(\mathbf{U}))$, and then iterates through all vectors in $[\Delta]^{\beta^{i^*}}$, in the same fixed order as the encoder. For each such vector x, the decoder again verifies whether x is in $\mathrm{span}(X)$, and if not, adds x to X and recovers $(\langle x, \mathbf{u}_{i^*} \rangle \mod \Delta)$ from the encoding. The decoder now constructs the $\beta^{i^*} \times \beta^{i^*}$ matrix A, having the vectors in X as rows. Similarly, he construct the vector $\mathbf{z}$ having one coordinate for each row of A. The coordinate of $\mathbf{z}$ corresponding to a row vector x, has the value $(\langle x, \mathbf{u}_{i^*} \rangle \mod \Delta)$. Since A has full rank, it follows that the linear system of equations $A \otimes \mathbf{y} = \mathbf{z}$ has a unique solution $\mathbf{y} \in [\Delta]^{\beta^{i^*}}$. Here $\otimes$ denotes matrix-vector multiplication modulo Δ. But $\mathbf{u}_{i^*} \in [\Delta]^{\beta^{i^*}}$ and $A \otimes \mathbf{u}_{i^*} = \mathbf{z}$, thus the decoder solves the linear system of equations $A \otimes \mathbf{y} = \mathbf{z}$ and uniquely recovers $\mathbf{u}_{i^*}$, and therefore also $\mathbf{U}_{i^*}$. This completes the decoding procedure.

Analysis. We now analyse the expected size of the encoding of $\mathbf{U}_{i^*}$. We first analyse the size of the encoding when $t_{i^*}(\mathbf{U}) \leq 2\mathbb{E}[t_{i^*}(\mathbf{U}, \mathbf{q})]$. In this case, the encoder sends a message of

$$2|C_{i^*}(\mathbf{U})|w + \lg \binom{n^2}{|Q(\mathbf{U})|} + (\beta^{i^*} - |Q(\mathbf{U})|)\lg \Delta$$
$$+O(w \lg_\beta n) + \sum_{j=1}^{i^*-1}(2|D_j(\mathbf{U})|w + \beta^j \lg \Delta)$$

bits. Since $\beta^{i^*} \lg \Delta = H(\mathbf{U}_{i^*})$ and $|C_{i^*}(\mathbf{U})|w = o(|Q(\mathbf{U})|)$, the above is upper bounded by

$$H(\mathbf{U}_{i^*}) - |Q(\mathbf{U})|\lg(\Delta/n^2) + o(|Q(\mathbf{U})|) + \sum_{j=1}^{i^*-1}(2|D_j(\mathbf{U})|w + \beta^j \lg \Delta).$$

Since $\beta \geq 2$, we also have $\sum_{j=1}^{i^*-1} \beta^j \lg \Delta \leq 2\beta^{i^*-1}\lg \Delta = o(|Q(\mathbf{U})|\lg \Delta)$. Similarly, we have $|D_j(\mathbf{U})| \leq \beta^j t_u$, which gives us $\sum_{j=1}^{i^*-1} 2|D_j(\mathbf{U})|w \leq 4\beta^{i^*-1}wt_u = o(|Q(\mathbf{U})|)$. From standard results on prime numbers, we have that the largest prime number smaller than n^4 is at least n^3 for infinitely many n, i.e. we can assume $\lg(\Delta/n^2) = \Omega(\lg \Delta)$. Therefore, the above is again upper bounded by

$$H(\mathbf{U}_{i^*}) - \Omega(|Q(\mathbf{U})|\lg \Delta) = H(\mathbf{U}_{i^*}) - \Omega(\beta^{i^*-3/4}\lg \Delta).$$

This part thus contributes at most

$$\Pr[t_{i^*}(\mathbf{U}) \leq 2\mathbb{E}[t_{i^*}(\mathbf{U},\mathbf{q})]] \cdot (H(\mathbf{U}_{i^*}) - \Omega(\beta^{i^*-3/4}\lg \Delta))$$

bits to the expected size of the encoding. The case where $t_{i^*}(\mathbf{U}) > 2\mathbb{E}[t_{i^*}(\mathbf{U},\mathbf{q})]$ similarly contributes $\Pr[t_{i^*}(\mathbf{U}) > 2\mathbb{E}[t_{i^*}(\mathbf{U},\mathbf{q})]] \cdot (H(\mathbf{U}_{i^*}) + O(1))$ bits to the expected size of the encoding. Now since $\mathbf{q}$ is uniform, we have $\mathbb{E}[t_{i^*}(\mathbf{U})] = \mathbb{E}[t_{i^*}(\mathbf{U},\mathbf{q})]$, we therefore get from Markov's inequality that

$$\Pr[t_{i^*}(\mathbf{U}) > 2\mathbb{E}[t_{i^*}(\mathbf{U},\mathbf{q})]] < \tfrac{1}{2}.$$

Therefore the expected size of the encoding is upper bounded by

$$O(1) + \tfrac{1}{2}H(\mathbf{U}_{i^*}) + \tfrac{1}{2}(H(\mathbf{U}_{i^*}) - \Omega(\beta^{i^*-3/4}\lg \Delta)) < H(\mathbf{U}_{i^*}).$$

This completes the proof of Lemma 2.5.

The Static Setup (Proof of Lemma 2.6).

In the following, we prove the last piece in the lower bound proof. As already mentioned, we prove Lemma 2.6 by extending on previous ideas for proving lower bounds on static range counting. We note that we have chosen a more geometric (and we believe more intuitive) approach to the proof than the previous papers.

For the remainder of the section, we let $U = U_{\lg_\beta n}, \dots, U_1$ be a fixed sequence of updates, where each U_j is a possible outcome of $\mathbf{U}_j$, and $i \geq \frac{15}{16}\lg_\beta n$ an epoch. Furthermore, we assume that the claimed data structure satisfies

$t_i(U) = o(\lg_\beta n)$, and our task is to show that the claimed cell set C_i and query set Q exists (see Lemma 2.6).

The first step is to find a geometric property of a set of queries Q, such that $\chi_i(Q)$ is a linearly independent set of vectors. One property that ensures this, is that the queries in Q are sufficiently *well spread*. To make this more formal, we introduce the following terminology:

A *grid* G with *width* $\mu \geq 1$ and *height* $\gamma \geq 1$, is the collection of *grid cells* $[j\mu : (j+1)\mu) \times [h\gamma : (h+1)\gamma)$ such that $0 \leq j < n/\mu$ and $0 \leq h < n/\gamma$. We say that a query point $q = (x, y) \in [n] \times [n]$ *hits* a grid cell $[j\mu : (j+1)\mu) \times [h\gamma : (h+1)\gamma)$ of G, if the point (x, y) lies within that grid cell, i.e. if $j\mu \leq x < (j+1)\mu$ and $h\gamma \leq y < (h+1)\gamma$. Finally, we define the *hitting number* of a set of queries Q' on a grid G, as the number of distinct grid cells in G that is hit by a query in Q'.

With this terminology, we have the following lemma:

Lemma 2.7. *Let Q' be a set of queries and G a grid with width μ and height $n^2/\beta^i\mu$ for some parameter $n/\beta^i \leq \mu \leq n$. Let h denote the hitting number of Q' on G. Then there is a subset of queries $Q \subseteq Q'$, such that $|Q| = \Omega(h - 6n/\mu - 6\mu\beta^i/n)$ and $\chi_i(Q)$ is a linearly independent set of vectors in $[\Delta]^{\beta^i}$.*

We defer the proof of Lemma 2.7 and instead continue the proof of Lemma 2.6.

In light of Lemma 2.7, we set out to find a set of cells $C_i \subseteq D_i(U)$ and a grid G, such that the set of queries Q_{C_i} that probe no cells in $D_i(U) \setminus C_i$, hit a large number of grid cells in G. For this, first define the grids $G_2, \ldots, G_{2i-2}$ where G_j has width $n/\beta^{i-j/2}$ and height $n/\beta^{j/2}$. The existence of C_i is guaranteed by the following lemma:

Lemma 2.8. *Let $i \geq \frac{15}{16}\lg_\beta n$ be an epoch and $U_{\lg_\beta n}, \ldots, U_1$ a fixed sequence of updates, where each U_j is a possible outcome of $\boldsymbol{U}_j$. Assume furthermore that the claimed data structure satisfies $t_i(U) = o(\lg_\beta n)$. Then there exists a set of cells $C_i \subseteq D_i(U)$ and an index $j \in \{2, \ldots, 2i-2\}$, such that $|C_i| = O(\beta^{i-1}w)$ and Q_{C_i} has hitting number $\Omega(\beta^{i-3/4})$ on the grid G_j.*

To not remove focus from the proof of Lemma 2.6 we have moved the proof of this lemma to the end of this section. We thus move on to show that Lemma 2.7 and Lemma 2.8 implies Lemma 2.6. By assumption we have $t_i(U) = o(\lg_\beta n)$. Combining this with Lemma 2.8, we get that there exists a set of cells $C_i \subseteq D_i(U)$ and an index $j \in \{2, \ldots, 2i-2\}$, such that $|C_i| = O(\beta^{i-1}w)$ and the set of queries Q_{C_i} has hitting number $\Omega(\beta^{i-3/4})$ on the grid G_j. Furthermore, we have that grid G_j is a grid of the form required by Lemma 2.7, with $\mu = n/\beta^{i-j/2}$. Thus by Lemma 2.7 there is a subset $Q \subseteq Q_{C_i}$ such that $|Q| = \Omega(\beta^{i-3/4} - 12\beta^{i-1}) = \Omega(\beta^{i-3/4})$ and $\chi_i(Q)$ is a linearly independent set of vectors in $[\Delta]^{\beta^i}$. This completes the proof of Lemma 2.6.

Proof of Lemma 2.7.

We prove the lemma by giving an explicit construction of the set Q.

First initialize Q to contain one query point from Q' from each cell of G that is hit by Q'. We will now repeatedly eliminate queries from Q until the remaining set is linearly independent. We do this by *crossing out* rows and columns of G. By crossing out a row (column) of G, we mean deleting all queries in Q that hits a cell in that row (column). The procedure for crossing out rows and columns is as follows:

First cross out the bottom two rows and leftmost two columns. Amongst the remaining columns, cross out either the even or odd columns, whichever of the two contains the fewest remaining points in Q. Repeat this once again for the columns, with even and odd redefined over the remaining columns. Finally, do the same for the rows. We claim that the remaining set of queries are linearly independent. To see this, order the remaining queries in increasing order of column index (leftmost column has lowest index), and secondarily in increasing order of row index (bottom row has lowest index). Let $q_1, \dots, q_{|Q|}$ denote the resulting sequence of queries. For this sequence, it holds that for every query q_j, there exists a coordinate $\chi_i(q_j)_h$, such that $\chi_i(q_j)_h = 1$, and at the same time $\chi_i(q_k)_h = 0$ for all $k < j$. Clearly this implies linear independence. To prove that the remaining vectors have this property, we must show that for each query q_j, there is some point in the scaled Fibonacci lattice F_{β^i} that is dominated by q_j, but not by any of $q_1, \dots, q_{j-1}$: Associate each remaining query q_j to the two-by-two crossed out grid cells to the bottom-left of the grid cell hit by q_j. These four grid cells have area $4n^2/\beta^i$ and are contained within the rectangle $[0 : n - n/\beta^i] \times [0 : n - n/\beta^i]$, thus from Lemma 2.4 it follows that at least one point of the scaled Fibonacci lattice F_{β^i} is contained therein, and thus dominated by q_j. But all q_k, where $k < j$, either hit a grid cell in a column with index at least three less than that hit by q_j (we crossed out the two columns preceding that hit by q_j), or they hit a grid cell in the same column as q_j but with a row index that is at least three lower than that hit by q_j (we crossed out the two rows preceding that hit by q_j). In either case, such a query cannot dominate the point inside the grid cells associated to q_j.

What remains is to bound the size of Q. Initially, we have $|Q| = h$. The bottom two rows have a total area of $2n^3/\beta^i\mu$, thus by Lemma 2.4 they contain at most $6n/\mu$ points. The leftmost two columns have area $2n\mu$ and thus contain at most $6\mu\beta^i/n$ points. After crossing out these rows and column we are therefore left with $|Q| \geq h - 6n/\mu - 6\mu\beta^i/n$. Finally, when crossing out even or odd rows we always choose the one eliminating fewest points, thus the remaining steps at most reduce the size of Q by a factor 16. This completes the proof of Lemma 2.7.

Proof of Lemma 2.8.

We prove the lemma using another encoding argument. However, this time we do not encode an update sequence, but instead we define a distribution over query sets, such that if Lemma 2.8 is not true, then we can encode such a query set in too few bits.

Let $U = U_{\lg_\beta n}, \dots, U_1$ be a fixed sequence of updates, where each U_j is a possible outcome of $\mathbf{U}_j$. Furthermore, assume for contradiction that the claimed data structure satisfies both $t_i(U) = o(\lg_\beta n)$ and for all cell sets $C \subseteq D_i(U)$ of size $|C| = O(\beta^{i-1}w)$ and every index $j \in \{2, \dots, 2i-2\}$, it holds that the hitting number of Q_C on grid G_j is $o(\beta^{i-3/4})$. Here Q_C denotes the set of all queries q in $[n] \times [n]$ such that the query algorithm of the claimed data structure probes no cells in $D_i(U) \setminus C$ when answering q after the sequence of updates U. Under these assumptions we will construct an impossible encoder. As mentioned, we will encode a set of queries:

Hard Distribution. Let $\mathbf{Q}$ denote a random set of queries, constructed by drawing one uniform random query (with integer coordinates) from each of the β^{i-1} vertical slabs of the form:

$$[hn/\beta^{i-1} : (h+1)n/\beta^{i-1}) \times [0:n),$$

where $h \in [\beta^{i-1}]$. Our goal is to encode $\mathbf{Q}$ in less than $H(\mathbf{Q}) = \beta^{i-1} \lg(n^2/\beta^{i-1})$ bits in expectation. Before giving the encoding and decoding procedures, we prove some simple properties of $\mathbf{Q}$:

Define a query q in a query set Q' to be *well-separated* if for all other queries $q' \in Q'$, where $q \neq q'$, q and q' do not lie within an axis-aligned rectangle of area $n^2/\beta^{i-1/2}$. Finally, define a query set Q' to be *well-separated* if at least $\frac{1}{2}|Q'|$ queries in Q' are well-separated. We then have:

Lemma 2.9. *The query set $\mathbf{Q}$ is well-separated with probability at least* $3/4$.

Proof. Let $\mathbf{q}_h$ denote the random query in $\mathbf{Q}$ lying in the h'th vertical slab. The probability that $\mathbf{q}_h$ lies within a distance of at most $n/\beta^{i-3/4}$ from the x-border of the h'th slab is precisely $(2n/\beta^{i-3/4})/(n/\beta^{i-1}) = 2/\beta^{1/4}$. If this is not the case, then for another query $\mathbf{q}_k$ in $\mathbf{Q}$, we know that the x-coordinates of $\mathbf{q}_h$ and $\mathbf{q}_k$ differ by at least $(|k-h|-1)n/\beta^{i-1} + n/\beta^{i-3/4}$. This implies that $\mathbf{q}_h$ and $\mathbf{q}_k$ can only be within an axis-aligned rectangle of area $n^2/\beta^{i-1/2}$ if their y-coordinates differ by at most $n/((|k-h|-1)\beta^{1/2} + \beta^{1/4})$. This happens with probability at most $2/((|k-h|-1)\beta^{1/2} + \beta^{1/4})$. The probability that a query $\mathbf{q}_h$ in $\mathbf{Q}$ is not well-separated is therefore bounded by

$$\begin{aligned} \frac{2}{\beta^{1/4}} + \left(1 - \frac{2}{\beta^{1/4}}\right) \sum_{k \neq j} \frac{2}{(|k-h|-1)\beta^{1/2} + \beta^{1/4}} &\leq \frac{10}{\beta^{1/4}} + \sum_{k \neq j} \frac{2}{|k-h|\beta^{1/2}} \\ &= O\left(\frac{1}{\beta^{1/4}} + \frac{\lg n}{\beta^{1/2}}\right). \end{aligned}$$

Since $\beta = (wt_u)^9 = \omega(\lg^2 n)$ this probability is $o(1)$, and the result now follows from linearity of expectation and Markov's inequality. $\square$

Now let $D_i(Q,U) \subseteq D_i(U)$ denote the subset of cells in $D_i(U)$ probed by the query algorithm of the claimed data structure when answering all queries in a set of queries Q after the sequence of updates U (i.e. the union of the cells probed for each query in Q). Since a uniform random query from $\mathbf{Q}$ is uniform in $[n] \times [n]$, we get by linearity of expectation that $\mathbb{E}[|D_i(\mathbf{Q},U)|] = \beta^{i-1} t_i(U)$. From this, Lemma 2.9, Markov's inequality and a union bound, we conclude

Lemma 2.10. *The query set $\mathbf{Q}$ is both well-separated and $|D_i(\mathbf{Q},U)| \leq 4\beta^{i-1} t_i(U)$ with probability at least* $1/2$.

With this established, we are now ready to encode $\mathbf{Q}$.

Encoding. In the following we describe the encoding procedure. The encoder receives as input the set of queries $\mathbf{Q}$. He then executes the following procedure:

1. The encoder first executes the fixed sequence of updates U on the claimed data structure, and from this obtains the sets $D_{\lg_\beta n}(U), \dots, D_1(U)$. He then runs the query algorithm for every query $q \in \mathbf{Q}$ and collects the set $D_i(\mathbf{Q}, U)$.

2. If $\mathbf{Q}$ is not well-separated or if $|D_i(\mathbf{Q}, U)| > 4\beta^{i-1}t_i(U)$, then the encoder sends a 1-bit followed by a straightforward encoding of $\mathbf{Q}$ using $H(\mathbf{Q}) + O(1)$ bits in total. This is the complete encoding procedure when either $\mathbf{Q}$ is not well-separated or $|D_i(\mathbf{Q}, U)| > 4\beta^{i-1}t_i(U)$.

3. If $\mathbf{Q}$ is both well-separated and $|D_i(\mathbf{Q}, U)| \leq 4\beta^{i-1}t_i(U)$, then the encoder first writes a 0-bit and then executes the remaining four steps.

4. The encoder examines $\mathbf{Q}$ and finds the at most $\frac{1}{2}|\mathbf{Q}|$ queries that are not well-separated. Denote this set $\mathbf{Q}'$. The encoder now writes down $\mathbf{Q}'$ by first specifying $|\mathbf{Q}'|$, then which vertical slabs contain the queries in $\mathbf{Q}'$ and finally what the coordinates of each query in $\mathbf{Q}'$ is within its slab. This takes $O(w) + \lg \binom{|\mathbf{Q}|}{|\mathbf{Q}'|} + |\mathbf{Q}'| \lg(n^2/\beta^{i-1}) = O(w) + O(\beta^{i-1}) + |\mathbf{Q}'| \lg(n^2/\beta^{i-1})$ bits.

5. The encoder now writes down the cell set $D_i(\mathbf{Q}, U)$, including **only** the addresses and **not** the contents. This takes $o(H(\mathbf{Q}))$ bits since

$$\begin{aligned} \lg \binom{|D_i(U)|}{|D_i(\mathbf{Q}, U)|} &= O(\beta^{i-1}t_i(U)\lg(\beta t_u)) \\ &= o(\beta^{i-1}\lg(n^2/\beta^{i-1})), \end{aligned}$$

   where in the first line we used that $|D_i(U)| \leq \beta^i t_u$ and $|D_i(\mathbf{Q}, U)| \leq 4\beta^{i-1}t_i(U)$. The second line follows from having $t_i(U) = o(\lg_\beta n) = O(\lg(n^2/\beta^{i-1})/\lg(\beta t_u))$ since $\beta = \omega(t_u)$.

6. Next we encode the x-coordinates of the well-separated queries in $\mathbf{Q}$. Since we have already encoded which vertical slabs contain well-separated queries (we really encoded the slabs containing queries that are not well-separated, but this is equivalent), we do this by specifying only the offset within each slab. This takes $(|\mathbf{Q}| - |\mathbf{Q}'|)\lg(n/\beta^{i-1}) + O(1)$ bits. Following that, the encoder considers the last grid G_{2i-2}, and for each well-separated query q, he writes down the y-offset of q within the grid cell of G_{2i-2} hit by q. Since the grid cells of G_{2i-2} have height n/β^{i-1}, this takes $(|\mathbf{Q}| - |\mathbf{Q}'|)\lg(n/\beta^{i-1}) + O(1)$ bits. Combined with the encoding of the x-coordinates, this step adds a total of $(|\mathbf{Q}| - |\mathbf{Q}'|)\lg(n^2/\beta^{2i-2}) + O(1)$ bits to the size of the encoding.

7. In the last step of the encoding procedure, the encoder simulates the query algorithm for every query in $[n] \times [n]$ and from this obtains the set $Q_{D_i(\mathbf{Q},U)}$, i.e. the set of all those queries that probe no cells in $D_i(U) \setminus$

$D_i(\mathbf{Q}, U)$. Observe that $\mathbf{Q} \subseteq Q_{D_i(\mathbf{Q},U)}$. The encoder now considers each of the grids G_j, for $j = 2, \ldots, 2i-2$, and determines both the set of grid cells $G_j^{Q_{D_i(\mathbf{Q},U)}} \subseteq G_j$ hit by a query in $Q_{D_i(\mathbf{Q},U)}$, and the set of grid cells $G_j^{\mathbf{Q}} \subseteq G_j^{Q_{D_i(\mathbf{Q},U)}} \subseteq G_j$ hit by a well-separated query in $\mathbf{Q}$. The last step of the encoding consists of specifying $G_j^{\mathbf{Q}}$. This is done by encoding which subset of $G_j^{Q_{D_i(\mathbf{Q},U)}}$ corresponds to $G_j^{\mathbf{Q}}$. This takes $\lg \binom{|G_j^{Q_{D_i(\mathbf{Q},U)}}|}{|G_j^{\mathbf{Q}}|}$ bits for each $j = 2, \ldots, 2i-2$.

Since $|D_i(\mathbf{Q}, U)| = o(\beta^{i-1} \lg_\beta n) = o(\beta^{i-1} w)$ we get from our contradictory assumption that the hitting number of $Q_{D_i(\mathbf{Q},U)}$ on each grid G_j is $o(\beta^{i-3/4})$, thus $|G_j^{Q_{D_i(\mathbf{Q},U)}}| = o(\beta^{i-3/4})$. Therefore the above amount of bits is at most

$$\begin{aligned}(|\mathbf{Q}| - |\mathbf{Q}'|) \lg(\beta^{i-3/4} e/(|\mathbf{Q}| - |\mathbf{Q}'|))(2i-3) &\leq \\ (|\mathbf{Q}| - |\mathbf{Q}'|) \lg(\beta^{1/4}) 2i + O(\beta^{i-1} i) &\leq \\ (|\mathbf{Q}| - |\mathbf{Q}'|) \tfrac{1}{4} \lg(\beta) 2 \lg_\beta n + O(\beta^{i-1} \lg_\beta n) &\leq \\ (|\mathbf{Q}| - |\mathbf{Q}'|) \tfrac{1}{2} \lg n + o(H(\mathbf{Q})).\end{aligned}$$

This completes the encoding procedure, and the encoder finishes by sending the constructed message to the decoder.

Before analysing the size of the encoding, we show that the decoder can recover $\mathbf{Q}$ from the encoding.

Decoding. In this paragraph we describe the decoding procedure. The decoder only knows the fixed sequence $U = U_{\lg_\beta n}, \ldots, U_1$ and the message received from the encoder. The goal is to recover $\mathbf{Q}$, which is done by the following steps:

1. The decoder examines the first bit of the message. If this is a 1-bit, the decoder immediately recovers $\mathbf{Q}$ from the remaining part of the encoding.

2. If the first bit is 0, the decoder proceeds with this step and all of the below steps. The decoder executes the updates U on the claimed data structure and obtains the sets $D_{\lg_\beta n}(U), \ldots, D_1(U)$. From step 4 of the encoding procedure, the decoder also recovers $\mathbf{Q}'$.

3. From step 5 of the encoding procedure, the decoder now recovers the addresses of the cells in $D_i(\mathbf{Q}, U)$. Since the decoder has the data structure, he already knows the contents. Following this, the decoder now simulates every query in $[n] \times [n]$, and from this and $D_i(\mathbf{Q}, U)$ recovers the set $Q_{D_i(\mathbf{Q},U)}$.

4. From step 6 of the encoding procedure, the decoder now recovers the x-coordinates of every well-separated query in $\mathbf{Q}$ (the offsets are enough since the decoder knows which vertical slabs contain queries in $\mathbf{Q}'$, and thus also those that contain well-separated queries). Following that, the

decoder also recovers the y-offset of each well-separated query $q \in \mathbf{Q}$ within the grid cell of G_{2i-2} hit by q (note that the decoder does not know what grid cell it is, he only knows the offset).

5. From the set $Q_{D_i(\mathbf{Q},U)}$ the decoder now recovers the set $G_j^{Q_{D_i(\mathbf{Q},U)}}$ for each $j = 2, \dots, 2i-2$. This information is immediate from the set $Q_{D_i(\mathbf{Q},U)}$. From $G_j^{Q_{D_i(\mathbf{Q},U)}}$ and step 7 of the encoding procedure, the decoder now recovers $G_j^{\mathbf{Q}}$ for each j. In grid G_2, we know that $\mathbf{Q}$ has only one query in every column, thus the decoder can determine uniquely from $G_2^{\mathbf{Q}}$ which grid cell of G_2 is hit by each well-separated query in $\mathbf{Q}$. Now observe that the axis-aligned rectangle enclosing all $\beta^{1/2}$ grid cells in G_{j+1} that intersects a fixed grid cell in G_j has area $n^2/\beta^{i-1/2}$. Since we are considering well-separated queries, i.e. queries where no two lie within an axis-aligned rectangle of area $n^2/\beta^{i-1/2}$, this means that $G_{j+1}^{\mathbf{Q}}$ contains at most one grid cell in such a group of $\beta^{1/2}$ grid cells. Thus if q is a well-separated query in $\mathbf{Q}$, we can determine uniquely which grid cell of G_{j+1} that is hit by q, directly from $G_{j+1}^{\mathbf{Q}}$ and the grid cell in G_j hit by q. But we already know this information for grid G_2, thus we can recover this information for grid $G_3, G_4, \dots, G_{2i-2}$. Thus we know for each well-separated query in $\mathbf{Q}$ which grid cell of G_{2i-2} it hits. From the encoding of the x-coordinates and the y-offsets, the decoder have thus recovered $\mathbf{Q}$.

Analysis. Finally we analyse the size of the encoding. First consider the case where $\mathbf{Q}$ is both well-separated and $|D_i(\mathbf{Q},U)| \le 4\beta^{i-1}t_i(U)$. In this setting, the size of the message is bounded by

$$|\mathbf{Q}'| \lg(n^2/\beta^{i-1}) + (|\mathbf{Q}| - |\mathbf{Q}'|)(\lg(n^2/\beta^{2i-2}) + \tfrac{1}{2}\lg n) + o(H(\mathbf{Q}))$$

bits. This equals

$$|\mathbf{Q}| \lg(n^{2+1/2}/\beta^{2i-2}) + |\mathbf{Q}'| \lg(\beta^{i-1}/n^{1/2}) + o(H(\mathbf{Q}))$$

bits. Since we are considering an epoch $i \ge \frac{15}{16}\lg_\beta n$, we have $\lg(n^{2+1/2}/\beta^{2i-2}) \le \lg(n^{5/8}\beta^2)$, thus the above amount of bits is upper bounded by

$$|\mathbf{Q}| \lg(n^{5/8}\beta^2) + |\mathbf{Q}'| \lg(n^{1/2}) + o(H(\mathbf{Q})).$$

Since $|\mathbf{Q}'| \le \frac{1}{2}|\mathbf{Q}|$, this is again bounded by

$$|\mathbf{Q}| \lg(n^{7/8}\beta^2) + o(H(\mathbf{Q}))$$

bits. But $H(\mathbf{Q}) = |\mathbf{Q}| \lg(n^2/\beta^i) \ge |\mathbf{Q}| \lg n$, i.e. our encoding uses less than $\frac{15}{16}H(\mathbf{Q})$ bits.

Finally, let E denote the event that $\mathbf{Q}$ is well-separated and at the same time $|D_i(\mathbf{Q},U)| \le 4\beta^{i-1}t_i(U)$, then the expected number of bits used by the entire encoding is bounded by

$$O(1) + \Pr[E](1 - \Omega(1))H(\mathbf{Q}) + (1 - \Pr[E])H(\mathbf{Q}).$$

The contradiction is now reached by invoking Lemma 2.10 to conclude that $\Pr[E] \geq 1/2$. We have thus proved Lemma 2.8, which was the last missing step of our lower bound proof. We hence conclude:

Theorem 2.3. *Any data structure for dynamic weighted orthogonal range counting in the cell probe model, must satisfy $t_q = \Omega((\lg n / \lg(wt_u))^2)$. Here t_q is the expected query time and t_u the worst case update time. This lower bound holds when the weights of the inserted points are $\Theta(\lg n)$-bit integers.*

2.3 Concluding Remarks

In this chapter we presented new techniques for proving both static and dynamic cell probe lower bounds. In both cases, our new techniques allowed us to obtain the highest lower bounds to date.

While our results have taken the field of cell probe lower bounds one step further, there is still a long way to go. Amongst the results that seems within grasp, we find it a very intriguing open problem to prove an $\omega(\lg n)$ dynamic lower bound for a problem where the queries have a one bit output (decision problems). Our new technique crucially relies on the output having more bits than it takes to describe a query, since otherwise the encoder cannot afford to tell the decoder which queries to simulate. Since many interesting data structure problems have a one bit output size, finding a technique for handling this case would allow us to attack many more fundamental data structure problems. As a technical remark, we note that when proving static lower bounds using the cell sampling idea, the encoder does not have to write down the queries to simulate. This is because queries are completely solved from the cell sample and need not read any other cells. Hence the decoder can simply try to simulate the query algorithm for every possible query and simply discard those that read cells outside the sample. In the dynamic case, we still have to read cells associated to other epochs. For the future epochs (small epochs), this is not an issue since we know all such cells. However, when simulating the query algorithm for a query that is not resolved by the sample, i.e. it reads other cells from the epoch we are deriving a contradiction for, we cannot recognize that the query fails. Instead, we will end up using the cell contents written in past epochs and could potentially obtain an incorrect answer for the query and we have no way of recognizing this. We believe that finding a way to circumvent the encoding of queries is the most promising direction for improvements.

Applying our new techniques to other problems is also an important task. However, for the dynamic case, such problems must again have a logarithmic number of bits in the output of queries.

Chapter 3

The Group Model

Our contribution to the field of range searching in the group model is two new techniques for proving lower bounds for oblivious data structures. The first technique establishes a connection between dynamic range searching in the group model and combinatorial discrepancy. We present this result in Section 3.1. Our second technique draws a connection to range reporting in the pointer machine model. This result is presented in Section 3.2.

As mentioned in Section 1.4, our results require a notion of *bounded coefficients*. We elaborate on this in the following: Recall from Section 1.4 that an oblivious data structure preprocesses an input set of n geometric objects into a collection of group elements, each corresponding to a linear combination over the weights assigned to the input objects. Queries are answered by again computing linear combinations over the precomputed group elements, and updates are supported by re-evaluating every linear combination involving the weight of the updated point. We define the *multiplicity* of an oblivious data structure as the largest absolute value occurring as a coefficient in any of these linear combinations. We note that every known data structure uses only coefficients amongst $\{-1, 0, +1\}$, thus all known data structures have multiplicity 1, but there is nothing inherent in the group model that prevents larger coefficients. Both of our new techniques gives lower bounds depending non-trivially on the multiplicity of the data structure. See Section 3.1 and Section 3.2 for details. Also note that we give a more formal and mathematical definition of oblivious data structures and multiplicity in Section 3.3.

3.1 Connection to Combinatorial Discrepancy

In the field of combinatorial discrepancy, the focus lies on understanding set systems. In particular, if $(Y, \mathcal{A})$ is a set system, where $Y = \{1, \dots, n\}$ are the elements and $\mathcal{A} = \{\mathcal{A}_1, \dots, \mathcal{A}_m\}$ is a family of subsets of Y, then the *minimum discrepancy problem* asks to find a 2-coloring $\chi : Y \to \{-1, +1\}$ of the elements in Y, such that each set in $\mathcal{A}$ is colored as evenly as possible, i.e. find χ

minimizing $\mathrm{disc}_\infty(\chi, Y, \mathcal{A})$, where

$$\mathrm{disc}_\infty(\chi, Y, \mathcal{A}) = \max_j \left| \sum_{i \in \mathcal{A}_j} \chi(i) \right| .$$

Understanding the best achievable colorings for various families of set systems has been an active line of research for decades, and the results obtained have found numerous applications in other areas of computer science, see for instance the seminal books of Matoušek [60] and Chazelle [37] for introductions to discrepancy theory, and for applications in complexity lower bounds, computational geometry, pseudo-randomness and communication complexity.

The results most important to our work are those related to families of set systems with a range searching flavor to them. More formally, if we let X be a universe of geometric objects (think of X as all possible input geometric objects to a range searching problem, for instance all points d-dimensional space), $P \subset X$ a set of n geometric input objects $\{p_1, \ldots, p_n\}$ (a concrete input to a range searching problem) and $\mathcal{R}$ a collection of query ranges, where each query range $R \in \mathcal{R}$ is a subset of X (for every query to the range searching problem, $\mathcal{R}$ contains a set consisting of those elements in X that intersects the query range), then we define the *induced set system* $(P, \mathcal{A}_{P,\mathcal{R}})$, where $\mathcal{A}_{P,\mathcal{R}} = \{R \cap P : R \in \mathcal{R}\}$ is the family of sets containing for each $R \in \mathcal{R}$, the set consisting of all input objects that are contained in R ($\mathcal{A}_{P,\mathcal{R}}$ is also known in the literature as the trace of $\mathcal{R}$ on P). With this definition, we define the ℓ_∞-*discrepancy* $\mathrm{disc}_\infty(P, \mathcal{R})$ as

$$\begin{aligned} \mathrm{disc}_\infty(P, \mathcal{R}) &= \min_{\chi : P \to \{-1,+1\}} \mathrm{disc}_\infty(\chi, P, \mathcal{A}_{P,\mathcal{R}}) \\ &= \min_{\chi : P \to \{-1,+1\}} \max_{\mathcal{A}_j \in \mathcal{A}_{P,\mathcal{R}}} \left| \sum_{p_i \in \mathcal{A}_j} \chi(p_i) \right| , \end{aligned}$$

thus the ℓ_∞-discrepancy measures the best achievable 2-coloring of the induced set system of P and $\mathcal{R}$. A similar measure, called the ℓ_2-discrepancy $\mathrm{disc}_2(P, \mathcal{R})$, also plays a key role in our results

$$\mathrm{disc}_2(P, \mathcal{R}) = \min_{\chi : P \to \{-1,+1\}} \sqrt{\frac{1}{|\mathcal{A}_{P,\mathcal{R}}|} \sum_{\mathcal{A}_j \in \mathcal{A}_{P,\mathcal{R}}} \left(\sum_{p_i \in \mathcal{A}_j} \chi(p_i) \right)^2 } .$$

Observe that we always have $\mathrm{disc}_2(P, \mathcal{R}) \leq \mathrm{disc}_\infty(P, \mathcal{R})$. Finally, we define the number of *distinct query ranges* of an induced set system as $|\mathcal{A}_{P,\mathcal{R}}|$, that is, as the number of distinct sets ($\mathcal{A}_{P,\mathcal{R}}$ is not a multiset).

To make absolutely clear the connection to range searching, consider as an example the d-dimensional orthogonal range searching problem. Here X is simply $\mathbb{R}^d$, i.e. the set of all d-dimensional points. The family $\mathcal{R}$ contains all axis-aligned rectangles in $\mathbb{R}^d$ (each axis-aligned rectangle is a subset of $\mathbb{R}^d$). Finally, we see that for any induced set system $(P, \mathcal{A}_{P,\mathcal{R}})$, where P is a set of n input points in X, the number of distinct query ranges is bounded by $O(n^{2d})$ since each axis-aligned rectangle defining a range in $\mathcal{R}$ can be shrunk to have one point from P on each of its $2d$ sides without changing the set of input points contained in the query range.

Previous Results. In the following, we review the discrepancy upper and lower bounds related to the most fundamental types of range searching. We note that a lower bound on the discrepancy of a range searching problem is a proof that *there exists* a subset $P \subset X$ of n input objects to the range searching problem, for which $\text{disc}_\infty(P, \mathcal{R})$ or $\text{disc}_2(P, \mathcal{R})$ is bounded from below, while upper bounds on the discrepancy imply that $\text{disc}_\infty(P, \mathcal{R})$ or $\text{disc}_2(P, \mathcal{R})$ is bounded from above *for all* subsets $P \subset X$ of n input objects. Since $\text{disc}_2(P, \mathcal{R}) \leq \text{disc}_\infty(P, \mathcal{R})$ we also get that lower bounds on the ℓ_2-discrepancy translates directly to lower bounds on the ℓ_∞-discrepancy, and similarly, upper bounds for the ℓ_∞-discrepancy translates to upper bounds for the ℓ_2-discrepancy. Hence when the upper bounds or lower bounds achieved for the two measures match, we only state the strongest result in the following.

The discrepancy of halfspace range searching is particularly well understood. If we let $\mathcal{H}_d$ denote the set of all halfspaces in d-dimensional space (where each halfspace $H \in \mathcal{H}_d$ is a subset of $\mathbb{R}^d$), then Alexander [10] proved that there exists a set P of n points in $\mathbb{R}^d$, such that $\text{disc}_2(P, \mathcal{H}_d) = \Omega(n^{1/2-1/2d})$. A matching upper bound was subsequently established by Matoušek [59], even for the ℓ_∞-discrepancy.

For orthogonal range searching (or axis-aligned rectangles), the picture is more muddy. On the lower bound side, Beck [20] proved that there exists a set P of n points in $\mathbb{R}^2$, such that $\text{disc}_\infty(P, \mathcal{B}_2) = \Omega(\lg n)$, where we use $\mathcal{B}_d$ to denote the family containing all axis-aligned rectangles in $\mathbb{R}^d$. However, in dimensions $d \geq 3$, the highest achieved lower bounds achieved are only $\text{disc}_\infty(P, \mathcal{B}_d) = \Omega(\lg^{(d-1)/2+\mu(d)} n)$, where $\mu(d) > 0$ is some small but strictly positive function of d [25]. For the ℓ_2-discrepancy, the highest lower bound is $\text{disc}_2(P, \mathcal{B}_d) = \Omega(\lg^{(d-1)/2} n)$ [76, 35]. On the upper bound side, Srinivasan [80] proved that $\text{disc}_\infty(P, \mathcal{B}_2) = O(\lg^{5/2} n)$ for any set P of n points in $\mathbb{R}^2$, and for dimensions $d \geq 3$, the best upper bound is $\text{disc}_\infty(P, \mathcal{B}_d) = O(\lg^{d+1/2} n \sqrt{\lg \lg n})$ [60].

If the ranges are balls with arbitrary radius, then an ℓ_2-discrepancy lower bound of $\Omega(n^{1/2-1/2d})$ can be established from the results on halfspace ranges [60] (a large enough ball looks locally like a halfspace). A matching lower bound for the ℓ_∞-discrepancy was proved in the two-dimensional case, even when all balls (discs) have a fixed radius [21].

For line range searching, Chazelle and Lvov proved that there exists a set P of n points in $\mathbb{R}^2$, such that $\text{disc}_\infty(P, \mathcal{L}_2) = \Omega(n^{1/6})$ [39]. Here $\mathcal{L}_2$ denotes the set of all lines in two-dimensional space.

Another interesting lower bound is related to arithmetic progressions. Let $(Y, \mathcal{A})$ be the set system where $Y = \{0, \ldots, n-1\}$ and $\mathcal{A}$ contains every arithmetic progression on Y, i.e. for every pair of integers i, d satisfying $0 \leq i, d < n$, $\mathcal{A}$ contains the set $\mathcal{A}_{i,d} = \{i + jd \mid j \in \{0, \ldots, \lfloor (n-i-1)/d \rfloor\}\}$. Then Roth proved $\text{disc}_2(Y, \mathcal{A}) = \Omega(n^{1/4})$ [77].

Finally, we conclude by mentioning some additional discrepancy upper bounds that are related to the later proofs in this paper. If $(Y, \mathcal{A})$ is a set system in which every $p_i \in Y$ is contained in at most t sets in $\mathcal{A}$, then Banaszczyk [16] proved that $\text{disc}_\infty(Y, \mathcal{A}) = O(\sqrt{t \lg |\mathcal{A}|})$ and $\text{disc}_2(Y, \mathcal{A}) = O(\sqrt{t})$. The best bound for the ℓ_∞-version of this problem, that is independent of $|\mathcal{A}|$ and $|Y|$, is due to Beck and Fiala [22] and it states that $\text{disc}_\infty(Y, \mathcal{A}) = O(t)$. We note that there exist results [23] improving on the additive constants in the bound of Beck and Fiala. While many discrepancy upper bounds are purely existential, we mention that Bansal [17] recently gave constructive discrepancy minimization algorithms for several central problems.

The Connection.

With the definitions of combinatorial discrepancy well established, we are ready to present the connection to range searching in the group model. The connection is very clean and allows us to immediately translate discrepancy lower bounds to data structure lower bounds. We have formulated our result in the following theorem:

Theorem 3.1. *Let $\mathcal{R}$ be the query ranges of a range searching problem, where each set in $\mathcal{R}$ is a subset of a universe X. Furthermore, let $P \subset X$ be a set of n geometric input objects to the range searching problem. Then any oblivious data structure for the range searching problem must satisfy $t_u t_q = \Omega(disc_\infty(P, \mathcal{R})^2/\Delta^4 \lg m)$ on the input set P. Here Δ denotes the multiplicity of the data structure, t_u its worst case update time, t_q its worst case query time and m the number of distinct query ranges in $(P, \mathcal{A}_{P,\mathcal{R}})$, i.e. $m = |\mathcal{A}_{P,\mathcal{R}}|$. For the ℓ_2-discrepancy, any oblivious data structure for the range searching problem must satisfy $t_u t_q = \Omega(disc_2(P, \mathcal{R})^2/\Delta^4)$.*

Thus for constant multiplicity oblivious data structures (which includes all known upper bounds), we get extremely high lower bounds compared to previous results. We mention these lower bounds in the following (for constant multiplicity), and note that the number of distinct query ranges for all of the considered problems is polynomial in the input size (i.e. $\lg m = \Theta(\lg n)$):

For halfspace range searching in d-dimensional space we get a lower bound of

$$t_u t_q = \Omega(n^{1-1/d}),$$

simply by plugging in the ℓ_2-discrepancy lower bound of $\Omega(n^{1/2-1/2d})$. This comes within a $\lg \lg n$ factor of Chan's upper bound, and is exponentially larger than the highest previous lower bound for any explicit problem of $t_q = \Omega((\lg n/\lg(\lg n + t_u))^2)$ (see Section 1.4). We note that halfspace range searching is a special case of simplex range searching, this bound therefore also applies to simplex range searching.

For orthogonal range searching, we do not improve on the best bounds in the two-dimensional case, but for d-dimensional orthogonal range searching we get a lower bound of

$$t_u t_q = \Omega(\lg^{d-1} n),$$

from the ℓ_2-discrepancy lower bound $\Omega(\lg^{(d-1)/2} n)$. By a standard reduction, this bound also applies to the well-studied problem of d-dimensional rectangle stabbing (range searching where the input set contains axis-aligned rectangles, and the queries are points).

For d-dimensional ball range searching, our lower bound matches that for halfspace range searching, and in the two-dimensional case, we get a lower bound of $t_u t_q = \Omega(n^{1/2}/\lg n)$ even when all query balls (discs) have the same fixed radius.

For line range searching, that is, range searching where the input is a set of n two-dimensional points and a query ask to sum the weights of all points intersecting a query line, we get a lower bound of $t_u t_q = \Omega(n^{1/3}/\lg n)$.

Finally, for the arithmetic progression range searching problem, i.e. the range searching problem where the input is a set of n ordered points $p_0, \dots, p_{n-1}$ and a query asks to sum the weights of the points in an arithmetic progression, we get a lower bound of $t_u t_q = \Omega(n^{1/2})$.

For more lower bounds we refer the reader to the books by Matoušek [60] and Chazelle [37].

Our result also has implications for the field of combinatorial discrepancy. By contraposition of Theorem 3.1, we get a discrepancy upper bound for d-dimensional orthogonal range searching (axis-aligned rectangles) of $\mathrm{disc}_\infty(P, \mathcal{B}_d) = O(\lg^{d+1/2} n)$ directly from the textbook range tree data structures with $t_u = t_q = O(\lg^d n)$. While the improvement over the best previous result is only a $\sqrt{\lg \lg n}$ factor in dimensions $d \geq 3$, we still find this a beautiful example of the interplay between data structures and combinatorial discrepancy.

Finally, we mention that our proof of Theorem 3.1 relies on what we believe to be a novel application of discrepancy upper bound techniques.

3.2 Connection to Range Reporting

The second technique we present for proving lower bounds in the group model draws a connection to previous work on lower bounds in the pointer machine model (see Section 1.5). As mentioned in Section 1.5, most pointer machine lower bounds have been proved by constructing *favorable* query sets. We recall the definition of (t, h)-favorable from Section 1.5: Let P be a set of input objects to a range searching problem and let $\mathcal{R}$ be a set of query ranges. Then we say that $\mathcal{R}$ is (t, h)*-favorable* if

1. $|R \cap P| \geq t$ for all $R \in \mathcal{R}$.
2. $|R_1 \cap R_2 \cap \cdots \cap R_h \cap P| = O(1)$ for all sets of h different queries $R_1, \dots, R_h \in \mathcal{R}$.

Favorable query sets were used in combination with the following theorem:

Restatement of Theorem 1.1 (Chazelle and Rosenberg [40])**.** *Let P be a set of n input objects to a range searching problem and $\mathcal{R}$ a set of m query ranges. If $\mathcal{R}$ is $(\Omega(t_q), h)$-favorable, then any pointer machine data structure for P with query time $t_q + O(k)$ must use space $\Omega(mt_q/h)$.*

Since favorable query sets play an important role in our group model results, we also mention a number of previous favorable query set constructions. All these results were used to prove pointer machine range reporting lower bounds by invoking Theorem 1.1. Chazelle [34] (or alternatively [35]), showed that one can construct a $(t, 2)$-favorable set of $\Theta(n/t \cdot (\lg n/\lg t)^{d-1})$ queries for orthogonal range reporting in d-dimensional space. Henceforth, $t = \Omega(1)$ is an adjustable parameter. For the "dual" problem of d-dimensional *rectangle stabbing* (the input is n axis-aligned rectangles and a query asks to report all rectangles containing a query point), Afshani et al. [5] showed that one can construct a $(t, 2)$-favorable set of $n/t \cdot 2^{\Theta\left(\frac{\lg n}{t^{1/(d-1)}}\right)}$ queries. For *line range reporting* (the input consists of n two-dimensional points and a query asks to report all points on a query line), a classical construction of Erdös (see e.g. [67]) shows that one can construct a $(\Theta(n^{1/3}), 2)$-favorable set of n query lines. Finally, for *convex polytope intersection reporting* in $\mathbb{R}^3$ (the input is an n-vertex convex polytope and a query asks to report all edges of the polytope intersecting a query plane), Chazelle and Liu [38] showed that one can construct a $(t, 2)$-favorable set of $\Theta(n^2/t^3)$ query planes.

The Connection.

Our second main result in the group model is an alternative to the connection to discrepancy theory. This relation allows us to almost immediately translate range reporting lower bounds in the pointer machine to dynamic group model lower bounds for data structures with bounded multiplicity. More specifically, let P be a set of n input objects to a range searching problem and $\mathcal{R}$ a set of m query ranges. We say that $\mathcal{R}$ is *strongly* (t, h)*-favorable* for P if

1. $|R \cap P| = \Theta(t)$ for all $R \in \mathcal{R}$.
2. $|R_1 \cap R_2 \cap \cdots \cap R_h \cap P| = O(1)$ for all sets of h different queries $R_1, \ldots, R_h \in \mathcal{R}$.
3. $|\{R \in \mathcal{R} \mid p \in R\}| = O(mt/n)$ for all $p \in P$.

Thus a favorable query set is strongly favorable, if in addition, all query ranges contain roughly equally many input objects and all input objects are contained in roughly equally many query ranges. With this definition, we prove the following result:

Theorem 3.2. *Let P be a set of n input objects to a range searching problem and $\mathcal{R}$ a set of $m \leq n$ query ranges. If $\mathcal{R}$ is strongly $(t, 2)$-favorable for P, then any oblivious data structure for the range searching problem must have $t_q t_u = \Omega(m^2 t / n^2 \Delta^4)$ on the input set P. Here t_q denotes the worst case query time, t_u the worst case update time and Δ the multiplicity of the data structure. For $m = \Theta(n)$ and $\Delta = O(1)$, this bound simplifies to $t_q t_u = \Omega(t)$.*

Fortunately, all the favorable query sets described earlier are also strongly favorable. By adjusting the parameter t (in the favorable query set constructions) such that the number of queries in the favorable query sets is $m = \Theta(n)$ and $m \leq n$, we immediately obtain the following lower bounds (listed here for constant multiplicity):

For d-dimensional orthogonal range searching, we get a lower bound of

$$t_q t_u = \Omega((\lg n / \lg \lg n)^{d-1}).$$

This is slightly weaker than the bound obtained through discrepancy, but we still find it an interesting example of the connection. For line range searching, we get a lower bound of $t_q t_u = \Omega(n^{1/3})$, which is an improvement of a $\lg n$ factor over the result obtained using the discrepancy connection. Additionally, we get a lower bound for convex polytope intersection searching in $\mathbb{R}^3$ of $t_q t_u = \Omega(n^{1/3})$ using the result of Chazelle and Liu [38].

Our proof of Theorem 3.2 is based on carefully bounding the eigenvalues of the incidence matrix corresponding to P and $\mathcal{R}$. In fact, we prove the following stronger theorem during our establishment of Theorem 3.2:

Theorem 3.3. *Let P be a set of n input objects to a range searching problem, $\mathcal{R}$ a set of m query ranges over P and A the corresponding incidence matrix. Then for every $3 \leq k \leq n$, any oblivious data structure for the range searching problem must have $t_q t_u = \Omega(\lambda_k k^2/mn\Delta^4)$ on the input set P. Here λ_k denotes the k'th largest eigenvalue of $A^T A$, t_q the worst case query time of the data structure, t_u the worst case update time and Δ the multiplicity of the data structure.*

This theorem can be considered a complement to Chazelle's theorem [36] for establishing lower bounds on offline range searching in the group model (see Section 1.4), however our dependence on λ_k is exponentially better than the one of Chazelle. Using this theorem, we also obtain a lower bound of $t_q t_u = n^{\Omega(1/\lg\lg n)}$ for orthogonal range searching in non-constant dimension $d = \Omega(\lg n/\lg\lg n)$ (see Section 3.4.3 for a proof).

3.3 Preliminaries

In the following we define range searching, oblivious data structures and discrepancy in terms of matrices.

Incidence Matrices. Let $(P, \mathcal{A}_{P,\mathcal{R}})$ be the induced set system of a set P of n geometric objects $\{p_1, \ldots, p_n\}$ and a family $\mathcal{R}$ of query ranges. Then we define the *incidence matrix* $\mathcal{C}_{P,\mathcal{R}} \in \{0,1\}^{|\mathcal{A}_{P,\mathcal{R}}| \times n}$ of $\mathcal{R}$ and P as the $\{0,1\}$-matrix having a column for each input object in P and a row for each set in the induced set system $\mathcal{A}_{P,\mathcal{R}} = \{\mathcal{A}_1, \ldots, \mathcal{A}_{|\mathcal{A}_{P,\mathcal{R}}|}\}$. The i'th row of $\mathcal{C}_{P,\mathcal{R}}$ has a 1 in the j'th column if $p_j \in \mathcal{A}_i$ and a 0 otherwise.

Oblivious Data Structures. Consider a range searching problem where the query ranges $\mathcal{R}$ are subsets of a universe X. Then an oblivious data structure for the range searching problem is a factorization of each incidence matrix $\mathcal{C}_{P,\mathcal{R}}$, where $P \subset X$ is a set of n geometric input objects, into two matrices $Q_{P,\mathcal{R}}$ and $D_{P,\mathcal{R}}$ such that $Q_{P,\mathcal{R}} \cdot D_{P,\mathcal{R}} = \mathcal{C}_{P,\mathcal{R}}$ [43].

The *data matrix* $D_{P,\mathcal{R}} \in \mathbb{Z}^{S \times n}$ represents the precomputed group sums stored by the data structure on input P. Each of the S rows is interpreted as a linear combination over the weights assigned to the n input objects, and we think of the data structure as maintaining the corresponding group sums when given an assignment of weights to the input objects.

The *query matrix* $Q_{P,\mathcal{R}} \in \mathbb{Z}^{|\mathcal{A}_{P,\mathcal{R}}| \times S}$ specifies the query algorithm. It has one row for each set $\mathcal{A}_i$ in the induced set system $\mathcal{A}_{P,\mathcal{R}}$, and we interpret this row as a linear combination over the precomputed group sums, denoting which elements to add and subtract when answering a query range intersecting precisely the input objects in $\mathcal{A}_i$.

With the above interpretations of the data and query matrix, we get that $Q_{P,\mathcal{R}} \cdot D_{P,\mathcal{R}} = \mathcal{C}_{P,\mathcal{R}}$ ensures that when given a query range $R \in \mathcal{R}$, the query algorithm adds and subtracts a subset of the precomputed linear combinations to finally yield the linear combination summing precisely the weights assigned to the input objects intersecting R. For a concrete example of what the matrices corresponding to a data structure might look like, we refer the reader to Section 3.4.2 where we review the classic data structure solution for orthogonal range searching.

The *worst case query time* of an oblivious data structure on an input set P, is defined as the maximum number of non-zero entries in a row of $Q_{P,\mathcal{R}}$. The *worst case update time* on an input set P is similarly defined as the maximum number of non-zero entries in a column of $D_{P,\mathcal{R}}$. The *space* of the data structure is the number of columns in $Q_{P,\mathcal{R}}$ (equivalently number of rows in $D_{P,\mathcal{R}}$). Finally, we define the *multiplicity* as the largest absolute value of an entry in $D_{P,\mathcal{R}}$ and $Q_{P,\mathcal{R}}$.

Combinatorial Discrepancy and Matrices. The definitions of discrepancy can also be stated in terms of matrices. Let P be a set of n geometric input objects and $\mathcal{R}$ a family of query ranges. Then

$$\mathrm{disc}_\infty(P,\mathcal{R}) = \min_{x \in \{-1,+1\}^n} \|\mathcal{C}_{P,\mathcal{R}} \cdot x\|_\infty$$

$$\mathrm{disc}_2(P,\mathcal{R}) = \min_{x \in \{-1,+1\}^n} \frac{1}{\sqrt{m}} \|\mathcal{C}_{P,\mathcal{R}} \cdot x\|_2$$

are easily seen to be the exact same definition of $\mathrm{disc}_\infty(P,\mathcal{R})$ and $\mathrm{disc}_2(P,\mathcal{R})$ as the ones presented in the introduction. Here m denotes the number of rows in $\mathcal{C}_{P,\mathcal{R}}$, $\|\cdot\|_\infty$ gives the ℓ_∞-norm of a vector (largest absolute value amongst the coordinates) and $\|\cdot\|_2$ gives the ℓ_2-norm of a vector.

3.4 Establishing the Connections

In this section we give the proofs of our three main theorems for proving group model lower bounds, namely Theorem 3.1, Theorem 3.2 and Theorem 3.3. We start by establishing the connection to discrepancy.

3.4.1 Combinatorial Discrepancy

In this section we prove our first main result, Theorem 3.1. We have restated the theorem here for ease of reference:

Restatement of Theorem 3.1. *Let $\mathcal{R}$ be the query ranges of a range searching problem, where each set in $\mathcal{R}$ is a subset of a universe X. Furthermore, let $P \subset$*

X be a set of n geometric input objects to the range searching problem. Then any oblivious data structure for the range searching problem must satisfy $t_u t_q = \Omega(disc_\infty(P,\mathcal{R})^2/\Delta^4 \lg m)$ on the input set P. Here Δ denotes the multiplicity of the data structure, t_u its worst case update time, t_q its worst case query time and m the number of distinct query ranges in $(P, \mathcal{A}_{P,\mathcal{R}})$, i.e. $m = |\mathcal{A}_{P,\mathcal{R}}|$. For the ℓ_2-discrepancy, any oblivious data structure for the range searching problem must satisfy $t_u t_q = \Omega(disc_2(P,\mathcal{R})^2/\Delta^4)$.

Let $\mathcal{R}$ be a collection of query ranges, all subsets of a universe X. Also let $P \subset X$ be a set of n geometric input objects. Our goal is to show that $\mathcal{C}_{P,\mathcal{R}}$ cannot be factored into two matrices $Q_{P,\mathcal{R}}$ and $D_{P,\mathcal{R}}$, unless $Q_{P,\mathcal{R}}$ has a row with many non-zero entries, or $D_{P,\mathcal{R}}$ has a column with many non-zero entries, i.e. either the query or update time of an oblivious data structure for the input set P must be high.

Our key idea for proving this, is to multiply a factorization by a cleverly chosen vector. More formally, if $Q_{P,\mathcal{R}} \cdot D_{P,\mathcal{R}} = \mathcal{C}_{P,\mathcal{R}}$ is a factorization provided by the oblivious data structure, then we find a vector $x \in \mathbb{R}^n$ such that $Q_{P,\mathcal{R}} \cdot D_{P,\mathcal{R}} \cdot x$ has small coefficients if $Q_{P,\mathcal{R}}$ and $D_{P,\mathcal{R}}$ are too sparse, and at the same time $\mathcal{C}_{P,\mathcal{R}} \cdot x$ has large coefficients. Since $Q_{P,\mathcal{R}} \cdot D_{P,\mathcal{R}} \cdot x = \mathcal{C}_{P,\mathcal{R}} \cdot x$ this gives us our lower bound. The trick in finding a suitable vector x is to consider vectors in $\{-1,+1\}^n$. Making this restriction immediately allows us to use combinatorial discrepancy lower bounds to argue that $\mathcal{C}_{P,\mathcal{R}} \cdot x$ has large coefficients, and at the same time we can use combinatorial discrepancy upper bound techniques to exploit the sparse rows and columns of $Q_{P,\mathcal{R}}$ and $D_{P,\mathcal{R}}$.

Proof of Theorem 3.1. Let $\mathcal{R}$ be the query ranges of the range searching problem and P a set of n geometric input objects. Furthermore, let m denote the number of distinct query ranges in $(P, \mathcal{A}_{P,\mathcal{R}})$, and assume that an oblivious data structure for the input set P exists, having worst case update time t_u, worst case query time t_q and multiplicity Δ.

Let $Q_{P,\mathcal{R}} \cdot D_{P,\mathcal{R}} = \mathcal{C}_{P,\mathcal{R}}$ denote the corresponding factorization provided by the oblivious data structure. Our first step is to argue that there exists a vector $x \in \{-1,+1\}^n$ for which $\|Q_{P,\mathcal{R}} \cdot D_{P,\mathcal{R}} \cdot x\|_\infty$ (or $\|Q_{P,\mathcal{R}} \cdot D_{P,\mathcal{R}} \cdot x\|_2$) is small. The existence of this vector is guaranteed by the following theorem:

Theorem 3.4. *Let $Q \in \mathbb{R}^{m \times p}$ and $D \in \mathbb{R}^{p \times n}$ be two matrices of reals, such that every row of Q has at most t_Q non-zero entries, and every column of D has at most t_D non-zero entries. Finally, let Δ be an upper bound on the absolute value of any entry in Q and D. Then, for the ℓ_∞-norm, there exists a vector $x \in \{-1,+1\}^n$, such that $\|QDx\|_\infty = O(\Delta^2 \sqrt{t_D t_Q \lg m})$. For the ℓ_2-norm, there exists a vector $x \in \{-1,+1\}^n$, such that $\|QDx\|_2 = O(\Delta^2 \sqrt{t_D t_Q m})$.*

Before proving the theorem, we show that it implies Theorem 3.1. Recall that all coefficients in $Q_{P,\mathcal{R}}$ and $D_{P,\mathcal{R}}$ are bounded in absolute value by the multiplicity Δ of the oblivious data structure. At the same time, each row of $Q_{P,\mathcal{R}}$ has at most t_q non-zero entries, and each column of $D_{P,\mathcal{R}}$ has at most t_u non-zero entries. Finally, since $(P, \mathcal{A}_{P,\mathcal{R}})$ has at most m distinct query ranges, we get that $Q_{P,\mathcal{R}}$ has at most m rows. Thus by Theorem 3.4, there must exist a vector $x^* \in \{-1,+1\}^n$ such that $\|Q_{P,\mathcal{R}} \cdot D_{P,\mathcal{R}} \cdot x^*\|_\infty = O(\Delta^2 \sqrt{t_u t_q \lg m})$. Since $x^* \in \{-1,+1\}^n$, we also have $\|\mathcal{C}_{P,\mathcal{R}} \cdot x^*\|_\infty \geq \mathrm{disc}_\infty(P,\mathcal{R})$. But $Q_{P,\mathcal{R}} \cdot D_{P,\mathcal{R}} \cdot x^* = \mathcal{C}_{P,\mathcal{R}} \cdot x^*$, thus it must hold that $\Delta^2 \sqrt{t_u t_q \lg m} = \Omega(\mathrm{disc}_\infty(P,\mathcal{R}))$. For the ℓ_2-norm, we similarly get that there exists a vector $x^* \in \{-1,+1\}^n$ such that $\|Q_{P,\mathcal{R}} \cdot D_{P,\mathcal{R}} \cdot x^*\|_2 = O(\Delta^2 \sqrt{t_u t_q m})$. At the same time, we have by the definition of ℓ_2-discrepancy that $\|\mathcal{C}_{P,\mathcal{R}} \cdot x^*\|_2 \geq \sqrt{m} \cdot \mathrm{disc}_2(P,\mathcal{R})$ which completes the proof of Theorem 3.1.

What remains is to prove Theorem 3.4.

Proof of Theorem 3.4.

This section is devoted to proving our main technical result, Theorem 3.4. Throughout the section, we let $Q \in \mathbb{R}^{m \times p}$ and $D \in \mathbb{R}^{p \times n}$ be matrices satisfying the constraints of Theorem 3.4. The main tool in our proof is a result in discrepancy theory due to Banaszczyk [16]. We first introduce some terminology and then present his result.

Let B_2^p denote the closed Euclidean unit ball in $\mathbb{R}^p$. Let γ_p denote the (standard) p-dimensional Gaussian measure on $\mathbb{R}^p$ with density $(2\pi)^{-p/2} e^{-\|x\|_2^2/2}$. Then the following holds

Theorem 3.5 (Banaszczyk [16])**.** *There is a numerical constant $c > 0$ with the following property. Let K be a convex body in $\mathbb{R}^p$ with $\gamma_p(K) \geq 1/2$. Then to each sequence $u_1, \ldots, u_n \in cB_2^p$ there correspond signs $\varepsilon_1, \ldots, \varepsilon_n \in \{-1,+1\}$ such that $\varepsilon_1 u_1 + \cdots + \varepsilon_n u_n \in K$.*

To prove Theorem 3.4, we seek a column vector $x \in \{-1,+1\}^n$ that somehow simultaneously exploits the sparse rows of Q and the sparse columns of D. We argue for the existence of this vector by carefully defining a convex body capturing the sparsity of Q, and a sequence of vectors in cB_2^p capturing the sparsity of D. The application of Theorem 3.5 on this choice of convex body and vectors in cB_2^p then yields the desired vector x. We define two different convex bodies for the ℓ_∞ and the ℓ_2 bounds.

Convex Body for the ℓ_∞-norm. For the ℓ_∞ result, we define the following convex body K_α in $\mathbb{R}^p$:

$$K_\alpha \quad := \quad \{y = (y_1, \ldots, y_p) \in \mathbb{R}^p \mid |\langle Q_1, y\rangle| \leq \alpha \wedge \cdots \wedge |\langle Q_m, y\rangle| \leq \alpha\},$$

where Q_i denotes the i'th row vector of Q, $\langle Q_i, y\rangle = \sum_{j=1}^p Q_{i,j} y_j$ is the standard inner product, and $\alpha \geq 0$ is a parameter to be fixed later. This body is clearly convex since each constraint $|\langle Q_i, y\rangle| \leq \alpha$ corresponds to the intersection of two halfspaces. Therefore K_α is the intersection of $2m$ halfspaces, i.e. K_α is convex.

In understanding our choice of K_α, think of each coordinate in $\mathbb{R}^p$ as representing a coordinate of Dx. In this setting, each of the constraints $|\langle Q_i, y\rangle| \leq \alpha$ intuitively forces the coordinates of QDx to be small. Our goal is to apply Theorem 3.5 on K_α, thus we find a value of α such that $\gamma_p(K_\alpha) \geq 1/2$:

Lemma 3.1. *If $\alpha = \Omega(\Delta \sqrt{t_Q \lg m})$, then $\gamma_p(K_\alpha) \geq 1/2$.*

Proof. Recall that $\gamma_p(K_\alpha)$ denotes the probability that a random vector $z \in \mathbb{R}^p$, with each coordinate distributed independently as a Gaussian with mean 0 and variance 1, lies within K_α. In computing $\Pr[z \in K_\alpha]$, we first bound $\Pr[|\langle Q_i, z\rangle| > \alpha]$ for a fixed i. Since each row Q_i has at most t_Q non-zero entries, we get that $\langle Q_i, z\rangle$ is a linear combination of at most t_Q independent Gaussians, each with mean 0 and variance 1. Furthermore, each coefficient in the linear combination is bounded by Δ in absolute value, thus $\langle Q_i, z\rangle$ is itself Gaussian with mean 0 and variance $\sigma_i^2 \leq t_Q\Delta^2$. By standard tail bounds for Gaussian distributions, we get that $\Pr[|\langle Q_i, z\rangle| > \alpha] = e^{-O(\alpha^2/\sigma_i^2)}$. Setting $\alpha = \Omega(\Delta\sqrt{t_Q \lg m}) = \Omega(\sigma_i\sqrt{\lg m})$ this is less than $1/m^2$. By a union bound over all m constraints in the definition of K_α, we conclude that $\Pr[z \notin K_\alpha] < 1/m$, i.e. $\gamma_p(K_\alpha) > 1 - 1/m > 1/2$. $\square$

Convex Body for the ℓ_2-norm. For the ℓ_2 result, we define the convex body:

$$C_\alpha := \{y \in \mathbb{R}^p \mid \|Qy\|_2 \leq \alpha\}.$$

To see that this body is convex, let $y_1, y_2 \in C_\alpha$. We must show that $ty_1 + (1-t)y_2 \in C_\alpha$ for any $t \in [0:1]$. But $\|Q(ty_1+(1-t)y_2)\|_2 = \|Qty_1+(1-t)Qy_2\|_2 \leq \|Qty_1\|_2+\|(1-t)Qy_2\|_2 \leq \alpha$. Again, we find a value of α such that $\gamma_p(C_\alpha) \geq 1/2$:

Lemma 3.2. *If $\alpha = \Omega(\Delta\sqrt{mt_Q})$, then $\gamma_p(C_\alpha) \geq 1/2$.*

Proof. Again let $z \in \mathbb{R}^p$ be a random vector with each coordinate distributed independently as a Gaussian with mean 0 and variance 1. We want to bound $\Pr[z \in C_\alpha]$. For this, we first prove a bound on $\mathbb{E}[\langle Q_i, z\rangle^2]$. From the arguments above, we have that $\langle Q_i, z\rangle$ is a Gaussian with mean 0 and variance $\sigma_i^2 \leq t_Q\Delta^2$. It follows that $\mathbb{E}[\langle Q_i, z\rangle^2] \leq t_Q\Delta^2$. By linearity of expectation, we have

$$\mathbb{E}\left[\sum_i \langle Q_i, z\rangle^2\right] = mt_Q\Delta^2.$$

From Markov's inequality it follows that $\sum_i \langle Q_i, z\rangle^2 \leq 2mt_Q\Delta^2$ with probability at least 1/2. This implies $\|Qz\|_2 \leq \sqrt{2mt_Q}\Delta$ with probability at least 1/2, which completes the proof. $\square$

Sequence of Vectors. We are now ready to define a sequence of vectors in cB_2^p and apply Theorem 3.5. Letting D_j denote the j'th column vector of D, we define the vectors $d_1, \ldots, d_n$, where $d_j = c/(\Delta\sqrt{t_D}) \cdot D_j$, and $c > 0$ is the constant in Theorem 3.5. Since each column of D has at most t_D non-zero entries, each bounded by Δ in absolute value, we get that $\|D_j\|_2 \leq \sqrt{t_D}\Delta$ for all j, and thus $d_1, \ldots, d_n \in cB_2^p$.

For the ℓ_∞ result, let $\alpha = \Theta(\Delta\sqrt{t_Q \lg m})$. We now get by Theorem 3.5, that there exist signs $\varepsilon_1, \ldots, \varepsilon_n \in \{-1, +1\}$ such that $\sum_{j=1}^n \varepsilon_j d_j \in K_\alpha$. Now define the vector $x = (\varepsilon_1, \ldots, \varepsilon_n)$. We claim that $\|QDx\|_\infty = O(\Delta^2\sqrt{t_D t_Q \lg m})$. To see this, note that $Dx = \sum_{j=1}^n \varepsilon_j D_j = c^{-1}\Delta\sqrt{t_D}\sum_{j=1}^n \varepsilon_j d_j$. Now consider the i'th coordinate of QDx. This coordinate is given by the inner product

$\langle Q_i, Dx\rangle = c^{-1}\Delta\sqrt{t_D}\langle Q_i, \sum_{j=1}^n \varepsilon_j d_j\rangle$. But since $\sum_{j=1}^n \varepsilon_j d_j \in K_\alpha$, this is by definition of K_α bounded in absolute value by $c^{-1}\Delta\sqrt{t_D}\alpha = O(\Delta^2\sqrt{t_D t_Q \lg m})$.

For the ℓ_2 result, let $\alpha = \Theta(\Delta\sqrt{m t_Q})$. We again get from Theorem 3.5, that there exist signs $\varepsilon_1, \dots, \varepsilon_n \in \{-1, +1\}$ such that $\sum_{j=1}^n \varepsilon_j d_j \in C_\alpha$. By similar arguments as above, it follows that the vector $x = (\varepsilon_1, \dots, \varepsilon_n)$ satisfies $\|QDx\|_2 = O(\Delta^2\sqrt{t_D t_Q m})$. This concludes the proof of Theorem 3.4.

3.4.2 Implications for Combinatorial Discrepancy

Having established Theorem 3.1, we now get to all of the immediate implications. To not waste space on repeating ourselves, we refer the reader to the list of results presented in Section 3.1 for an overview of the range searching lower bounds achieved (they follow directly by plugging in the discrepancy lower bounds from Section 3.1 in Theorem 3.1). Thus for the remainder of the section, we present our combinatorial discrepancy upper bound for orthogonal range searching (axis-aligned rectangles).

Combinatorial Discrepancy Upper Bounds. In this section we review the classic data structure solution to orthogonal range searching [24]. We give a rather thorough review to also make clear the translation of a data structure into matrix factorization. We finally summarize the implications of combining the solution with Theorem 3.1.

1-d Orthogonal Range Searching. We set out in the one-dimensional case. Here the input to orthogonal range searching is a set P of n points on the real line, and the goal is to support computing the group sum of the weights assigned to the input points intersecting a query interval. This problem clearly includes the partial sums problem as a special case.

The standard solution to this problem, is to construct a complete binary tree $\mathcal{T}$ over the input points ordered by their coordinates. Each leaf of $\mathcal{T}$ is associated to one input point, and each internal node v is associated to the range of points associated to the leaves of the subtree rooted at v. The data structure stores one group sum for each node in $\mathcal{T}$. The group sum stored for a node v is simply the sum of the weights assigned to the input points associated to v.

Let $[x_1 : x_2]$ be a range query. To answer the range query, we first find the two leaves v_1 and v_2 containing the successor of x_1 and the predecessor of x_2, respectively. Let u denote the lowest common ancestor of v_1 and v_2. We now traverse the path from the left child of u to v_1, and for each node w that is a right child of a node on this path, but not itself on the path, we add up the group sum stored at w. We then do the same with v_1 replaced by v_2 and the roles of left and right reversed, and finally we add up the group sums stored at v_1 and v_2. This is easily seen to sum precisely the weights of the points with a coordinate in the range $[x_1 : x_2]$.

Since the height of $\mathcal{T}$ is $O(\lg n)$, we get that the data structure answers queries in $t_q = O(\lg n)$ group operations. The weight of each input point p is associated to one node at each level of the tree, namely the ancestor nodes of the leaf that is associated to p. Thus the update time is also $t_u = O(\lg n)$, since an update consists of re-evaluating the group sums stored in these nodes.

For completeness, we also sketch what the matrices $Q_{P,\mathcal{R}}$ and $D_{P,\mathcal{R}}$ look like for this data structure. $D_{P,\mathcal{R}}$ has one row for each node in $\mathcal{T}$ and one column for each input point. The row corresponding to a node v has a 1 in the column corresponding to a point p if v is associated to p, and a 0 otherwise. $Q_{P,\mathcal{R}}$ has a column for each node in $\mathcal{T}$ (i.e. for each stored group sum). Furthermore, if $p_1, \dots, p_n$ denotes the input points ordered by their coordinate, then $Q_{P,\mathcal{R}}$ has one row for every pair of points p_i and p_j, $(i \le j)$. For the row corresponding to a pair p_i and p_j, let $[x_i : x_j]$ denote a query range containing precisely the coordinates of the points $p_i, \dots, p_j$. Then that row has a 1 in each column corresponding to a node for which the stored group sum is added when executing the above query algorithm on $[x_i : x_j]$, and a 0 elsewhere.

Higher Dimensions. The above data structure is easily extended to higher dimensions: Construct the one-dimensional layout on the last coordinate of the input points, i.e. construct a complete binary tree over the sorted list of points. Each node in the tree no longer maintains the group sum of the weights assigned to points in the subtree, but instead stores a $(d-1)$-dimensional data structure on the projection of the points in the subtree onto the first $(d-1)$ dimensions.

A query range $[x_1 : x_2] \times \cdots \times [x_{2d-1} : x_{2d}]$ is answered analogous to the one-dimensional approach, except that whenever the one-dimensional data structure adds up the group sum stored in a node, we instead project the query range onto the first $d-1$ dimensions, and ask the resulting query to the $(d-1)$-dimensional data structure stored in that node. Since each queried $(d-1)$-dimensional data structure is implemented only on points with a d'th coordinate inside $[x_{2d-1} : x_{2d}]$, this correctly computes the answer to the query range.

It is easily seen that the weight of a point is included in $O(\lg^d n)$ stored group sums, thus the update time of this data structure is $t_u = O(\lg^d n)$. The query algorithm similarly adds up $t_q = O(\lg^d n)$ stored group sums. Finally, we observe that this data structure has multiplicity 1 since the corresponding matrix factorizations use only coefficients amongst $\{0, 1\}$. By contraposition of Theorem 3.1 we thus conclude

Corollary 3.1. *For any set P of n points in $\mathbb{R}^d$, it holds that $disc_\infty(P, \mathcal{B}^d) = O(\lg^{d+1/2} n)$, where $\mathcal{B}^d$ denotes the set of all axis-aligned rectangles in $\mathbb{R}^d$.*

3.4.3 Range Reporting

As we saw in Section 3.4.1, proving lower bounds for oblivious data structures boils down to arguing when an incidence matrix cannot be factored into two sparse matrices Q and D. The key insight in this section is that for sparse matrices Q and D, with bounded coefficients, the product QD must have small singular values (i.e. $(QD)^T QD$ has small eigenvalues). Thus if A has large singular values, Q and D cannot be sparse if $QD = A$. This is precisely the intuition behind our proof of Theorem 3.3. We have restated the theorem here for convenience:

Restatement of Theorem 3.3. *Let P be a set of n input objects to a range searching problem, $\mathcal{R}$ a set of m query ranges over P and A the corresponding incidence matrix. Then for every $3 \leq k \leq n$, any oblivious data structure for the range searching problem must have $t_q t_u = \Omega(\lambda_k k^2/mn\Delta^4)$ on the input set P. Here λ_k denotes the k'th largest eigenvalue of $A^T A$, t_q the worst case query time of the data structure, t_u the worst case update time and Δ the multiplicity of the data structure.*

Proof of Theorem 3.3. Let P, $\mathcal{R}$ and A be as in Theorem 3.3. Furthermore, let $QD = A$ be the factorization of A provided by an oblivious data structure, where Q is an $m \times S$ matrix such that each row has at most t_q non-zero entries and where D is an $S \times n$ matrix where each column has at most t_u non-zero entries. Finally, let Δ be the multiplicity of the oblivious data structure, i.e. any coefficient in Q and D is bounded in absolute value by Δ. Now let $U(D)\Sigma(D)V(D)^T$ be the singular value decomposition of D. Here $U(D)$ and $V(D)$ are unitary matrices and $\Sigma(D)$ is a diagonal matrix where the diagonal entries equals the singular values of D, i.e. if we let $\gamma_i(D^T D) \geq 0$ denote the i'th largest eigenvalue of the $n \times n$ positive semi-definite matrix $D^T D$, then the i'th diagonal entry of $\Sigma(D)$ is $\sigma_{i,i}(D) = \sqrt{\gamma_i(D^T D)}$. Similarly, let $U(Q)\Sigma(Q)V(Q)^T$ be the singular value decomposition of Q. Letting $\gamma_i(Q^T Q) \geq 0$ denote the i'th largest eigenvalue of $Q^T Q$, we have that the i'th diagonal entry of $\Sigma(Q)$ is $\sigma_{i,i}(Q) = \sqrt{\gamma_i(Q^T Q)}$. Letting $d_{i,j}$ denote entry (i,j) in D, it now follows from $D^T D$ being square and real that

$$\sum_i \gamma_i(D^T D) = \operatorname{tr}(D^T D) = \sum_{i,j} d_{i,j}^2 \leq t_u \Delta^2 n,$$

where we used that the coefficients of D are bounded in absolute value by Δ. Similarly, we have $\sum_i \gamma_i(Q^T Q) \leq t_q \Delta^2 m$. Finally since $\gamma_i(D^T D)$ and $\gamma_i(Q^T Q)$ are non-negative for all i, we conclude that $\gamma_{\lfloor k/2 \rfloor}(D^T D) = O(t_u \Delta^2 n/k)$ and $\gamma_{\lceil k/2 \rceil - 1}(Q^T Q) = O(t_q \Delta^2 m/k)$.

Our last step is to bound from above the eigenvalues of $(QD)^T QD$. Letting $\gamma_k((QD)^T QD)$ denote the k'th largest eigenvalue of $(QD)^T QD$, we get from the Courant-Fischer characterization of eigenvalues that

$$\gamma_k((QD)^T QD) = \min_{S:\dim(S) \geq n-k+1} \max_{x \in S : \|x\|_2 = 1} \|QDx\|_2^2,$$

i.e. $\gamma_k((QD)^T QD)$ equals the minimum over all subspaces S of $\mathbb{R}^n$ of dimension at least $n-k+1$, of the maximum square of the stretch of a unit length vector x when multiplying with QD. We thus aim to find a subspace S of dimension at least $n-k+1$, such that every unit vector in S is scaled as little as possible when multiplied with QD. We choose the subspace S consisting of all vectors x, for which $\langle v_i(D), x \rangle = 0$ for $i = 1, \dots, \lfloor k/2 \rfloor$ and $\langle v_i(Q), Dx \rangle = 0$ for $i = 1, \dots, \lceil k/2 \rceil - 1$. Here $v_i(D)$ denotes the i'th column vector of $V(D)$ and $v_i(Q)$ denotes the i'th column vector of $V(Q)$. Clearly $\dim(S) \geq n-k+1$. Now let $x \in S$ be a unit length vector and consider first the product $Dx = U(D)\Sigma(D)V(D)^T x$. Since x is orthogonal to the first $\lfloor k/2 \rfloor$

row vectors in $V(D)^T$ and since $\sigma_{i,i}(D) = O(\sqrt{t_u n/k}\Delta)$ for $i \geq \lfloor k/2 \rfloor$, we get that $\|\Sigma(D)V(D)^T x\|_2 = O(\sqrt{t_u n/k}\Delta)$. Since $U(D)$ is unitary, this implies $\|Dx\|_2 = O(\sqrt{t_u n/k}\Delta)$. Finally, since Dx is orthogonal to the first $\lceil k/2 \rceil - 1$ row vectors of $V(Q)^T$, we conclude $\|QDx\|_2^2 = O(t_q t_u \Delta^4 mn/k^2)$. We have thus shown that $\gamma_k((QD)^T QD) = O(t_q t_u \Delta^4 mn/k^2)$. But $A^T A = (QD)^T QD$ and hence $\gamma_k((QD)^T QD) = \lambda_k$, i.e. $t_q t_u = \Omega(\lambda_k k^2/mn\Delta^4)$. This completes the proof of Theorem 3.3.

With Theorem 3.3 established, we can now prove Theorem 3.2 which connects range reporting and group model range searching. The theorem is restated here:

Restatement of Theorem 3.2. *Let P be a set of n input objects to a range searching problem and $\mathcal{R}$ a set of $m \leq n$ query ranges over P. If $\mathcal{R}$ is strongly $(t, 2)$-favorable, then any oblivious data structure for the range searching problem must have $t_q t_u = \Omega(m^2 t/n^2 \Delta^4)$ on the input set P. Here t_q denotes the worst case query time, t_u the worst case update time and Δ the multiplicity of the data structure. For $m = \Theta(n)$ and $\Delta = O(1)$, this bound simplifies to $t_q t_u = \Omega(t)$.*

Proof of Theorem 3.2. Let P and $\mathcal{R}$ be as in Theorem 3.2, i.e. $\mathcal{R}$ is a strongly $(t, 2)$-favorable set of queries. Furthermore, let A be the $m \times n$ incidence matrix corresponding to P and $\mathcal{R}$, where $m \leq n$. Our proof is based on lower bounding the eigenvalues of $M = A^T A$, and then applying Theorem 3.3. We lower bound these eigenvalues using the following theorem of Chazelle and Lvov:

Theorem 3.6 (Chazelle and Lvov [39]). *Let A be an $m \times n$ real matrix where $m \leq n$ and let $M = A^T A$. Then M has at least*

$$\frac{n}{16\,\mathrm{tr}(M^2)n/9\,\mathrm{tr}(M)^2 - 7/9}$$

eigenvalues that are greater than or equal to $\mathrm{tr}(M)/4n$.

To use Theorem 3.6, we bound $\mathrm{tr}(M)$ and $\mathrm{tr}(M^2)$. The first is easily seen to be $\mathrm{tr}(M) = \sum_{R \in \mathcal{R}} |R| = \Omega(mt)$ and the latter is bounded by

$$\begin{aligned}
\mathrm{tr}(M^2) &= \sum_{R_1 \in \mathcal{R}} \sum_{R_2 \in \mathcal{R}} |R_1 \cap R_2|^2 \\
&= \sum_{R \in \mathcal{R}} |R|^2 + \sum_{R_1 \in \mathcal{R}} \sum_{R_2 \in \mathcal{R} | R_1 \neq R_2} |R_1 \cap R_2|^2 \\
&= O(mt^2) + \\
&\quad \sum_{p \in P} \sum_{R_1 \in \mathcal{R} | p \in R_1} \sum_{R_2 \in \mathcal{R} | p \in R_2 \wedge R_1 \neq R_2} |R_1 \cap R_2| \\
&= O(mt^2) + \sum_{p \in P} \sum_{R_1 \in \mathcal{R} | p \in R_1} \sum_{R_2 \in \mathcal{R} | p \in R_2 \wedge R_1 \neq R_2} O(1) \\
&= O(mt^2) + \sum_{p \in P} O((mt/n)^2) \\
&= O(mt^2).
\end{aligned}$$

Plugging these values into Theorem 3.6, we conclude that $M = A^T A$ has $\Omega(nm^2t^2/mt^2n) = \Omega(m)$ eigenvalues greater than $\Omega(mt/n)$. Finally invoking Theorem 3.3, we get that $t_q t_u = \Omega(mt/n \cdot m^2/mn\Delta^4) = \Omega(m^2t/n^2\Delta^4)$, which completes the proof of Theorem 3.2.

Implications. As already mentioned in Section 3.2, we obtain a number of lower bounds from Theorem 3.2 by reusing favorable query sets constructed for proving pointer machine range reporting lower bounds. Thus in the following, we only mention our proof of the lower bound for orthogonal range reporting in non-constant dimension $d = \Omega(\lg n/\lg\lg n)$.

For non-constant dimensions $d = \Omega(\lg n/\lg\lg n)$, Chazelle and Lvov [39] showed one can construct a set of n points and n query rectangles such that the corresponding incidence matrix A satisfies $\mathrm{tr}(A^T A) = n^{1+\Omega(1/\lg\lg n)}$ and $\mathrm{tr}((A^T A)^2) = O(\mathrm{tr}(A^T A)^2/n)$. From Theorem 3.6, this means that $A^T A$ has $\Omega(n)$ eigenvalues that are greater than $n^{\Omega(1/\lg\lg n)}$. The result follows immediately from Theorem 3.3.

3.5 Concluding Remarks

In this chapter we established three powerful theorems relating the update and query time of dynamic range searching data structures in the group model to combinatorial discrepancy, eigenvalues and range reporting, respectively. Our result immediately implied a whole range of data structure lower bounds, and also an improved upper bound for the discrepancy of axis-aligned rectangles in dimensions $d \geq 3$.

We believe our results represents a big leap in the right direction, but there are still a number of open problems to consider. Most importantly, we would like to remove the dependence on the multiplicity of data structures. Proving lower bounds independent of the multiplicity seems closely related to matrix rigidity and depth-2 linear circuits computing linear operators, and thus might turn out to be very challenging. A breakthrough in this direction might also help towards establishing higher lower bounds for static range searching problems. On the other hand, it would also be interesting to find an example of a range searching problem for which high multiplicity helps. If possible, this seems to involve finding a completely new approach to designing data structures, and might inspire improved solutions to many natural problems.

Extending the achieved lower bounds to weakly oblivious data structures would also be a major result, especially if this could be done independent of the multiplicity. Previous such lower bounds could even be extended to the cell probe model.

The lower bounds obtained from our technique are all for the product of the update time and query time. Finding a technique that can prove different types of tradeoffs between the two would also be extremely interesting. In particular, such a technique might be used to prove a lower bound for halfspace range searching that (essentially) matches the entire tradeoff curve of [61].

Finally, we mention that there is some hope of removing the $\lg m$ factor in the ℓ_∞-discrepancy results. More specifically, the famous Komlós conjecture states that any sequence of unit length vectors can be assigned signs such that the signed sum of the vectors have all coordinates bounded by a constant. Except for the linear transformation with Q, this is exactly the property we used to derive our ℓ_∞-discrepancy bounds. Thus it seems likely that a proof of Komlós' conjecture could be extended to our case and thus remove the $\lg m$ factor.

Chapter 4

The Pointer Machine and the I/O-Model

In this chapter, we first present a new technique for proving lower bounds for range reporting problems in the pointer machine and the I/O-model. First recall from Section 1.5 that in the pointer machine model, a data structure for a range reporting problem is a directed graph, with one node designated as the root. Each node of the graph stores either an input object or some auxiliary data, plus a constant number of pointers. When answering queries, the data structure explores a subset of the nodes in the graph by following pointers. The exploration always starts at the root node, and at each step, one edge leaving the currently explored nodes may be chosen and the node pointed to is retrieved. We require that every input object reported by a query must be stored in one of the explored nodes. The space is defined as the size of the graph and the query time is the number of nodes explored when answering a query.

The bounded degree of the nodes in the graph and the indivisibility requirement (input objects have to be stored in the nodes in "plain" form) are the two crucial properties that allow us to prove high lower bounds. The lack of random accesses effectively forces the objects reported by a query to be stored close together, since otherwise the query time would be high. From the bounded degree, we know that each node of the graph is close to only a few other nodes and hence input objects must be replicated many times if they can be reported by many queries with almost disjoint output sets. This intuition is at the heart of our new technique.

We apply our new technique to obtain tight lower bounds for the *rectangle stabbing* problem, i.e. preprocess a set of d-dimensional axis-aligned rectangles such that the rectangles containing a query point can be reported efficiently. By a simple reduction, this gives the lower bounds mentioned in Section 1.5 and Section 1.6 for orthogonal range reporting. Finally, we present another lower bound for orthogonal range reporting in the I/O-model, using the indexability framework of Hellerstein et al. [47].

4.1 Lower Bounds for Rectangle Stabbing

In the following, we introduce our new technique for proving lower bounds in the pointer machine and I/O-model and apply it to the rectangle stabbing problem. We start by introducing the technique in the pointer machine setting and later show how to extend the results to the I/O-model. We have stated the main lower bound proved in this section here:

Theorem 4.1. *For $d \geq 2$, any pointer machine data structure for rectangle stabbing in d-dimensional space, having space usage S and query time $t_q + t_k k$, must satisfy $t_q = \Omega(\lg n \cdot \lg_h^{d-2} n)$ where $h = \max\{S/n, 2^{t_k}\}$. Here k denotes the size of the output.*

This lower bound is known to be tight for $S \geq n \lg^{d-2+\varepsilon} n$ for any constant $\varepsilon > 0$. As mentioned earlier, we also obtain lower bounds for orthogonal range reporting using a reduction from rectangle stabbing. The reduction is as follows: Given a set of n input axis-aligned rectangles in d-dimensional space, we can solve the d-dimensional rectangle stabbing problem using a data structure for $2d$-dimensional orthogonal range reporting. This is done by mapping each input rectangle $[x_{1,1} : x_{1,2}] \times \cdots \times [x_{d,1} : x_{d,2}]$ to the $2d$-dimensional point $(x_{1,1}, x_{1,2}, x_{2,1}, \ldots, x_{d,2})$. To answer a d-dimensional stabbing query with the query point $(y_1, \ldots, y_d)$, we ask the $2d$-dimensional orthogonal range reporting query $(-\infty : y_1] \times [y_1 : \infty) \times \cdots \times [y_d : \infty)$. Correctness follows immediately and we conclude:

Corollary 4.1. *For $d \geq 4$, any pointer machine data structure for orthogonal range reporting in d-dimensional space, having space usage S and query time $t_q + t_k k$, must satisfy $t_q = \Omega(\lg n \cdot \lg_h^{\lfloor d/2 \rfloor - 2} n)$ where $h = \max\{S/n, 2^{t_k}\}$. Here k denotes the size of the output.*

4.1.1 Pointer Machine Lower Bound

Consider a pointer machine data structure D that answers rectangle stabbing queries in d-dimensional space. Let S be the space used by D. Furthermore, assume D answers every query in $t_q + t_k k$ time in which k is the output size. Define $h = \max\{S/n, 2^{4t_k}\}$. Our goal is to prove that $t_q = \Omega(\lg n \cdot \lg_h^{d-2} n)$.

We build an input set, $\mathcal{R}$, of rectangles that is in fact a simplification of the ones used before [5, 34, 35]. Consider a d-dimensional cube $\mathcal{C}$ with side length m, $m > \sqrt{n}$, in which m is a parameter to be determined later. Let $r = c_r h^3$, in which c_r is a large enough constant, and let $\text{Log}_r m = \lfloor \lg_r m \rfloor - 1$. For every choice of $d-1$ indices, $1 \leq i_1, \cdots, i_{d-1} \leq \text{Log}_r m$, we divide the j-th dimension of $\mathcal{C}$ into r^{i_j} equal length pieces for $1 \leq j \leq d-1$, and the d-th dimension of $\mathcal{C}$ into $\lfloor m/r^{i_1+\cdots+i_{d-1}} \rfloor$ equal length pieces. The number of such choices is $k = \text{Log}_r^{d-1} m$. Observe that for each such choice of $i_1, \ldots, i_{d-1}$, we create between $m/2$ and m rectangles and thus $\Theta(mk)$ rectangles are created in total. Furthermore, each rectangle has a volume between m^{d-1} and $2m^{d-1}$. We pick m such that this number equals n, thus $n = \Theta(mk)$. We let $\mathcal{R}$ denote the set of input rectangles. The crucial property of this input set is the following.

Observation 4.1. *The volume of the intersection of any two rectangles in $\mathcal{R}$ is at most $2m^{d-1}/r$.*

For the lower bound proof, consider a single query point $\mathbf{q}$ chosen uniformly at random inside $\mathcal{C}$. Note that the output size of $\mathbf{q}$ is k. The rest of this section is devoted to proving that with positive probability answering $\mathbf{q}$ needs $\Omega(k \lg h) = \Omega(\mathrm{Log}_r^{d-1} m \cdot \lg h) = \Omega(\lg n \cdot \lg_h^{d-2} n)$ time.

Let G be the directed graph obtained by implementing D on $\mathcal{R}$. By increasing the space and query time by a constant factor, we can ensure that each node of G has out-degree at most 2 instead of some larger constant. We thus make this assumption.

For a node $u \in G$, let $\mathrm{In}(u)$ denote the set of nodes in G that have a directed path of size at most $\lg h$ to u. Similarly, define $\mathrm{Out}(u)$ to be the set nodes in G that can be reached from u using a path of size at most $\lg h$. For a rectangle R, let V_R be the set of nodes of G that store R. We define $\mathrm{In}(R) = \sum_{v \in V_R} \mathrm{In}(v)$ and $\mathrm{Out}(R) = \sum_{v \in V_R} \mathrm{Out}(v)$. We say R is *popular* if $\mathrm{In}(R) > ch^2$.

Lemma 4.1. *There are at most n/c popular rectangles.*

Proof. As each node in G has at most two out edges, for every $u \in G$, the size of $\mathrm{Out}(u)$ is at most $2^{\lg h} = h$. Also, if $v \in \mathrm{Out}(u)$, then $u \in \mathrm{In}(v)$ and vice versa. Thus,

$$\sum_{R \in Q} |\mathrm{In}(R)| = \sum_{R \in Q} |\mathrm{Out}(R)| = \sum_{v \in G} |\mathrm{In}(v)| = \sum_{u \in G} |\mathrm{Out}(u)| \leq Sh \leq nh^2.$$

Here we used that $S \leq nh$ by our definition of h. This bound implies that the number of rectangles R with $\mathrm{In}(R) > ch^2$ is at most n/c. □

Lemma 4.2. *With probability at least $3/5$, $\mathbf{q}$ is enclosed by at most $k/4$ rectangles that are popular, if c is to be a chosen sufficiently large constant.*

Proof. By Lemma 4.1, the number of popular rectangles is at most n/c. As each rectangle has volume at most $2m^{d-1}$, the total volume of popular rectangles is at most $2m^{d-1}n/c$. Let A be the region of $\mathcal{C}$ that contains all the points covered by more than $k/4$ popular rectangles. We have,

$$\mathrm{Vol}(A)k/4 \leq 2m^{d-1}n/c = \Theta(m^d k/c)$$

which implies $\mathrm{Vol}(A) = O(m^d/c) < 2m^d/5$ if c is large enough. Thus, with probability at least $3/5$, $\mathbf{q}$ will not be picked in A. □

Let S' be the subset of rectangles that are not popular and let $n' = |S'|$. We say two rectangles $R_1, R_2 \in S'$ are *close*, if there exists a node $u \in G$ such that from u we can reach a node that stores R_1 and a node that stores R_2 with directed paths of size at most $\lg h$ each.

Lemma 4.3. *A rectangle $R \in S'$ can be close to at most ch^3 other rectangles in S'.*

Proof. If a rectangle R is close to a rectangle R', then there exists a node $u \in \text{In}(R)$ such that R' is stored in a node in $\text{Out}(u)$. For every $u \in G$ we have $\text{Out}(u) \leq h$, and $|\text{In}(R)| \leq ch^2$. Thus, R can be close to ch^3 different rectangles. □

Consider a rectangle $R \in S'$. Let $\mathcal{R}_R$ be the subset of rectangles in $\mathcal{R}$ that are close to R. We call $\cup_{R' \in \mathcal{R}_R}(R \cap R')$ the *close region* of R and denote it with C_R.

Lemma 4.4. *With probability at least 1/5, answering* $\mathbf{q}$ *needs* $k/3 \cdot \lg h$ *time.*

Proof. Consider a rectangle $R \in S'$ and the set $\mathcal{R}_R$ defined above. By Lemma 4.3, $|\mathcal{R}_R| \leq ch^3$. By Observation 4.1, for every $R' \in \mathcal{R}_R$, $\text{Vol}(R \cap R') \leq 2m^{d-1}/r$ and thus

$$\text{Vol}(C_R) \leq \sum_{R' \in \mathcal{R}_R} \text{Vol}(R \cap R') \leq 2ch^3 m^{d-1}/r = 2cm^{d-1}/c_r.$$

As there are at most n rectangles in S', the sum of the volumes of the close regions of all the rectangles in S' is at most $n \cdot 2cm^{d-1}/c_r = \Theta(ckm^d/c_r)$. Consider the region B of all the points p such that p is inside the close region of at least $k/4$ rectangles of S'. We have,

$$\text{Vol}(B)k/4 \leq \sum_{R \in S'} \text{Vol}(C_R) \leq \Theta(ckm^d/c_r)$$

which means $\text{Vol}(B) = O(cm^d/c_r)$. If we choose c_r large enough, this volume is less than $2m^d/5$, which means with probability at least 3/5, the query $\mathbf{q}$ will not be inside B. Combined with Lemma 4.2, this gives the following: with probability at least 1/5, $\mathbf{q}$ will be inside at least $3k/4$ rectangles that are in S' (by Lemma 4.2) and it will also be inside the close region of at most $k/4$ rectangles. Consider the event when this happens and let $H_\mathbf{q}$ be the subgraph of G that is explored by the query procedure while answering $\mathbf{q}$. Assume $\mathbf{q}$ needs to output rectangles $R_1, \cdots, R_{k'}$ from S'. We have $k' \geq 3k/4$. Let v_i be a node in $H_\mathbf{q}$ that stores R_i, $1 \leq i \leq k'$. Also, let W_i be the set of nodes in $H_\mathbf{q}$ that are reachable by traversing up to $\lg h$ edges from v_i, where the direction of edges have been reversed. If two sets W_i and W_j, $1 \leq i < j \leq k'$, share a common node, it means that $\mathbf{q}$ is inside the close region of both R_i and R_j. However, we know that $\mathbf{q}$ is inside the close region of at most $k/4$ rectangles. This means that there is a subset of at least $3k/4 - k/4 = k/2$ sets W_i that are pairwise disjoint. Furthermore, from each v_i, there is a path from v_i to the root node of the data structure using only reversed edges. Thus amongst the $k/2$ disjoint sets, at most one such set can have size less than $\lg h$ (since the sets are disjoint, only one contains the root and hence can reach the root in less than $\lg h$ steps). It follows that the size of $H_\mathbf{q}$ is at least $(k/2-1) \cdot \lg h \geq k/3 \cdot \lg h$. □

Since any query reports k rectangles, it must hold that $t_q + t_k k \geq k/3 \cdot \lg h$. But $t_k k \leq k/4 \cdot \lg h$ by definition of h. Hence $t_q = \Omega(k \lg h) = \Omega(\lg n \cdot \lg_h^{d-2} n)$. Defining $h' = \max\{S/n, 2^{t_k}\}$ we get that $t_q = \Omega(\lg n \cdot \lg_h^{d-2} n) = \Omega(\lg n \cdot \lg_{h'}^{d-2} n)$. This completes the proof of Theorem 4.1.

4.1.2 Extension to the I/O-Model

In the following, we extend the proof technique from the previous section to also obtain lower bounds in the I/O-model. Recall that in the I/O-model, a data structure stores input objects in blocks of B elements and we measure the query cost in the number of blocks that must be read to retrieve all elements inside a query range. As in the pointer machine model, we again have a notion of two input objects being stored close to each other, namely two input objects are close, if they are stored in the same disk block. Thus we have both of the two key ingredients from the pointer machine lower bound proof: indivisibility and closeness. It should not come as a surprise that our proof follows the pointer machine proof almost uneventfully.

Assume an I/O-model data structure exists with query time $t_q + t_k k/B$ and space S words (S/B blocks). As in Section 4.1.1, define $h = \max\{S/n, 32t_k\}$ and $r = c_r h^2$ for a sufficiently large constant c_r (note the dependence on t_k is exponentially better than in the pointer machine setting). Also define $\alpha = B/8t_k$. We assume throughout the proof that $\sqrt{n} \geq B$, which is true when making the standard *tall-cache assumption* ($n \geq M \geq B^2$). Our hard input instance is a modified version of the input instance used for the pointer machine lower bound in Section 4.1.1. Again we consider the d-dimensional cube $\mathcal{C}$ with side length $m, m > \sqrt{n/\alpha} \geq n^{1/4}$, in which m is a parameter to be determined later. We let $\mathrm{Log}_r m = \lfloor \lg_r m \rfloor - 1$. For every choice of $d-1$ indices, $1 \leq i_1, \cdots, i_{d-1} \leq \mathrm{Log}_r m$, we divide the j-th dimension of $\mathcal{C}$ into r^{i_j} equal length pieces for $1 \leq j \leq d-1$, and the d-th dimension of $\mathcal{C}$ into $\lfloor m/r^{i_1+\cdots+i_{d-1}} \rfloor$ equal length pieces. The number of such choices is $k = \mathrm{Log}_r^{d-1} m$. With each such choice, we create between $m/2$ and m rectangles and thus $\Theta(mk)$ rectangles are created in total. We pick m such that this number equals n/α, thus $n = \Theta(mk\alpha)$. Also, note that the volume of each rectangle is between m^{d-1} and $2m^{d-1}$. We let I denote the set of constructed rectangles. We call these rectangles the *meta-rectangles*. For each meta-rectangle $M \in I$, we make α copies of that meta-rectangle and add the copies to a set of rectangles $\mathcal{R}$. The set $\mathcal{R}$ constitutes the hard input instance for the I/O-model. Note that $|\mathcal{R}| = n$. The input rectangles have the following property (the equivalent of Observation 4.1):

Observation 4.2. *The volume of the intersection of any two rectangles in $\mathcal{R}$, that are not copies of the same meta-rectangle, is at most $2m^{d-1}/r$.*

Again we consider a single query point $\mathbf{q}$ chosen uniformly at random inside $\mathcal{C}$. The output size of $\mathbf{q}$ is $k\alpha$. Now consider the set of blocks $\mathbb{B}$ stored by a data structure supporting rectangle stabbing queries on $\mathcal{R}$. Each block $b \in \mathbb{B}$ stores B input rectangles from $\mathcal{R}$. We note that $|\mathbb{B}| \leq S/B$.

We define a pair of rectangles $(R_1, R_2) \in \mathcal{R} \times \mathcal{R}$ to be a *close pair* for block $b \in \mathbb{B}$, if both R_1 and R_2 are stored in b, and at the same time, R_1 and R_2 are not copies of the same meta-rectangle. We also define the *close region* of a close pair (R_1, R_2) for a block b, as the region $R_1 \cap R_2$. By Observation 4.2, the close region of a close pair has volume at most $2m^{d-1}/r$.

We now prove:

Lemma 4.5. *With probability at least $1/2$, answering $\mathbf{q}$ needs at least $k/4$ I/Os.*

Proof. For each block $b \in \mathbb{B}$, let C_b be the set of all close pairs for block b. Since each disk block contributes at most B^2 close pairs, we have $\sum_{b\in\mathbb{B}} |C_b| \leq S/B \cdot B^2 = SB$. Combined with Observation 4.2, we get that the total volume of all close regions is bounded by $2SBm^{d-1}/r$. Now consider the region A, of all the points p, such that p is inside the close region of at least $B\alpha k/h$ close pairs (possibly counting the same pair of rectangles multiple times if they are stored together in multiple blocks). We have

$$\mathrm{Vol}(A)B\alpha k/h \leq \sum_{b\in\mathbb{B}} \sum_{(R_1,R_2)\in C_b} \mathrm{Vol}(R_1 \cap R_2) \leq 2SBm^{d-1}/r.$$

which means that

$$\begin{aligned}\mathrm{Vol}(A) &= O(SBm^{d-1}h/B\alpha kr) = O(nh^2m^{d-1}/\alpha kr) \\ &= O(h^2m^d/r) = O(m^d/c_r).\end{aligned}$$

If we choose c_r large enough, the volume is less than $m^d/2$, which means that with probability $1/2$, the query $\mathbf{q}$ will not be inside A.

Consider the event when this happens and let $\mathbb{B}_\mathbf{q}$ be the subset of blocks read by the query algorithm when answering the query $\mathbf{q}$. Also let $\mathcal{R}_\mathbf{q} = \{R_{1,1}, \dots, R_{1,\alpha}, \dots, R_{k,\alpha}\}$ be the set of rectangles containing the query $\mathbf{q}$, where all $R_{i,j}$ for a fixed $i \in \{1, \dots, k\}$ are copies of the same meta-rectangle. There is at least one block in $\mathbb{B}_\mathbf{q}$ storing each $R_{i,j}$. We also have that $\mathbf{q} \in R_{i,j} \cap R_{i',j'}$ for any two such rectangles, hence if $(R_{i,j}, R_{i',j'})$ is a close pair for some block, then $\mathbf{q}$ is inside the close region of that pair.

We now bound the total number of different rectangles from $\mathcal{R}_\mathbf{q}$ that are stored in $\mathbb{B}_\mathbf{q}$. For this, observe that if a block b stores $\beta \geq 2\alpha$ different rectangles from $\mathcal{R}_\mathbf{q}$, then there are at least $\beta(\beta-\alpha)/2 \geq \beta^2/4$ close pairs for block b, where $\mathbf{q}$ is inside the close region of all those pairs. Now let $b_1, \dots, b_{|\mathbb{B}_\mathbf{q}|}$ be the blocks in $\mathbb{B}_\mathbf{q}$ and let β_i denote the number of different rectangles from $\mathcal{R}_\mathbf{q}$ that are stored in b_i. Since $\mathbf{q}$ is not in A, we have

$$\sum_{i:\beta_i\geq 2\alpha} \beta_i^2/4 \leq Bk\alpha/h.$$

This implies

$$\sum_{i:\beta_i\geq 2\alpha} \beta_i \leq 2\sqrt{|\mathbb{B}_\mathbf{q}|Bk\alpha/h}.$$

We can finally bound the total number of different rectangles from $\mathcal{R}_\mathbf{q}$ that are stored in $\mathbb{B}_\mathbf{q}$ as $|\mathbb{B}_\mathbf{q}|2\alpha + 2\sqrt{|\mathbb{B}_\mathbf{q}|B\alpha/h}$. But all rectangles from $\mathcal{R}_\mathbf{q}$ must be reported and $|\mathcal{R}_\mathbf{q}| = k\alpha$, hence we must have

$$\begin{aligned}|\mathbb{B}_\mathbf{q}|2\alpha + 2\sqrt{|\mathbb{B}_\mathbf{q}|Bk\alpha/h} &\geq k\alpha \Rightarrow \\ |\mathbb{B}_\mathbf{q}| + \sqrt{|\mathbb{B}_\mathbf{q}|Bk/h\alpha} &\geq k/2.\end{aligned}$$

For this to be satisfied, we must have either $|\mathbb{B}_\mathbf{q}| \geq k/4$ or $\sqrt{|\mathbb{B}_\mathbf{q}|Bk/h\alpha} \geq k/4$, which implies

$$|\mathbb{B}_\mathbf{q}| \geq \min\{k/4, kh\alpha/16B\} \geq \min\{k/4, 2kt_k\alpha/B\} \geq k/4,$$

which completes the proof. □

We are finally ready to derive the lower bound. Since the query time is $t_q + t_k k\alpha/B = t_q + k/8$, we have $|\mathbb{B}_{\mathbf{q}}| \leq t_q + k/8$. Combining this with Lemma 4.5, we get $t_q \geq k/8 = \Omega(\lg_r^{d-1} m) = \Omega(\lg_h^{d-1}(n/B))$. Since $n \geq B^2$ under the tall-cache assumption, the lower bound simplifies to $t_q = \Omega(\lg_h^{d-1} n)$.

4.2 Indexability Lower Bound

In this section we use the indexability theory of Hellerstein et al. [47] to prove a space lower bound for orthogonal range reporting in the I/O-model. Our lower bound states that any data structure answering queries in $t_q + t_k k/B$ I/Os must use $\Omega(n(\lg n/\lg(t_q t_k))^{d-1})$ space.

Recall from Section 1.6.1 that, in the indexability model [47], an indexing problem is described by a *workload* $W = (I, Q)$, where I is a set of input elements and Q is a set of subsets of I; the elements of Q are called *queries*. Given a workload W and a block size B, an *indexing scheme* S is defined on I by a block assignment function, $\mathbb{B}$, which is a set of B-sized subsets of I. We think of all the elements in a set $b \in \mathbb{B}$ as being stored in one block.

Restatement of Theorem 1.2 (Refined Redundancy Theorem [15]). *For a workload $W = (I, Q)$ where $Q = \{q_1, q_2, \ldots, q_m\}$, let $(I, \mathbb{B})$ be an indexing scheme for W with query cost (t_q, t_k) with $t_k \leq \sqrt{B}/8$ such that for any $1 \leq i, j \leq m, i \neq j : |q_i| \geq Bt_q$ and $|q_i \cap q_j| \leq B/(64t_k^2)$. Then the space of $(I, \mathbb{B})$ is at least $\frac{1}{12}\sum_{i=1}^{m} |q_i|$.*

Consider a data structure for d-dimensional orthogonal range reporting with $t_q + t_k k/B$ query cost. The refined redundancy theorem states that if we can construct a set of n points and m query rectangles $q_1, \ldots, q_m$, such that any rectangle contains at least Bt_q points and where the intersection of any pair of rectangles contains at most $B/(64t_k^2)$ points, then the amount of space needed by any data structure for that input is $\Omega(\sum_{i=1}^{m} |q_i|)$. The goal is thus to maximize the sum of the sizes of the queries.

In two-dimensions, the $\Omega(n \lg n/\lg(t_q t_k))$ space lower bound of Hellerstein et al. [47] was obtained using a *Fibonacci workload*. However, generalizing the Fibonacci workload to higher dimensions seems hard, and the previously best known d-dimensional $\Omega(n(\lg B/\lg(t_q t_k))^{d-1})$ space lower bound instead utilizes a simple point set consisting of an $n^{1/d} \times \cdots \times n^{1/d}$ grid. In the pointer machine, the $\Omega(n(\lg n/\lg(t_q))^{d-1})$ space lower bound for d-dimensional orthogonal range reporting was proved by Chazelle [34]. In 2-d, a fairly simple point set (workload) was used to prove the bounds, whereas a much more complex point set and a randomized argument was used in higher dimensions.

Here we generalize Chazelle's planar point set to higher dimensions using a deterministic construction. Such a deterministic generalization was given by Chazelle as well, but for offline orthogonal range searching in the semi-group model [35]. In fact, by modifying the parameters used in his proof, one can prove the $\Omega(n(\lg n/\lg(t_q t_k))^{d-1})$ space lower bound we are after; however, the lower bound will be valid only if $B = (t_q t_k)^{O(1)}$, which is not an interesting setting when e.g. $t_q = \lg_B n$ and $t_k = O(1)$. Relaxing this constraint seems to require more substantial changes, e.g., changing the query or the point set. Here we present an alternate but similar construction that achieves this. Our lower bound holds for $2 \leq B \leq \sqrt{M} \leq \sqrt{n}$, i.e. the *tall-cache assumption*, which is a much more reasonable assumption. Throughout the proof, we assume $t_q \leq n^{1/4}$. This is not an issue, since otherwise we only claim a linear space lower bound.

Point set I. Let $a_1 = 64 t_q t_k^2$ and $a_j = (\prod_{i=1}^{j-1} a_i) + 1$ for $j = 2, \dots, d-1$. It is easily verified that $a_1, a_2, \dots, a_{d-1}$ are relatively prime (for any a_i and a_j where $i \neq j$, the greatest common divisor satisfies $\gcd(a_i, a_j) = 1$). We define the point set $I := \{(p_{a_1}(i), p_{a_2}(i), \dots, p_{a_{d-1}}(i), i) \mid i = 0, 1, \dots, N-1\}$, where $p_{a_j}(i)$ is obtained by first writing i in base a_j, then removing all but the $\lfloor \lg_{a_j} n^{\frac{1}{4d}} \rfloor$ least significant digits, and finally reversing all the digits of the constructed number (adding leading 0-digits first if the number has less than $\lfloor \lg_{a_j} n^{\frac{1}{4d}} \rfloor$ digits). We will use $\overleftarrow{m_{k-1} \dots m_0}$ to denote the reversal of $m_{k-1} \dots m_0$, that is, $\overleftarrow{m_{k-1} \dots m_0} = m_0 \dots m_{k-1}$. The following lemma is an easy consequence of the definitions given above.

Lemma 4.6. *Consider the i'th point $p_i = (p_{a_1}(i), \dots, p_{a_{d-1}}(i), i)$ in I. The k most significant digits of the j'th coordinate $p_{a_j}(i)$ are precisely $\overleftarrow{i \mod a_j^k}$ for $k \leq \lfloor \lg_{a_j} n^{\frac{1}{4d}} \rfloor$.*

Let $\mathcal{C}$ be the rectangle in the positive quadrant anchored at the origin $(0, 0, \dots, 0)$ with dimensions $a_1^{\lfloor \lg_{a_1} n^{1/4d} \rfloor} \times \cdots \times a_{d-1}^{\lfloor \lg_{a_{d-1}} n^{1/4d} \rfloor} \times n$; $\mathcal{C}$ contains all points in I. Now consider a rectangle q inside $\mathcal{C}$, and let $[x_1 : x_2]$ be the range it spans in the j'th dimension. If $x_1 = m_0 \dots m_{k-1} 00 \dots 0$ and $x_2 = m_0 \dots m_{k-1} (a_j - 1)(a_j - 1) \dots (a_j - 1)$ in base a_j for some $m_0 \dots m_{k-1}$, it follows from Lemma 4.6 that each point p_i with $(i \mod a_j^k) = \overleftarrow{m_0 \dots m_{k-1}}$ has the j'th coordinate in the range $[x_1 : x_2]$. If this holds for each of the first $d-1$ dimensions, we can determine whether a point is inside q simply by looking at its d'th coordinate.

Query set Q. Consider the set R consisting of one rectangle with each of the following dimensions $a_1^{i_1} \times a_2^{i_2} \times \cdots \times a_{d-1}^{i_{d-1}} \times B t_q a_1^{k_1} a_2^{k_2} \dots a_{d-1}^{k_{d-1}}$ for $i_j \in \{0, \dots, \lfloor \lg_{a_j} n^{\frac{1}{4d}} \rfloor\}$ and $k_j = \lfloor \lg_{a_j} n^{\frac{1}{4d}} \rfloor - i_j$.

Lemma 4.7. *Any $r \in R$ placed at the origin in d-dimensional space is completely contained in $\mathcal{C}$. Furthermore, $|R| = \Omega((\lg n / \lg t_q t_k)^{d-1})$*

Proof. The first $d-1$ dimensions of r are obviously within $\mathcal{C}$. Using the tall-cache assumption we get that $B t_q \leq \sqrt{n} t_q$ which by our assumption $t_q \leq n^{1/4}$ is bounded by $n^{3/4}$. Thus the size of the d'th dimension of r is bounded by $B t_q \cdot \prod_{i=1}^{d-1} a_i^{\lfloor \lg_{a_i} n^{1/4d} \rfloor} \leq n^{\frac{3}{4}} \cdot (n^{\frac{1}{4d}})^{d-1} \leq n$, and therefore r fits within $\mathcal{C}$.

To see the bound on the size of R, simply count the number of combinations of i_j in the definition of R

$$\begin{aligned}
|R| &= \prod_{i=1}^{d-1} \lfloor \lg_{a_i} n^{\frac{1}{4d}} \rfloor + 1 \geq \prod_{i=1}^{d-1} \lg_{a_i} n^{\frac{1}{4d}} \\
&\geq \left(\lg_{a_{d-1}} n^{\frac{1}{4d}}\right)^{d-1} \geq \left(\lg_{a_1^{2^d}} n^{\frac{1}{4d}}\right)^{d-1} \\
&= \left(\lg_{a_1} n^{\frac{1}{4d\cdot 2^d}}\right)^{d-1} = \left(\frac{\lg_{a_1} n}{4d\cdot 2^d}\right)^{d-1} \\
&= \frac{\left(\lg_{a_1} n\right)^{d-1}}{(4d)^{d-1}\cdot 2^{d^2-d}} = \Omega\left(\left(\lg_{a_1} n\right)^{d-1}\right) \\
&= \Omega\left(\left(\lg_{t_q t_k^2} n\right)^{d-1}\right) = \Omega\left(\left(\frac{\lg n}{\lg t_q t_k}\right)^{d-1}\right).
\end{aligned}$$

□

Our query set Q consists of the rectangles obtained by tiling $\mathcal{C}$ with each of the rectangles $r \in R$ in turn, starting at the origin. Notice that we will use only those queries that are completely contained in $\mathcal{C}$.

Lemma 4.8. *For any query $q \in Q$, $|q| = Bt_q$.*

Proof. Let q be a rectangle in Q with dimensions $a_1^{i_1}\times\cdots\times a_{d-1}^{i_{d-1}}\times Bt_q a_1^{k_1}\dots a_{d-1}^{k_{d-1}}$, and consider its j'th dimension ($j < d$). Since q was placed by tiling from the origin, q will span the range $I_j = [c_j a_j^{i_j} : (c_j+1)a_j^{i_j})$ in the j'th dimension for some $c_j = m_0 m_1 \dots m_{k_j-1}$, where c_j is written in base a_j. From Lemma 4.6 it then follows that the i'th point $p_i = (p_{a_1}(i), \dots, p_{d-1}(i), i)$ of I is inside q if and only if

$$\forall\, 1 \leq j \leq d-1, \quad (i \mod a_j^{k_j}) = \overleftarrow{c_j} \tag{4.1}$$

and

$$c_d B t_q a_1^{k_1}\dots a_{d-1}^{k_{d-1}} \leq i < (c_d+1) B t_q a_1^{k_1}\dots a_{d-1}^{k_{d-1}}.$$

As $a_1, \dots, a_{d-1}$ are relatively prime, by the Chinese Remainder Theorem (see below), there is a unique value of i modulo $a_1^{k_1} a_2^{k_2}\dots a_{d-1}^{k_{d-1}}$ that satisfies all of the $d-1$ requirements in (1). Since $q \subset \mathcal{C}$, it follows from the last requirement on i that q contains precisely $(Bt_q a_1^{k_1}\dots a_{d-1}^{k_{d-1}})/(a_1^{k_1}\dots a_{d-1}^{k_{d-1}}) = Bt_q$ points. □

Theorem 4.2 (Chinese Remainder Theorem)**.** *Let $n = a_1 a_2 \dots a_k$ such that $a_1, a_2, \dots, a_k \in \mathbb{Z}\setminus\{0\}$ and $\gcd(a_i, a_j) = 1$ for $i \neq j$. Then for the system of congruences*

$$\begin{aligned}
X &\equiv n_1 \pmod{a_1} \\
X &\equiv n_2 \pmod{a_2} \\
&\;\vdots \\
X &\equiv n_k \pmod{a_k}
\end{aligned}$$

there is precisely one remainder class X mod n that satisfies the system.

Having defined our workload $W = (I, Q)$, we now bound the number of points in the intersection of any two query rectangles in Q.

Lemma 4.9. *For any two query rectangles $q_1, q_2 \in Q$, $|q_1 \cap q_2| \leq B/(64t_k^2)$*[1]*.*

Proof. If q_1 and q_2 have the same dimensions, we get from the tiling that $q_1 \cap q_2 = \emptyset$ and the lemma follows. Now consider the case where q_1 and q_2 differ in at least one dimension. Let $a_1^{i_1} \times \cdots \times a_{d-1}^{i_{d-1}} \times Bt_q a_1^{k(i,1)} \dots a_{d-1}^{k(i,d-1)}$ be the dimensions of q_1 and $a_1^{j_1} \times \cdots \times a_{d-1}^{j_{d-1}} \times Bt_q a_1^{k(j,1)} \dots a_{d-1}^{k(j,d-1)}$ be the dimensions of q_2. Let $l < d$ be any dimension where $i_l \neq j_l$. W.l.o.g we assume that $i_l > j_l$. Since $a_l^{i_l}$ is just a multiplicative of $a_l^{j_l}$, it follow from the tiling that the intersection of q_1 and q_2 is either empty in the l'th dimension, or spans the same range as q_2. If the range is empty, our proof is done, so assume it equals the range of q_2. Now consider the rectangle J that spans exactly the same ranges as q_1, except in the l'th dimension, where it spans the same range as q_2. Clearly $q_1 \cap q_2 \subset J$. Using the Chinese Remainder Theorem, we get that J contains at most

$$\frac{Bt_q a_1^{k(i,1)} \dots a_{d-1}^{k(i,d-1)}}{a_1^{k(i,1)} \dots a_l^{k(j,l)} \dots a_{d-1}^{k(i,d-1)}} \leq \frac{Bt_q}{a_l} \leq \frac{B}{64t_k^2}$$

points. Since $q_1 \cap q_2 \subset J$, we have that $|q_1 \cap q_2| \leq |J| \leq B/(64t_k^2)$ and the lemma follows. $\square$

We now apply the redundancy theorem. By Lemma 4.8 and Lemma 4.9, our workload $W = (I, Q)$ fulfills the requirements of the Refined Redundancy Theorem. Thus the space of any solution for this workload is at least $\frac{1}{12}\sum|q_i|$.

Now consider any rectangle $r \in R$. By Lemma 4.7 we know that r will be contained in $\mathcal{C}$ if placed at the origin. We also get from the definition of R, that the $d-1$ first dimensions of $\mathcal{C}$ are multiplicative of the $d-1$ first dimensions of r. It then follows from the tiling that every point in I will have its $d-1$ first coordinates inside one query rectangle for every $r \in R$, and at least half the points will have their d'th coordinate inside one query rectangle for every $r \in R$. Therefore $\sum |q_i| \geq |R|\frac{n}{2}$. Using Lemma 4.7 we thus conclude that the space must be at least $\Omega(n(\lg n/\lg t_q t_k)^{d-1})$.

Theorem 4.3. *There exist a workload (i.e., a set of points and a set of queries) W for d-dimensional orthogonal range reporting such that any data structure for W that can answer queries in $t_q + t_k k/B$ I/Os needs $\Omega(n(\lg n/\lg(t_q t_k))^{d-1})$ space.*

4.3 Concluding Remarks

In this chapter, we first presented a new technique for proving lower bounds in the pointer machine and I/O-model. The idea of using geometry and volume

[1]This property is one of the main difference between our point set and the one developed by Chazelle; his construction ensures that $|q_1 \cap q_2| = O(1)$ which is a more strict condition than ours that results in a weaker lower bound.

based arguments to prove lower bounds is quite novel, and as mentioned, it has already led to improved simplex range reporting lower bounds [1]. Finding further applications of our technique is still open.

Another important implication of our lower bounds is that the optimal query time for orthogonal range reporting in the pointer machine has to increase from $O(\lg n + k)$ to $\Omega((\lg n/\lg\lg n)^2 + k)$ somewhere between four and six dimensions (in 4-d, the best obtained query time was $\lg n\sqrt{\lg n/\lg\lg n}$ with $n\lg^{O(1)} n$ space). One intriguing open problem is of course to pinpoint the dimension where the optimal query bound jumps. For the I/O-model, we find it an extremely interesting consequence of our lower bounds that it is not possible to obtain a query time of $\lg_B^{O(1)} n + O(k/B)$ in dimensions $d \geq 4$.

All known upper bounds in the pointer machine and I/O-model have a query time that increases by almost a $\lg n$ factor every dimension while the increase in the lower bound happens only every other dimension. Another interesting question is therefore which of the two is closer to the behavior of the optimal query bound. In this context, it is worth noting that many other bounds in computational geometry (such as the worst case convex hull complexity, and halfspace range reporting query bounds) increase every other dimension.

Finally, we used the indexability framework of Hellerstein et al. [47] to tighten the I/O-model lower bounds for orthogonal range reporting.

Bibliography

[1] P. Afshani. Improved pointer machine and I/O lower bounds for simplex range reporting and related problems. In *Proc. 28th ACM Symposium on Computational Geometry*, pages 339–346, 2012.

[2] P. Afshani, P. K. Agarwal, L. Arge, K. G. Larsen, and J. M. Phillips. (approximate) uncertain skylines. In *Proc. 14th International Conference on Database Theory*, pages 186–196, 2011.

[3] P. Afshani, M. Agrawal, B. Doerr, K. Mehlhorn, K. G. Larsen, and C. Winzen. The query complexity of finding a hidden permutation. Manuscript.

[4] P. Afshani, L. Arge, and K. D. Larsen. Orthogonal range reporting in three and higher dimensions. In *Proc. 50th IEEE Symposium on Foundations of Computer Science*, pages 149–158, 2009.

[5] P. Afshani, L. Arge, and K. D. Larsen. Orthogonal range reporting: Query lower bounds, optimal structures in 3d, and higher dimensional improvements. In *Proc. 26th ACM Symposium on Computational Geometry*, pages 240–246, 2010.

[6] P. Afshani, L. Arge, and K. G. Larsen. Higher-dimensional orthogonal range reporting and rectangle stabbing in the pointer machine model. In *Proc. 28th ACM Symposium on Computational Geometry*, pages 323–332, 2012.

[7] P. Afshani and T. M. Chan. Optimal halfspace range reporting in three dimensions. In *Proc. 20th ACM/SIAM Symposium on Discrete Algorithms*, pages 180–186, 2009.

[8] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31:1116–1127, 1988.

[9] M. Ajtai. A lower bound for finding predecessors in yao's cell probe model. *Combinatorica*, 8(3):235–247, 1988.

[10] R. Alexander. Geometric methods in the study of irregularities of distribution. *Combinatorica*, 10(2):115–136, 1990.

[11] S. Alstrup, T. Husfeldt, and T. Rauhe. Marked ancestor problems. In *Proc. 39th IEEE Symposium on Foundations of Computer Science*, pages 534–544, 1998.

[12] A. Andoni, P. Indyk, and M. Pătraşcu. On the optimality of the dimensionality reduction method. In *Proc. 47th IEEE Symposium on Foundations of Computer Science*, pages 449–458, 2006.

[13] L. Arge and K. G. Larsen. I/O-efficient spatial data structures for range queries. *SIGSPATIAL Special*, 4(2):2–7, July 2012.

[14] L. Arge, K. G. Larsen, T. Mølhave, and F. van Walderveen. Cleaning massive sonar point clouds. In *Proc. 18th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems*, pages 152–161, 2010.

[15] L. Arge, V. Samoladas, and K. Yi. Optimal external-memory planar point enclosure. In *Proc. 12th European Symposium on Algorithms*, pages 40–52, 2004.

[16] W. Banaszczyk. Balancing vectors and gaussian measures of n-dimensional convex bodies. *Random Structures & Algorithms*, 12:351–360, July 1998.

[17] N. Bansal. Constructive algorithms for discrepancy minimization. In *Proc. 51st IEEE Symposium on Foundations of Computer Science*, pages 3–10, 2010.

[18] O. Barkol and Y. Rabani. Tighter bounds for nearest neighbor search and related problems in the cell probe model. In *Proc. 32nd ACM Symposium on Theory of Computation*, pages 388–396, 2000.

[19] P. Beame and F. E. Fich. Optimal bounds for the predecessor problem and related problems. *Journal of Computer and System Sciences*, 65:38–72, August 2002.

[20] J. Beck. Balanced two-colorings of finite sets in the square I. *Combinatorica*, 1(4):327–335, 1981.

[21] J. Beck. On irregularities of point sets in the unit square. In *Combinatorics. Proc. 7th Hungarian colloquium*, pages 63–74, 1988.

[22] J. Beck and T. Fiala. Integer-making theorems. *Discrete Applied Mathematics*, 3:1–8, February 1981.

[23] D. Bednarchak and M. Helm. A note on the Beck-Fiala theorem. *Combinatorica*, 17(1):147–149, 1997.

[24] J. L. Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, 1980.

[25] D. Bilyk, M. T. Lacey, and A. Vagharshakyan. On the small ball inequality in all dimensions. *Journal of Functional Analysis*, 254:2470–2502, May 2008.

[26] K. Bringmann and K. G. Larsen. Succinct sampling from discrete distributions. In *Proc. 45th ACM Symposium on Theory of Computation*, 2013. To appear.

[27] G. S. Brodal and K. G. Larsen. Optimal planar orthogonal skyline counting queries. Manuscript.

[28] J. Brody and K. G. Larsen. Adapt or die: Polynomial lower bounds for non-adaptive dynamic data structures. Manuscript.

[29] M. E. Caspersen, K. D. Larsen, and J. Bennedsen. Mental models and programming aptitude. In *Proc. 12th SIGCSE Conference on Innovation and Technology in Computer Science Education*, pages 206–210, 2007.

[30] T. M. Chan. Optimal partition trees. In *Proc. 26th ACM Symposium on Computational Geometry*, pages 1–10, 2010.

[31] T. M. Chan, S. Durocher, K. G. Larsen, J. Morrison, and B. T. Wilkinson. Linear-space data structures for range mode query in arrays. In *Proc. 29th Symposium on Theoretical Aspects of Computer Science*, pages 290–301, 2012.

[32] T. M. Chan, K. G. Larsen, and M. Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proc. 27th ACM Symposium on Computational Geometry*, pages 1–10, 2011.

[33] A. Chattopadhyay, J. Edmonds, F. Ellen, and T. Pitassi. A little advice can be very helpful. In *Proc. 23rd ACM/SIAM Symposium on Discrete Algorithms*, pages 615–625, 2012.

[34] B. Chazelle. Lower bounds for orthogonal range searching: I. the reporting case. *Journal of the ACM*, 37(2):200–212, 1990.

[35] B. Chazelle. Lower bounds for off-line range searching. In *Proc. 27th ACM Symposium on Theory of Computation*, pages 733–740, 1995.

[36] B. Chazelle. A spectral approach to lower bounds with applications to geometric searching. *SIAM Journal on Computing*, 27:545–556, 1998.

[37] B. Chazelle. *The Discrepancy Method: Randomness and Complexity.* Cambridge University Press, 2000.

[38] B. Chazelle and D. Liu. Lower bounds for intersection searching and fractional cascading in higher dimension. In *Proc. 33rd ACM Symposium on Theory of Computation*, pages 322–329, 2001.

[39] B. Chazelle and A. Lvov. A trace bound for the hereditary discrepancy. In *Proc. 16th ACM Symposium on Computational Geometry*, pages 64–69, 2000.

[40] B. Chazelle and B. Rosenberg. Simplex range reporting on a pointer machine. *Computational Geometry: Theory and Applications*, 5:237–247, January 1996.

[41] A. Fiat and A. Shamir. How to find a battleship. *Networks*, 19:361–371, 1989.

[42] M. Fredman and M. Saks. The cell probe complexity of dynamic data structures. In *Proc. 21st ACM Symposium on Theory of Computation*, pages 345–354, 1989.

[43] M. L. Fredman. The complexity of maintaining an array and computing its partial sums. *Journal of the ACM*, 29:250–260, January 1982.

[44] A. Gál and P. B. Miltersen. The cell probe complexity of succinct data structures. *Theoretical Computer Science*, 379:405–417, July 2007.

[45] A. Golynski. Cell probe lower bounds for succinct data structures. In *Proc. 20th ACM/SIAM Symposium on Discrete Algorithms*, pages 625–634, 2009.

[46] M. Greve, A. G. Jørgensen, K. D. Larsen, and J. Truelsen. Cell probe lower bounds and approximations for range mode. In *Proc. 37th International Colloquium on Automata, Languages, and Programming*, pages 605–616, 2010.

[47] J. M. Hellerstein, E. Koutsoupias, D. P. Miranker, C. H. Papadimitriou, and V. Samoladas. On a model of indexability and its bounds for range queries. *Journal of the ACM*, 49(1):35–55, 2002.

[48] J. Iacono and M. Pǎtraşcu. Using hashing to solve the dictionary problem (in external memory). In *Proc. 23rd ACM/SIAM Symposium on Discrete Algorithms*, pages 570–582, 2012.

[49] A. G. Jørgensen and K. G. Larsen. Range selection and median: Tight cell probe lower bounds and adaptive data structures. In *Proc. 22nd ACM/SIAM Symposium on Discrete Algorithms*, pages 805–813, 2011.

[50] K. S. Kedlaya and C. Umans. Fast modular composition in any characteristic. In *Proc. 49th IEEE Symposium on Foundations of Computer Science*, pages 146–155, 2008.

[51] E. Kushilevitz and N. Nisan. *Communication complexity.* Cambridge University Press, 1997.

[52] K. G. Larsen. On range searching in the group model and combinatorial discrepancy. In *Proc. 52nd IEEE Symposium on Foundations of Computer Science*, pages 542–549, 2011.

[53] K. G. Larsen. The cell probe complexity of dynamic range counting. In *Proc. 44th ACM Symposium on Theory of Computation*, pages 85–94, 2012.

[54] K. G. Larsen. Higher cell probe lower bounds for evaluating polynomials. In *Proc. 53rd IEEE Symposium on Foundations of Computer Science*, pages 293–301, 2012.

[55] K. G. Larsen and H. L. Nguyen. Improved range searching lower bounds. In *Proc. 28th ACM Symposium on Computational Geometry*, pages 171–178, 2012.

[56] K. G. Larsen and R. Pagh. I/O-efficient data structures for colored range and prefix reporting. In *Proc. 23rd ACM/SIAM Symposium on Discrete Algorithms*, pages 583–592, 2012.

[57] K. G. Larsen and F. van Walderveen. Near-optimal range reporting structures for categorical data. In *Proc. 24th ACM/SIAM Symposium on Discrete Algorithms*, 2013. To appear.

[58] D. Liu. A strong lower bound for approximate nearest neighbor searching. *Information Processing Letters*, 92:23–29, October 2004.

[59] J. Matoušek. Tight upper bounds for the discrepancy of half-spaces. *Discrete and Computational Geometry*, 13:593–601, 1995.

[60] J. Matoušek. *Geometric Discrepancy.* Springer, 1999.

[61] J. Matoušek. Efficient partition trees. *Discrete and Computational Geometry*, 8:315–334, 1992.

[62] J. Matoušek. Range searching with efficient hierarchical cuttings. In *Proc. 8th ACM Symposium on Computational Geometry*, pages 276–285, 1992.

[63] P. B. Miltersen. The bit probe complexity measure revisited. In *Proc. 10th Symposium on Theoretical Aspects of Computer Science*, pages 662–671, 1993.

[64] P. B. Miltersen. Lower bounds for union-split-find related problems on random access machines. In *Proc. 26th ACM Symposium on Theory of Computation*, pages 625–634, 1994.

[65] P. B. Miltersen. On the cell probe complexity of polynomial evaluation. *Theoretical Computer Science*, 143:167–174, May 1995.

[66] P. B. Miltersen, N. Nisan, S. Safra, and A. Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences*, 57(1):37–49, 1998.

[67] J. Pach and P. K. Agarwal. *Combinatorial geometry.* Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1995.

[68] R. Panigrahy, K. Talwar, and U. Wieder. Lower bounds on near neighbor search via metric expansion. In *Proc. 51st IEEE Symposium on Foundations of Computer Science*, pages 805–814, 2010.

[69] M. Pǎtraşcu. Lower bounds for 2-dimensional range counting. In *Proc. 39th ACM Symposium on Theory of Computation*, pages 40–46, 2007.

[70] M. Pătraşcu. Unifying the landscape of cell-probe lower bounds. In *Proc. 49th IEEE Symposium on Foundations of Computer Science*, pages 434–443, 2008.

[71] M. Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *Proc. 42nd ACM Symposium on Theory of Computation*, pages 603–610, 2010.

[72] M. Pătraşcu and E. D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35:932–963, April 2006.

[73] M. Pătraşcu and M. Thorup. Time-space trade-offs for predecessor search. In *Proc. 38th ACM Symposium on Theory of Computation*, pages 232–240, 2006.

[74] M. Pătraşcu and M. Thorup. Higher lower bounds for near-neighbor and further rich problems. *SIAM Journal on Computing*, 39(2):730–741, 2010.

[75] M. Pătraşcu and M. Thorup. Don't rush into a union: Take time to find your roots. In *Proc. 43rd ACM Symposium on Theory of Computation*, 2011. To appear. See also arXiv:1102.1783.

[76] K. F. Roth. On irregularities of distribution. *Mathematika*, 7:73–79, 1954.

[77] K. F. Roth. Remark concerning integer sequences. *Acta Arithmetica*, 9:257–260, 1964.

[78] P. Sen and S. Venkatesh. Lower bounds for predecessor searching in the cell probe model. *Journal of Computer and System Sciences*, 74:364–385, May 2008.

[79] C. Sommer, E. Verbin, and W. Yu. Distance oracles for sparse graphs. In *Proc. 50th IEEE Symposium on Foundations of Computer Science*, pages 703–712, 2009.

[80] A. Srinivasan. Improving the discrepancy bound for sparse matrices: better approximations for sparse lattice approximation problems. In *Proc. 8th ACM/SIAM Symposium on Discrete Algorithms*, pages 692–701, 1997.

[81] R. E. Tarjan. A class of algorithms that require nonlinear time to maintain disjoint sets. *Journal of Computer and System Sciences*, 18:110–127, 1979.

[82] A. C. C. Yao. Should tables be sorted? *Journal of the ACM*, 28(3):615–628, 1981.

[83] Y. Yin. Cell-probe proofs. *ACM Transactions on Computation Theory*, 2:1:1–1:17, November 2010.