Lower Bound Techniques for Data Structures

by

Mihai Pătraşcu

Submitted to the Department of
Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2008

Author . . . .
Department of
Electrical Engineering and Computer Science
August 8, 2008

Certified by . . . .
Erik D. Demaine
Associate Professor
Thesis Supervisor

Accepted by . . . .
Professor Terry P. Orlando
Chair, Department Committee on Graduate Students

Lower Bound Techniques for Data Structures

by

Mihai Pătraşcu

Submitted to the Department of
Electrical Engineering and Computer Science
on August 8, 2008, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

We describe new techniques for proving lower bounds on data-structure problems, with the following broad consequences:

- the first $\Omega(\lg n)$ lower bound for *any* dynamic problem, improving on a bound that had been standing since 1989;
- for static data structures, the first separation between linear and polynomial space. Specifically, for some problems that have constant query time when polynomial space is allowed, we can show $\Omega(\lg n / \lg\lg n)$ bounds when the space is $O(n \cdot \text{polylog}\, n)$.

Using these techniques, we analyze a variety of central data-structure problems, and obtain improved lower bounds for the following:

- the partial-sums problem (a fundamental application of augmented binary search trees);
- the predecessor problem (which is equivalent to IP lookup in Internet routers);
- dynamic trees and dynamic connectivity;
- orthogonal range stabbing.
- orthogonal range counting, and orthogonal range reporting;
- the partial match problem (searching with wild-cards);
- $(1+\epsilon)$-approximate near neighbor on the hypercube;
- approximate nearest neighbor in the ℓ_∞ metric.

Our new techniques lead to surprisingly non-technical proofs. For several problems, we obtain simpler proofs for bounds that were already known.

Thesis Supervisor: Erik D. Demaine
Title: Associate Professor

Acknowledgments

This thesis is based primarily on ten conference publications: [84] from SODA'04, [83] from STOC'04, [87] from ICALP'05, [89] from STOC'06, [88] and [16] from FOCS'06, [90] from SODA'07, [81] from STOC'07, [13] and [82] from FOCS'08. It seems appropriate to sketch the story of each one of these publications.

It all began with two papers I wrote in my first two undergraduate years at MIT, which appeared in SODA'04 and STOC'04, and were later merged in a single journal version [86]. I owe a lot of thanks to Peter Bro Miltersen, whose survey of cell probe complexity [72] was my crash course into the field. Without this survey, which did an excellent job of taking clueless readers to the important problems, a confused freshman would never have heard about the field, nor the problems. And, quite likely, STOC would never have seen a paper proving information-theoretic lower bounds from an author who clearly did not know what "entropy" meant.

Many thanks also go to Erik Demaine, who was my research advisor from that freshman year until the end of this PhD thesis. Though it took years before we did any work together, his unconditional support has indirectly made all my work possible. Erik's willingness to pay a freshman to do whatever he wanted was clearly not a mainstream idea, though in retrospect, it was an inspired one. Throughout the years, Erik's understanding and tolerance for my unorthodox style, including in the creation of this thesis, have provided the best possible research environment for me.

My next step would have been far too great for me to take alone, but I was lucky enough to meet Mikkel Thorup, now a good friend and colleague. In early 2004 (if my memory serves me well, it was in January, during a meeting at SODA), we began thinking about the predecessor problem. It took us quite some time to understand that what had been labeled "optimal bounds" were not optimal, that proving an optimal bound would require a revolution in static lower bounds (the first bound to beat communication complexity), and to finally find an idea to break this barrier. This 2-year research journey consumed us, and I would certainly have given up along the way, had it not been for Mikkel's contagious and constant optimism, constant influx of ideas, and the many beers that we had together.

This research journey remained one of the most memorable in my career, though, unfortunately, the ending was underwhelming. Our STOC 2006 [89] paper went largely unnoticed, with maybe 10 people attending the talk, and no special issue invitation. I consoled myself with Mikkel's explanation that ours had been paper with too many new ideas to be digested soon after publication.

Mikkel's theory got some support through our next joint paper. I proposed that we look at some lower bounds via the so-called *richness method* for hard problems like partial match or nearest neighbor. After 2 years of predecessor lower bounds, it was a simple exercise to obtain better lower bound by richness; in fact, we felt like we were *simplifying* our technique for beating communication complexity, in order to make it work here. Despite my opinion that this paper was too simple to be interesting, Mikkel convinced me to submit it to FOCS. Sure enough, the paper was accepted to FOCS'06 [88] with raving reviews and a special issue invitation. One is reminded to listen to senior people once in while.

After my initial interaction with Mikkel, I had begun to understand the field, and I was able to make progress on several fronts at the same time. Since my first paper in 2003, I kept working on a very silly dynamic problem: prove a lower bound for partial sums in the bit-probe model. It seemed doubtful that anyone would care, but the problem was so clean and simple, that I couldn't let go. One day, as I was sitting in an MIT library (I was still an undergraduate and didn't have an office), I discovered something entirely unexpected. You see, before my first paper proving an $\Omega(\lg n)$ dynamic lower bound, there had been just one technique for proving dynamic bounds, dating back to Fredman and Saks in 1989. Everybody had tried, and failed, to prove a logarithmic bound by this technique. And there I was, seeing a clear and simple proof of an $\Omega(\lg n)$ bound by this classic technique. This was a great surprise to me, and it had very interesting consequences, including a new bound for my silly little problem, as well as a new record lower bound in the bit-probe model. With the gods clearly on my side (Miltersen was on the PC), this paper [87] got the Best Student Paper award at ICALP.

My work with Mikkel continued with a randomized lower bound for predecessor search (our first bound only applied to deterministic algorithms). We had the moral obligation to "finish" the problem, as Mikkel put it. This took quite some work, but in the end, we succeeded, and the paper [90] appeared in SODA'07.

At that time, I also wrote my first paper with Alex Andoni, an old and very good friend, with whom I would later share an apartment and an office. Surprisingly, it took us until 2006 to get our first paper, and we have only written one other paper since then, despite our very frequent research discussions over beer, or while walking home. These two papers are a severe underestimate of the entertaining research discussions that we have had. I owe Alex a great deal of thanks for the amount of understanding of high dimensional geometry that he has passed on to me, and, above all, for being a highly supportive friend.

Our first paper was, to some extent, a lucky accident. After a visit to my uncle in Philadelphia, I was stuck on a long bus ride back to MIT, when Alex called to say that he had some intuition about why $(1+\varepsilon)$-approximate nearest neighbor should be hard. As always, intuition is hard to convey, but I understood at least that he wanted to think in very high dimensions, and let the points of the database be at constant pairwise distance. Luckily, on that bus ride I was thinking of lower bounds for lopsided set disjointness (a problem left open by the seminal paper of Miltersen et al. [73] from STOC'95). It didn't take long after passing to realize the connection, and I was back on the phone with Alex explaining how his construction can be turned in a reduction from lopsided set disjointness to nearest neighbor.

Back at MIT, the proof obviously got Piotr Indyk very excited. We later merged with another result of his, yielding a FOCS'06 paper [16]. Like in the case of Alex, the influence that Piotr has had on my thinking is not conveyed by our number of joint papers (we only have one). Nonetheless, my interaction with him has been both very enjoyable and very useful, and I am very grateful for his help.

My second paper with Alex Andoni was [13] from FOCS'08. This began in spring 2007 as a series of discussions between me and Piotr Indyk about approximate near neighbor in ℓ_∞, discussions which got me very excited about the problem, but didn't really lead to any solution. By the summer of 2007, I had made up my mind that we had to seek an asymmetric communication lower bound. That summer, both I and Alex were interns at IBM Almaden, and I convinced him to join on long walks on the beautiful hills at Almaden, and discuss this problem. Painfully slowly, we developed an information-theoretic understanding of the best previous upper bound, and an idea about how the lower bound should be proved.

Unfortunately, our plan for the lower bound led us to a rather nontrivial isoperimetric inequality, which we tried to prove for several weeks in the fall of 2007. Our proof strategy seemed to work, more or less, but it led to a very ugly case analysis, so we decided to outsource the theorem to a mathematician. We chose none other than our third housemate, Dorian Croitoru, who came back in a few weeks with a nicer proof (if not lacking in case analysis).

My next paper on the list was my (single author) paper [81] from STOC'07. Ever since we broke the communication barrier with Mikkel Thorup, it had been clear to me that the most important and most exciting implication had to be range query problems, where there is a huge gap between space $O(n^2)$ and space $O(n \operatorname{polylog} n)$. Extending the ideas to these problems was less than obvious, and took quite a bit of time to find the right approach, but eventually I ended up with a proof of an $\Omega(\lg n / \lg \lg n)$ lower bound for range counting in 2 dimensions.

In FOCS'08 I wrote follow-up to this, the (single author) paper [82], which I found very surprising in its techniques. That paper is perhaps the coronation of the work in this thesis, showing that proofs for many interesting lower bounds (both static and dynamic) can be obtained by simple reductions from lopsided set disjointness.

The most surprising lower bound in that paper is perhaps the one for 4-dimensional range reporting. Around 2004–2005, I got motivated to work on range reporting, by talking to Christian Mortensen, then a student at ITU Copenhagen. Christian had some nice results for the 2-dimensional case, and I was interested in proving that the 3-dimensional case is hard. Unfortunately, my proofs always seemed to break down one way or another. Christian eventually left for industry, and the problem slipped from my view.

In 2007, I got a very good explanation for why my lower bound had failed: Yakov Nekrich [79] showed in SoCG'07 a surprising $O((\lg \lg n)^2)$ upper bound for 3 dimensions (making it "easy"). It had never occurred to me that a better upper bound could exist, which in some sense served to remind me why lower bounds are important. I was very excited by this development, and immediately started wondering whether the 4-dimensional would also collapse to near-constant time. My bet was that it wouldn't, but what kind of structure could prove 4 dimensions hard, if 3 dimensions were easy?

Yakov Nekrich and Marek Karpinski invited me for a one-week visit to Bonn in October 2007, which prompted a discussion about the problem, but not much progress towards a lower bound. After my return to MIT, the ideas became increasingly more clear, and by the end of the fall I had an almost complete proof, which nonetheless seemed "boring" (technical) and unsatisfying.

In the spring of 2008, I was trying to submit 4 papers to FOCS, during my job interview season. Needless to say, this was a challenging experience. I had an interview at Google scheduled two days before the deadline, and I made an entirely unrealistic plan to go to New York with the 5am train, after working through the night. By 5am, I had a serious headache and the beginning of a cold, and I had to postpone the interview at the last minute. However, this proved to be a very auspicious incident. As I was lying on an MIT couch the next day trying to recover, I had an entirely unexpected revelation: the 4-dimensional lower bound could be proved by a series of reductions from lopsided set disjointness! This got developed into the set of reductions in [82], and submitted to the conference. I owe my sincere apologies to the Google researchers for messing their schedules with my unrealistic planning. But in the end, I'm glad things hapenned this way, and the incident was clearly good for science.

Contents

Chapter 1

Introduction

1.1 What is the Cell-Probe Model?

Perhaps the best way to understand the cell-probe model is as the "von Neumann bottleneck," a term coined by John Backus in his 1977 Turing Award lecture. In the von Neumann architecture, which is pervasive in today's computers, the memory and the CPU are isolated components of the computer, communicating through a simple interface. At sufficiently high level of abstraction, this interface allows just two functions: reading and writing the atoms of memory (be they words, cache lines, or pages). In Backus' words, "programming is basically planning and detailing the enormous traffic of words through the von Neumann bottleneck."

Formally, let the memory be an array of cells, each having w bits. The data structure is allowed to occupy S consecutive cells of this memory, called "the space." The queries, and, for dynamic data structures, the updates, are implemented through algorithms that read and write cells. The running time of an operation is just the number of cell probes, i.e. the number of reads and writes executed. All computation based on cells that have been read is free.

Formally, the cell-probe model is a "non-uniform" model of computation. For a static data structure the memory representation is an arbitrary function of the input database. The query and update algorithms may maintain unbounded state, which is reset between operations. The query/update algorithms start with a state that stores the parameters of the operation; any information about the data structure must be learned through cell probes. The query and updates algorithms are arbitrary functions that specify the next read and write based on the current state, and specify how the state changes when a value is read.

1.1.1 Examples

Let us now exemplify with two fundamental data structure problems that we want to study. In Chapter 2, we introduce a larger collection of problems that the thesis is preoccupied with.

In the *partial sums* problem, the goal is to maintain an array $A[1 \mathinner{..} n]$ of words (w-bit integers), initialized to zeroes, under the following operations:

UPDATE(k, Δ): modify $A[k] \leftarrow \Delta$.
SUM(k): returns the partial sum $\sum_{i=1}^{k} A[i]$.

The following are three very simple solutions to the problem:

- store that array A in plain form. The update requires 1 cell probe (update the corresponding value), while the query requires up to n cell probes to add values $A[1] + \cdots + A[k]$.
- maintain an array $S[1 \mathinner{..} n]$, where $S[k] = \sum_{i=1}^{k} A[i]$. The update requires $O(n)$ cell probes to recompute all sums, while the query runs in constant time.
- maintain an augmented binary search tree with the array element in the leaves, where each internal node stores the sum of the leaves in its subtree. Both the query time and the update time are $O(\log n)$.

Dynamic problems are characterized by a trade-off between the update time t_u and the query time t_q. Our goal is to establish the optimal trade-off, by describing a data structure and a proof of a matching lower bound.

In most discussion in the literature, one is interested in the "running time per operation", i.e. a single value $\max\{t_u, t_q\}$ which describes the complexity of both operations. In this view, we will say that the partial sums problem admits an upper bound of $O(\lg n)$, and we will seek an $\Omega(\lg n)$ lower bound. Effectively, we want to prove that binary search trees are the optimal solution for this problem.

To give an example of a static problem, consider *predecessor search*. The goal is to preprocess a set A of n integers (of w bits each), and answer the following query efficiently:

PREDECESSOR(x): find $\max\{y \in A \mid y < A\}$.

This problem can be solved with space $O(n)$, and query time $O(\lg n)$: simply store the array in sorted order, and solve the query by binary search. Static problems can always be solved by complete tabulation: storing the answers to all possible queries. For predecessor search, this gives space $O(2^w)$, and constant query time. Later, we will see better upper bounds for this problem.

Static problems are characterized by a trade-off between the space S and the query time t, which we want to understand via matching upper and lower bounds. In the literature, one is often interested in the query time achievable with linear, or near-linear space $S = O(n \operatorname{polylog} n)$.

1.1.2 Predictive Power

For maximal portability, upper bounds for data structures are often designed in the *word RAM* model of computation, which considers any operation from the `C` programming language as "constant time." Sometimes, however, it makes sense to consider variations of the model to allow a closer alignment of the theoretical prediction and practical performance. For example, one might augment the model with a new operation available on a family of CPUs. Alternatively, some operations, like division or multiplication, may be too slow on a

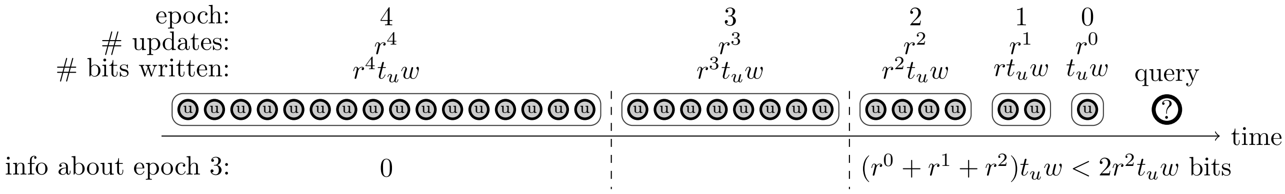


Figure 1-1: Epochs grow exponentially at a rate of r, looking back from the query. Not enough information about epoch 3 is recorded outside it, so the query needs to probe a cell from epoch 3 with constant probability.

particular CPU, and one may exclude them from the set of constant-time operations while designing a new data structure.

By contrast, it has been accepted for at least two decades that the "holy grail" of lower bounds is to show results in the cell-probe model. Since the model ignores details particular to the CPU (which operations are available in the instruction set), a lower bound will apply to any computer that fits the von Neumann model. Furthermore, the lower bound holds for any implementation in a high-level programming language (such as `C`), since these languages enforce a separation between memory and the processing unit.

To obtain realistic lower bounds, it is standard to assume that the cells have $w = \Omega(\lg n)$ bits. This allows pointers and indices to be manipulated in constant time, as is the case on modern computers and programming languages. To model external memory (or CPU caches), one can simply let $w \approx B \lg n$, where B is the size of a page or cache line.

1.2 Overview of Cell-Probe Lower Bounds

The cell-probe model was introduced three decades ago, by Yao [106] in FOCS'78. Unfortunately, his initial lower bounds were not particularly strong, and, as it often happens in Computer Science, the field had to wait for a second compelling paper before it took off.

In the case of cell-probe complexity, this came not as a second paper, but as a pair of almost simultaneous papers appearing two decades ago. In 1988, Ajtai [3] proved a lower bound for a static problem: predecessor search. In STOC'89, Fredman and Saks [51] proved the first lower bounds for dynamic problems, specifically for the partial sums problem and for union-find.

Static and dynamic lower bounds continued as two largely independent threads of research, with the sets of authors working on the two fields being almost disjoint. Dynamic lower bounds relied on the ad hoc *chronogram technique* of Fredman and Saks, while static lower bounds were based on communication complexity, a well-studied field of complexity theory.

1.2.1 Dynamic Bounds

We first sketch the idea behind the technique of Fredman and Saks [51]; see Figure 1-1. The proof begins by generating a sequence of random updates, ended by exactly one query. Looking back in time from the query, we partition the updates into exponentially growing epochs: for a certain r, epoch i contains the r^i updates immediately *before* epoch $i-1$. The goal is to prove that for each i, the query needs to read a cell written in epoch i with constant probability. Then, by linearity of expectation over all epochs, we can bound the expected query time to $t_q = \Omega(\log_r n)$.

Observe that information about epoch i cannot be reflected in earlier epochs (those occurred back in time). On the other hand, the latest $i-1$ epochs contain less than $2 \cdot r^{i-1}$ updates. Assume the cell-probe complexity of each update is bounded by t_u. Then, during the latest $i-1$ epochs, only $2\,r^{i-1}t_u w$ bits are written. Setting $r = C \cdot t_u w$ for a sufficiently large constant C, we have $r^i \gg 2\,r^{i-1}t_u w$, so less than one bit of information is reported about each update in epoch i. Assume that, in the particular computation problem that we want to lower bound, a random query is forced to learn information about a random update from each epoch. Then, the query must read a cell from epoch i with constant probability, because complete information about the needed update is not available outside the epoch. We have a lower bound of $t_q = \Omega(\log_r n) = \Omega(\lg n / \lg(wt_u))$. For the natural case $w = \Theta(\lg n)$, we have $\max\{t_u, t_q\} = \Omega(\lg n / \lg\lg n)$.

All subsequent papers [22, 70, 74, 59, 8, 49, 5, 9, 58] on dynamic lower bounds used this epoch-based approach of Fredman and Saks (sometimes called the "chronogram technique").

Perhaps the most influential in this line of research was that the paper by Alstrup, Husfeldt, and Rauhe [8] from FOCS'98. They proved a bound for the so-called *marked ancestor problem*, and showed many reductions from this single bound to various interesting data-structure problems.

In STOC'99, Alstrup, Ben-Amram, and Rauhe [5] showed optimal trade-offs for the union-find problem, improving on the original bound of Fredman and Saks [51].

1.2.2 Round Elimination

The seminal paper of Ajtai [3] turned out to have a bug invalidating the results, though this did not stop it from being one of the most influential papers in the field. In STOC'94, Miltersen [71] corrected Ajtai's error, and derived a new set of results for the predecessor problem. More importantly, he recasted the proof as a lower bound for asymmetric communication complexity.

The idea here is to consider a communication game in which Alice holds a query and Bob holds a database. The goal of the players is to answer the query on the database, through communication. A cell-probe algorithm immediately gives a communication protocol: in each round, Alice sends $\lg S$ bits (an address), and Bob replies with w bits (the contents of the memory location). Thus, a lower bound on the communication complexity implies a lower bound on the cell-probe complexity.

The technique for proving a communication lower bound, implicit in the work of Ajtai [3] and Miltersen [71], was made explicit in STOC'95 by Miltersen, Nisan, Safra, and Wigderson [73]. This technique is based on the *round elimination lemma*, a result that shows how to eliminate a message in the communication protocol, at the cost of reducing the problem size. If all messages can be eliminated and the final problem size is nontrivial (superconstant), the protocol cannot be correct.

Further work on the predecessor problem included the PhD thesis of Bing Xiao [105], the paper of Beame and Fich from STOC'99 [21], and the paper of Sen and Venkatesh [95]. These authors obtained better lower bounds for predecessor search, by showing tighter versions of round elimination.

Another application of round elimination was described by Chakrabarti, Chazelle, Gum, and Lvov [26] in STOC'99. They considered the d-dimensional approximate nearest neighbor problem, with constant approximation and polynomial space, and proved a time lower bound of $\Omega(\lg\lg d/\lg\lg\lg d)$. This initial bound was deterministic, but Chakrabarti and Regev [27] proved a randomized bound via a new variation of the round elimination lemma.

1.2.3 Richness

Besides explicitly defining the round elimination concept, Miltersen et al. [73] also introduced a second technique for proving asymmetric lower bounds for communication complexity. This technique, called the *richness method*, could prove bounds of the form: either Alice communicates a bits, or Bob communicates b bits (in total over all messages). By converting a data structure with cell-probe complexity t into a communication protocol, Alice will communicate $t\lg S$ bits, and Bob will communicate $t \cdot w$ bits. Thus, we obtain the lower bound $t \geq \min\{\frac{a}{\lg S}, \frac{b}{w}\}$. Typically, the b values in such proofs are extremely large, and the minimum is given by the first term. Thus, the lower bound is $t \geq a/\lg S$, or equivalently $S \geq 2^{a/t}$.

The richness method is usually used for "hard" problems, like nearest neighbor in high dimensions, where we want to show a large (superpolynomial) lower bound on the space. The bounds we can prove, having form $S \geq 2^{a/t}$, are usually very interesting for constant query time, but they degrade very quickly with larger t.

The problems that were analyzed via the richness method were:

- partial match (searching with wildcards), by Borodin, Ostrovsky, and Rabani [25] in STOC'99, and by Jayram, Khot, Kumar, and Rabani [64] in STOC'03.

- exact near neighbor search, by Barkol and Rabani [20] in STOC'00. Note that there exists a reduction from partial match to exact near neighbor. However, the bounds for partial match were not optimal, and Barkol and Rabani chose to attack near neighbor directly.

- deterministic approximate near neighbor, by Liu [69].

1.3 Our Contributions

In our work, we attack all the fronts of cell-probe lower bounds described above.

1.3.1 The Epoch Barrier in Dynamic Lower Bounds

As the natural applications of the chronogram technique were being exhausted, the main barrier in dynamic lower bounds became the chronogram technique itself. By design, the epochs have to grow at a rate of at least $\Omega(t_u)$, which means that the large trade-off that can be shown is $t_q = \Omega(\lg n / \lg t_u)$, making the largest possible bound $\max\{t_u, t_q\} = \Omega(\lg n / \lg\lg n)$.

This creates a rather unfortunate gap in our understanding, since upper bounds of $O(\lg n)$ are in abundance: such bounds are obtained via binary search trees, which are probably the single most useful idea in dynamic data structures. For example, the well-studied partial sums problem had an $\Omega(\lg n / \lg\lg n)$ lower bound from the initial paper of Fredman and Saks [51], and this bound could not be improved to show that binary search trees are optimal.

In 1999, Miltersen [72] surveyed the field of cell-probe complexity, and proposed several challenges for future research. Two of his three "big challenges" asked to prove an $\omega(\lg n / \lg\lg n)$ lower bound for *any* dynamic problem in the cell-probe model, respectively an $\omega(\lg n)$ lower bound for any problem in the bit-probe model (see §1.3.2). Of the remaining four challenges, one asked to prove $\Omega(\lg n)$ for partial sums.

These three challenges are solved in our work, and this thesis. In joint work with Erik Demaine appearing in SODA'04 [84], we introduced a new technique for dynamic lower bounds, which could prove an $\Omega(\lg n)$ lower bound for the partial sums problem. Though this was an important 15-year old open problem, the solution turned out to be extremely clean. The entire proof is some 3 pages of text, and includes essentially no calculation.

In STOC'04 [83], we augmented our technique through some new ideas, and obtained $\Omega(\lg n)$ lower bounds for more problems, including an instance of list manipulation, dynamic connectivity, and dynamic trees. These two publications were merged into a single journal paper [86].

The logarithmic lower bounds are described in Chapter 3. These may constitute the easiest example of a data-structural lower bound, and can be presented to a large audience. Indeed, our proof has been taught in a course by Jeff Erickson at the University of Illinois at Urbana and Champaign, and in a course co-designed by myself and Erik Demaine at MIT.

1.3.2 The Logarithmic Barrier in Bit-Probe Complexity

The bit-probe model of computation is an instantiation of the cell-probe model with cells of $w = 1$ bit. While this lacks the realism of the cell-probe model with $w = \Omega(\lg n)$ bits per cell, it is a clean theoretical model that has been studied often.

In this model, the best dynamic lower bound was $\Omega(\lg n)$, shown by Miltersen [74]. In his survey of cell-probe complexity [72], Miltersen lists obtaining an $\omega(\lg n)$ bound as one of the three "big challenges" for future research.

We address this challenge in our joint work with Tarniţă [87] appearing in ICALP'05. Our proof is based on a surprising discovery: we present a subtle improvement to the classic chronogram technique of Fredman and Saks [51], which, in particular, allows it to prove logarithmic lower bounds in the cell-probe model. Given that the chronogram technique was the only known approach for dynamic lower bounds before our work in 2004, it is surprising to find that the solution to the desired logarithmic bound has always been this close.

To summarize our idea, remember that in the old epoch argument, the information revealed by epochs $1, \dots, i-1$ about epoch i was bounded by the number of cells written in these epochs. We will now switch to a simple alternative bound: the number of cells read during epochs $1, \dots, i-1$ and written during epoch i. This doesn't immediately help, since all cell reads from epoch $i-1$ could read data from epoch i, making these two bounds identical. However, one can randomize the epoch construction, by inserting the query after a *random* number of updates. This randomization "smoothes" out the distribution of epochs from which cells are read, i.e. a query reads $O(t_q/\log_r n)$ cells from every epoch, in expectation over the randomness in the epoch construction. Then, the $O(r^{i-1})$ updates in epochs $1, \dots, i-1$ only read $O(r^{i-1} \cdot t_u/\log_r n)$ cells from epoch i. This is not enough information if $r \gg t_u/\log_r n = \Theta(t_u/t_q)$, which implies $t_q = \Omega(\log_r n) = \Omega(\lg n/\lg \frac{t_u}{t_q})$. From this, it follows that $\max\{t_u, t_q\} = \Omega(\lg n)$.

Our improved epoch technique has an immediate advantage: it is the first technique that can prove a bound for the regime $t_u < t_q$. In particular, we obtain a tight update/query trade-off for the partial sums problem, whereas previous approaches could only attack the case $t_u > t_q$. Furthermore, by carefully using the new lower bound in the regime $t_u < t_q$, we obtain an $\Omega\big((\frac{\lg n}{\lg\lg n})^2\big)$ lower bound in the bit-prove model. This offers the highest known bound in the bit-probe model, and answers Miltersen's challenge.

Intuitively, performance in the bit-probe model should typically be slower by a $\lg n$ factor compared to the cell-probe model. However, our $\widetilde{\Omega}(\lg^2 n)$ bound in the bit-probe world is far from an echo of an $\widetilde{\Omega}(\lg n)$ bound in the cell-probe world. Indeed, $\Omega(\frac{\lg n}{\lg\lg n})$ bounds in the cell-probe model have been known since 1989, but the bit-probe record has remained just the slightly higher $\Omega(\lg n)$. In fact, our bound is the first to show a quasi-optimal $\widetilde{\Omega}(\lg n)$ separation between bit-probe complexity and the cell-probe complexity, for superconstant cell-probe complexity.

Our improved epoch construction, as well as our bit-probe lower bounds, are presented in Chapter 4.

1.3.3 The Communication Barrier in Static Complexity

All bounds for static data structures were shown by transforming the cell-probe algorithm into a communication protocol, and proving a communication lower bound for the problem at hand, either by round elimination, or by the richness method.

Intuitively, we do not expect this relation between cell-probe and communication complexity to be tight. In the communication model, Bob can remember past communication, and may answer new queries based on this. Needless to say, if Bob is just a table of cells, he cannot "remember" anything, and his responses must be a function of Alice's last message (i.e. the address being probed).

The most serious consequence of the reduction to communication complexity is that the lower bounds are insensitive to polynomial changes in the space S. For instance, going from space $S = O(n^3)$ to space $S = O(n)$ only translates into a change in Alice's message size by a factor of 3. In the communication game model, this will increase the number of rounds by at most 3, since Alice can break a longer message into three separate messages, and Bob can remember the intermediate parts.

Unfortunately, this means that communication complexity cannot be used to separate data structures of polynomial space, versus data structures of linear space. By contrast, for many natural data-structure problems, the most interesting behavior occurs close to linear space. In fact, when we consider applications to large data bases, the difference between space $O(n^3)$ and space $O(n \operatorname{polylog} n)$ is essentially the difference between interesting and uninteresting solutions.

In our joint work with Mikkel Thorup [89] appearing in STOC'06, we provided the first separation between data structures of polynomial size, and data structures of linear size. In fact, our technique could prove an asymptotic separation between space $\Theta(n^{1+\varepsilon})$ and space $n^{1+o(1)}$, for any constant $\varepsilon > 0$.

Our idea was to consider a communication game in which Alice holds k independent queries to the same database, for large k. At each step of the cell-probe algorithm, these queries need to probe a *subset* of at most k cells from the space of S cells. Thus, Alice needs to send $O(\lg \binom{S}{k}) = O(k \lg \frac{S}{k})$ bits to describe the memory addresses. For k close to n, and $S = n^{1+o(1)}$, this is asymptotically smaller than $k \lg S$. This means that the "amortized" complexity per query is $o(\lg S)$, and, if our lower bound *per query* is as high as for a single query, we obtain a better cell-probe lower bound. A result in communication complexity showing that the complexity of k independent problems increases by a factor of $\Omega(k)$ compared to one problem is called a *direct sum* result.

In our paper [89], we showed a direct sum result for the round elimination lemma. This implied an optimal lower bound for predecessor search, finally closing this well-studied problem. Since predecessor search admits better upper bounds with space $O(n^2)$ than with space $O(n \operatorname{polylog} n)$, a tight bound could not have been obtained prior to our technique breaking the communication barrier.

For our direct sum version of round elimination, and our improved predecessor bounds, the reader is referred to Chapter 9.

1.3.4 Richness Lower Bounds

A systematic improvement. A broad impact of our work on the richness method was to show that the cell-probe consequence of richness lower bounds had been systematically underestimated: any richness lower bound implies better cell-probe lower bounds than what was previously thought, when the space is $n^{1+o(1)}$. In other words, we can take any lower bound shown by the richness method, and obtain a better lower bound for small-space data structures by *black-box* use of the old result.

Remember that proving Alice must communicate $\Omega(a)$ bits essentially implies a cell-probe lower bound of $t = \Omega(a/\lg S)$. By our trick for beating communication complexity, we will consider $k = O(n/\operatorname{poly}(w))$ queries in parallel, which means that Alice will communicate $O(t \cdot k \lg \frac{S}{k}) = O(tk \lg \frac{Sw}{n})$ bits in total.

Our novel technical result is a direct sum theorem for the richness measure: for any problem f that has an $\Omega(a)$ communication lower bound by richness, k independent copies of f will have a lower bound of $\Omega(k \cdot a)$. This implies that $t \cdot k \lg \frac{Sw}{n} = \Omega(k \cdot a)$, so $t = \Omega(a/\lg \frac{Sw}{n})$. Our new bounds is better than the classic $t = \Omega(a/\lg S)$, for space $n^{1+o(1)}$. In particular, for the most important case of $S = O(n \operatorname{polylog} n)$ and $w = O(\operatorname{polylog} n)$, our improved bound is better by $\Theta(\lg n/\lg\lg n)$.

Our direct sums theorems for richness appeared in joint work with Mikkel Thorup [88] at FOCS'06, and are described in Chapter 5.

Specific problems. Our work has also extended the reach of the richness method, proving lower bound for several interesting problem.

One such problem is lopsided set disjointness (LSD), in which Alice and Bob receive sets S and T ($|S| \ll |T|$), and they want to determine whether $S \cap T = \emptyset$. In their seminal work from STOC'95, Miltersen et al. [73] proved a deterministic lower bound for lopsided set disjointness, and left the randomized case as an "interesting open problem." The first randomized bounds were provided in our joint work with Alexandr Andoni and Piotr Indyk [16], appearing in FOCS'06, and strengthened in our (single-author) paper [82] appearing in FOCS'08. Some of these lower bounds are described in Chapter 5.

Traditionally, LSD was studied for its fundamental appeal, not because of data-structural applications. However, in [82] we showed a simple reduction from LSD to partial match. Despite the work of [73, 25, 64], the best bound for partial match was suboptimal, showing that Alice must communicate $\Omega(d/\lg n)$ bits, where d is the dimension. Our reduction, coupled with our LSD result, imply an optimal bound of $\Omega(d)$ on Alice's communication, closing this issue. The proof of our reduction appears in Chapter 6.

Since partial match reduces to exact near neighbor, we also obtain an optimal communication bound for that problem. This supersedes the work of [20], who had a more complicated proof of a tight lower bound for exact near neighbor.

Turning our attention to $(1+\varepsilon)$-approximate nearest neighbor, we find the existing lower bounds unsatisfactory. The upper bounds for this problem use $n^{O(1/\varepsilon^2)}$ space, by dimensionality reduction. On the other hand, the lower bounds of [26, 27] allowed constant ε and polynomial space, proving a tight time lower bound of $\Omega(\lg\lg d/\lg\lg\lg d)$ in d dimensions. However, from a practical perspective, the main issue with the upper bound is the prohibitive space usage, not the doubly logarithmic query time. To address this concern, our paper with Andoni and Indyk [16] proved a communication lower bound in which Alice's communication is $\Omega(\frac{1}{\varepsilon^2} \lg n)$ bits. This shows that the space must be $n^{\Omega(1/(t\varepsilon^2))}$, for query time t. Our lower bound is once more by reduction from LSD, and can be found in Chapter 6.

Finally, we consider the near neighbor problem in the ℓ_∞ metric, in which Indyk [61] had shown an exotic trade-off between approximation and space, obtaining $O(\lg\lg d)$ approximation for any polynomial space. In our joint work with Andoni and Croitoru [13], we gave a very appealing reformulation of Indyk's algorithm in information-theoretic terms, which made his space/approximation trade-off more natural. Then, we described a richness lower bound for the problem, showing that this space/approximation trade-off is optimal. These results can be found in Chapter 8.

1.3.5 Lower Bounds for Range Queries

By far, the most exciting consequence of our technique for surpassing the communication barrier is that it opens the door to tight bounds for range queries. Such queries are some of the most natural examples of what computers might be queried for, and any statement about their importance is probably superfluous. The introductory lecture of any database course is virtually certain to contain an example like "find employees with a salary between 70000 and 95000, who have been hired between 1998 and 2001."

Orthogonal range queries can be solved in constant time if polynomial space is allowed, by simply precomputing all answers. Thus, any understanding of these problems hinges on our technique to obtain better lower bounds for space $O(n \operatorname{polylog} n)$.

In my papers [81, 82], I prove optimal bounds for range counting in 2 dimensions, stabbing in 2 dimensions, and range reporting in 4 dimensions. Surprisingly, our lower bounds are once more by reduction from LSD.

Range counting problems have traditionally been studied in algebraic models. In the group and semigroup models, each point has an associated weight from an arbitrary commutative (semi)group and the "counting" query asks for the sum of the weights of the points in the range. The data structure can only manipulate weights through the black-box addition operator (and, in the group model, subtraction).

The semigroup model has allowed for every strong lower bounds, and nearly optimal results are known in all dimensions. By contrast, as soon as we allow subtraction (the group model), the lower bounds are very weak, and we did not even have a good bound for 2 dimensions. This rift between our understanding of the semigroup and stronger models has a deeper conceptual explanation. In general, semigroup lower bounds hide arguments of a very geometric flavor. If a value is included in a sum, it can never be taken out again (no subtraction is available), and this immediately places a geometric restriction on the set of ranges that could use the cell. Thus, semigroup lower bounds are essentially bounds on a certain kind of "decomposability" of geometric shapes. On the other hand, bounds in the group or cell-probe model require a different, information theoretic understanding of the problem. In the group model, the sum stored in a cell does not tell us anything in isolation, as terms could easily be subtracted out. The lower bound must now find certain bottlenecks in the manipulation of information that make the problem hard. Essentially, the difference in the type of reasoning behind semigroup 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 semigroup model has been extremely slow.

Proving good bounds outside the restrictive semigroup model has long been recognized as

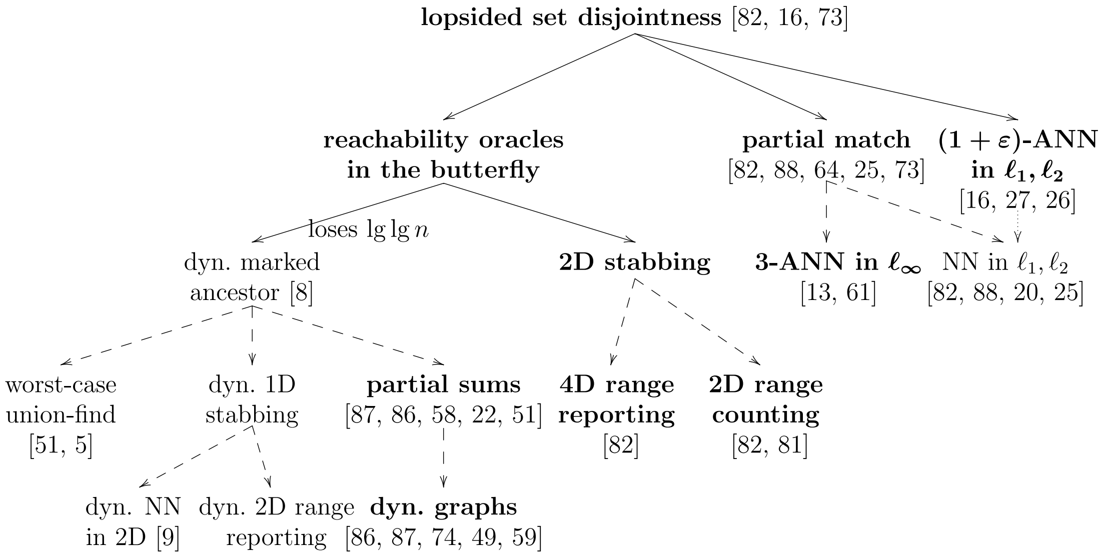


Figure 1-2: Dashed lines indicate reductions that were already known, while solid lines indicate novel reductions. For problems in bold, the results of this thesis are stronger than what was previously known. Citations indicate work on lower bounds.

an important challenge. As early as 1982, Fredman [48] asked for better bounds in the group model for dynamic range counting. In FOCS'86, Chazelle [31] echoed this, and also asked about the static case. Our results answer these old challenges, at least in the 2-dimensional case.

1.3.6 Simple Proofs

An important contribution of our work is to show clean, simple proofs of lower bounds, in an area of Theoretical Computer Science that is often dominated by exhausting technical details.

Perhaps the best illustration of this contribution is in Figure 1-2, which shows the lower bounds that can be obtained by reduction from lopsided set disjointness. (The reader unfamiliar with the problems can consult Chapter 2.)

This reduction tree is able to unify a large fraction of the known results in dynamic data structures and static data structures (both in the case of large space, and for near-linear space). In many cases, our proofs by reduction considerably simplify the previous proofs. Putting all the work in a single lower bound has exciting consequences from the teaching perspective: instead of presenting many different lower bounds (even "simple" ones are seldom light on technical detail!), we can now present many interesting results through clean algorithmic reductions, which are easier to grasp.

Looking at the reduction tree, it may seem hard to imagine a formal connection between lower bounds for such different problems, in such different settings. Much of the magic of our results lies in considering a somewhat artificial problem, which nonetheless gives the right link between the problems: reachability queries in butterfly graphs. Once we decide to use this middle ground, it is not hard to give reductions to and from set disjointness, dynamic marked ancestor, and static 4-dimensional range reporting. Each of these reductions is natural, but the combination is no less surprising.

These reductions are described in Chapters 6 and 7.

Chapter 2

Catalog of Problems

This section introduces the various data-structure problems that we consider throughout this thesis. It is hoped that the discussion of each topic is sufficiently self-contained, that the reader can quickly jump to his or her topic of interest.

2.1 Predecessor Search

2.1.1 Flavors of Integer Search

Binary search is certainly one of the most fundamental ideas in computer science — but what do we use it for? Given a set S with $|S| = n$ values from some ordered universe U, the following are natural and often-asked queries the can be solved by binary search:

exact search: given some $x \in U$, is $x \in S$?

predecessor search: given some $x \in U$, find $\max\{y \in S \mid y < S\}$, i.e. the predecessor of x in the set S.

1D range reporting: given some interval $[x, y]$, report all/one of the points in $S \cap [x, y]$.

Exact search is ubiquitous in computer science, while range reporting is a natural and important database query. Though at first sight predecessor search may seem less natural, it may in fact be the most widely used of the three. Some of its many applications include:

- IP-lookup, perhaps the most important application, is the basic query that must be answered by every Internet router when forwarding a packet (which probably makes it the most executed algorithmic problem in the world). The problem is equivalent to predecessor search [44].
- to make data structures persistent, each cell is replaced with a list of updates. Deciding which version to use is a predecessor problem.
- orthogonal range queries start by converting any universe U into "rank space" $\{1, \ldots, n\}$. The dependence on the universe becomes an additive term in the running time.
- the succinct incarnation of predecessor search is known as the "rank/select problem," and it underlies most succinct data structures.

- $(1+\varepsilon)$-approximate near neighbor in any constant dimension (with the normal Euclidean metric) is equivalent to predecessor search [28].
- in the emergency planning version of dynamic connectivity, a query is equivalent to predecessor search; see our work with Mikkel Thorup [85].

It is possible to study search problems in an arbitrary universe U endowed with a comparison operation (the so called *comparison model*), in which case the $\Theta(\lg n)$ running time of binary search is easily seen to be optimal. However, this "optimality" of binary search is misleading, given that actual computers represent data in some well-specified formats with bounded precision.

The most natural universe is $U = \{0, \dots, 2^w - 1\}$, capturing the assumption that values are arbitrary w-bit integers that fit in a machine word. The standard floating point representation (IEEE 754, 1985) was designed so that two w-bit floating point values compare the same as two w-bit integers. Thus, any algorithm for search queries that applies to integers carries over to floating point numbers. Furthermore, most algorithms apply naturally to strings, which can be interpreted as numbers of high, variable precision.

By far, the best known use of bounded precision is hashing. In fact, when one thinks of exact search queries, the first solution that comes to mind is probably hash tables, not binary search. Hash tables provide an optimal solution to this problem, at least if we accept randomization in the construction. Consider the following theorem due to Fredman, Komlós, and Szemerédi [50]:

Theorem 2.1. *There exists a dictionary data structure using $O(n)$ words of memory that answers exact search queries deterministically in $O(1)$ time. The data structure can be constructed by a randomized algorithm in $O(n)$ time with high probability.*

Dietzfelbinger and Meyer auf der Heide [39] give a dynamic dictionary that implements queries deterministically in constant time, while the randomized updates run in constant time with high probability.

By contrast, constant time is not possible for predecessor search. The problem has been studied intensely in the past 20 years, and it was only in our recent work with Mikkel Thorup [89, 90] that the optimal bounds were understood. We discuss the history of this problem in the next section.

One-dimensional range reporting remains the least understood of the three problems. In our work with Christian Mortensen and Rasmus Pagh [76], we have shown surprising upper bounds for dynamic range reporting in one dimension, with a query time that is exponentially faster than the optimal query time for predecessor search. However, there is currently no lower bound for the problem, and we do not know whether it requires superconstant time per operation.

2.1.2 Previous Upper Bounds

The famous data structure of van Emde Boas [101] from FOCS'75 can solve predecessor search in $O(\lg w) = O(\lg\lg U)$ time, using linear space [103]. The main idea of this data structure is to binary search for the longest common prefix between the query and a value in the database.

It is interesting to note that the solution of van Emde Boas remains very relevant in modern times. IP look-up has received considerable attention in the networking community, since a very fast solution is needed for the router to keep up with the connection speed. Research [44, 36, 102, 1] on software implementations of IP look-up has rediscovered the van Emde Boas solution, and engineered it into very efficient practical solutions.

In external memory with pages of B words, predecessor search can be solved by B-trees in time $O(\log_B n)$. In STOC'90, Fredman and Willard [52] introduced fusion trees, which use an ingenious packing of multiple values into a single word to simulate a page of size $B = w^\varepsilon$. Fusion trees solve predecessor search in linear space and $O(\log_w n)$ query time. Since $w = \Omega(\lg n)$, the search time is always $O(\lg n/ \lg\lg n)$, i.e. fusion trees are always asymptotically better than binary search. In fact, taking the best of fusion trees and van Emde Boas yields a search time of $O(\min\{\frac{\lg n}{\lg w}, \lg w\}) \le O(\sqrt{\lg n})$. Variations on fusion trees include [53, 104, 11, 10], though the $O(\log_w n)$ query time is not improved.

In 1999, Beame and Fich [21] found a theoretical improvement to van Emde Boas' data structure, bringing the search time down to $O(\frac{\lg w}{\lg\lg w})$. Combined with fusion trees, this gave them a bound of $O(\min\{\frac{\lg n}{\lg w}, \frac{\lg w}{\lg\lg w}\}) \le O(\sqrt{\frac{\lg n}{\lg\lg n}})$. Unfortunately, the new data structure of Beame and Fich uses $O(n^2)$ space, and their main open problems asked whether the space could be improved to (near) linear.

The *exponential trees* of Andersson and Thorup [12] are an intriguing construction that uses a predecessor structure for n^γ "splitters," and recurses in each bucket of $O(n^{1-\gamma})$ elements found between pairs of splitters. Given any predecessor structure with polynomial space and query time $t_q \ge \lg^\varepsilon n$, this idea can improve it in a black-box fashion to use $O(n)$ space and $O(t_q)$ query time. Unfortunately, applied to the construction of Beame and Fich, exponential trees cannot reduce the space to linear without increasing the query to $O(\lg w)$. Nonetheless, exponential trees seem to have generated optimism that the van Emde Boas bound can be improved.

Our lower bounds, which are joint work with Mikkel Thorup [89, 90] and appear Chapter 9, refute this possibility and answer the question of Beame and Fich in the negative. We show that for near-linear space, such as space $n \cdot \lg^{O(1)} n$, the best running time is essentially the minimum of the two classic solutions: fusion trees and van Emde Boas.

2.1.3 Previous Lower Bounds

Ajtai [3] was the first to prove a superconstant lower bound. His results, with a correction by Miltersen [71], show that for polynomial space, there exists n as a function of w making the query time $\Omega(\sqrt{\lg w})$, and likewise there exists w a function of n making the query complexity $\Omega(\sqrt[3]{\lg n})$.

Miltersen et al. [73] revisited Ajtai's proof, extending it to randomized algorithms. More importantly, they captured the essence of the proof in an independent *round elimination lemma*, which is an important tool for proving lower bounds in asymmetric communication.

Beame and Fich [21] improved Ajtai's lower bounds to $\Omega(\frac{\lg w}{\lg\lg w})$ and $\Omega(\sqrt{\frac{\lg n}{\lg\lg n}})$ respectively. Sen and Venkatesh [95] later gave an improved round elimination lemma, which extended the lower bounds of Beame and Fich to randomized algorithms.

Chakrabarti and Regev [27] introduced an alternative to round elimination, the message compression lemma. As we showed in [89], this lemma can be used to derive an optimal space/time trade-off when the space is $S \geq n^{1+\varepsilon}$. In this case, the optimal query time turns out to be $\Theta\big(\min\big\{\log_w n, \frac{\lg w}{\lg\lg S}\big\}\big)$.

Our result seems to have gone against the standard thinking in the community. Sen and Venkatesh [95] asked whether message compression is really needed for the result of Chakrabarti and Regev [27], or it could be replaced by standard round elimination. By contrast, our result shows that message compression is essential even for classic predecessor lower bounds.

It is interesting to note that the lower bounds for predecessor search hold, by reductions, for all applications mentioned in the previous section. To facilitate these reductions, the lower bounds are in fact shown for the *colored predecessor* problem: the values in S are colored red or blue, and the query only needs to return the *color* of the predecessor.

2.1.4 Our Optimal Bounds

Our results from [89, 90] give an optimal trade-off between the query time and the space S. Letting $a = \lg\frac{S\cdot w}{n}$, the query time is, up to constant factors:

$$\min\begin{cases} \log_w n \\ \lg\frac{w-\lg n}{a} \\ \dfrac{\lg\frac{w}{a}}{\lg\left(\frac{a}{\lg n}\cdot\lg\frac{w}{a}\right)} \\ \dfrac{\lg\frac{w}{a}}{\lg\left(\lg\frac{w}{a}\;/\lg\frac{\lg n}{a}\right)} \end{cases} \tag{2.1}$$

The upper bounds are achieved by a deterministic query algorithm. For any space S, the data structure can be constructed in time $O(S)$ with high probability, starting from a sorted list of integers. In addition, our data structure supports efficient dynamic updates: if t_q is the query time, the (randomized) update time is $O(t_q + \frac{S}{n})$ with high probability. Thus, besides locating the element through one predecessor query, updates change a minimal fraction of the data structure.

In the external memory model with pages of B words, an additional term of $\log_B n$ is added to (2.1). Thus, our result shows that it is always optimal to either use the standard B-trees, or ignore external memory completely, and use the best word RAM strategy.

For space $S = n\cdot\mathrm{poly}(w\lg n)$ and $w \geq (1+\varepsilon)\lg n$, the trade-off is simplified to:

$$\min\Big\{\ \log_w n,\ \ \lg w/\lg\frac{w}{\lg\lg n}\ \Big\} \tag{2.2}$$

The first branch corresponds to fusion trees. The second branch demonstrates that van Emde Boas cannot be improved in general (for all word sizes), though a very small improvement can be made for word size $w > (\lg n)^{\omega(1)}$. This improvement is described in our paper with Mikkel Thorup [89].

Our optimal lower bound required a significant conceptual development in the field. Previously, all lower bounds were shown via communication complexity, which fundamentally cannot prove a separation between data structures of polynomial size and data structures of linear size (see Chapter 1). This separation is needed to understand predecessor search, since Beame and Fich beat the van Emde Boas bound using quadratic space.

The key idea is to analyze many queries simultaneously, and prove a direct-sum version of the round elimination lemma. As opposed to previous proofs, our results cannot afford to increase the distributional error probability. Thus, a second conceptual idea is to consider a stronger model for the induction on cell probes: in our model, the algorithm is allowed to *reject* a large fraction of the queries before starting to make probes.

This thesis. Since this thesis focuses of lower bounds, we omit a discussion of the upper bounds. Furthermore, we want to avoid the calculations involved in deriving the entire trade-off of (2.1), and keep the thesis focused on the techniques. Thus, we will only prove the lower bound for the case $w = \Theta(\lg n)$, which is enough to demonstrate the optimality of van Emde Boas, and a separation between linear and polynomial space. The reader interested in the details of the entire trade-off calculation is referred to our publication [89].

In Chapter 9, we first review fusion trees and the van Emde Boas data structure from an information-theoretic perspective, building intuition for our lower bound. Then, we prove our direct-sum round elimination for multiple queries, which implies the full trade-off (2.1) (though, we only include the calculation for $w = 3 \lg n$).

2.2 Dynamic Problems

2.2.1 Maintaining Partial Sums

This problem, often described as "maintaining a dynamic histogram," asks to maintain an array $A[1 \mathinner{..} n]$ of integers, subject to:

UPDATE(k, Δ)**:** modify $A[k] \leftarrow \Delta$, where Δ is an arbitrary w-bit integer.

SUM(k)**:** returns the "partial sum" $\sum_{i=1}^{k} A[i]$.

This problem is a prototypical application of *augmented* binary search trees (BSTs), which give an $O(\lg n)$ upper bound. The idea is to consider a fixed balanced binary tree with n leaves, which represent the n values of the array. Each internal node is augmented with a value equal to the sum of the values in its children (equivalently, the sum of all leaves in the node's subtree).

A large body of work in lower bounds has been dedicated to understanding the partial sums problem, and in fact, it is by far the best studied dynamic problem. This should not come as a surprise, since augmented BSTs are a fundamental technique in dynamic data structures, and the partial sums problem likely offers the cleanest setting in which this technique can be studied.

2.2.2 Previous Lower Bounds

The partial-sums problem has been a favorite target for new lower bound ideas since the dawn of data structures. Early efforts concentrated on algebraic models of computation. In the semigroup or group models, the elements of the array come from a black-box (semi)group. The algorithm can only manipulate the Δ inputs through additions and, in the group model, subtractions; all other computations in terms of the indices touched by the operations are free.

In the semigroup model, Fredman [47] gave a tight $\Omega(\lg n)$ bound. Since additive inverses do not exist in the semigroup model, an update $A[i] \leftarrow \Delta$ invalidates all memory cells storing sums containing the old value of $A[i]$. If updates have the form $A[i] \leftarrow A[i] + \Delta$, Yao [107] proved a lower bound of $\Omega(\lg n / \lg\lg n)$. Finally, in FOCS'93, Hampapuram and Fredman [54] proved an $\Omega(\lg n)$ lower bound for this version of the problem.

In the group model, a tight bound (including the lead constant) was given by Fredman [48] for the restricted class of "oblivious" algorithms, whose behavior can be described by matrix multiplication. In STOC'89, Fredman and Saks [51] gave an $\Omega(\lg n / \lg\lg n)$ bound for the general case, which remained the best known before our work.

In fact, the results of Fredman and Saks [51] also contained the first dynamic lower bounds in the cell-probe model. Their lower bound trade-off between the update time t_u and query time t_q states that $t_q = \Omega(\lg n / \lg(w + t_u + \lg n))$, which implies that $\max\{t_u, t_q\} = \Omega(\lg n / \lg(w + \lg n))$.

In FOCS'91, Ben-Amram and Galil [22] reproved the lower bounds of Fredman and Saks in a more formalized framework, centered around the concepts of problem and output variability. Using these ideas, they showed [23] $\Omega(\lg n / \lg\lg n)$ lower bounds in more complicated algebraic models (allowing multiplication, etc).

Fredman and Henzinger [49] reproved the lower bounds of [51] for a more restricted version of partial sums, which allowed them to construct reductions to dynamic connectivity, dynamic planarity testing, and other graph problems. Husfeldt, Rauhe, Skyum [59] also explored variations of the lower bound allowing reduction to dynamic graph problems.

Husfeldt and Rauhe [58] gave the first technical improvement after [51], by proving that the lower bounds hold even for nondeterministic query algorithms. A stronger improvement was given in FOCS'98 by Alstrup, Husfeldt, and Rauhe [8], who prove that $t_q \lg t_u = \Omega(\lg n)$. While this bound cannot improve $\max\{t_u, t_q\} = \Omega(\lg n / \lg\lg n)$, it at least implies $t_q = \Omega(\lg n)$ when the update time is constant.

The partial sums problem has also been considered in the bit-probe model of computation, where each $A[i]$ is a bit. Fredman [48] showed a bound of $\Omega(\lg n / \lg\lg n)$ for this case. The highest bound for *any* dynamic problem in the bit-probe model was $\Omega(\lg n)$, due to Miltersen et al. [74].

In 1999, Miltersen [72] surveyed the field of cell-probe complexity, and proposed several challenges for future research. Two of his three "big challenges" asked to prove an $\omega(\lg n/\lg\lg n)$ lower bound for any dynamic problem in the cell-probe model, respectively an $\omega(\lg n)$ lower bound for any problem in the bit-probe model. Of the remaining four challenges, one asked to prove $\Omega(\lg n)$ for partial sums, at least in the group model of computation.

2.2.3 Our Results

In our work with Erik Demaine [86, 87], we broke these barriers in cell-probe and bit-probe complexity, and showed optimal bounds for the partial sums problem.

In Chapter 3, we describe a very simple proof of an $\Omega(\lg n)$ lower bound for partial sums, finally showing that the natural upper bound of binary search trees is optimal. The cleanness of this approach (the proof is roughly 3 pages of text, involving no calculation) stands in sharp contrast to the 15 years that the problem remained open.

In Chapter 4, we prove a versatile trade-off for partial sums, via a subtle variation to the classic chronogram technique of Fredman and Saks [51]. This trade-off is optimal in the cell-probe model, in particular reproving the $\Omega(\lg n)$ lower bound. It is surprising to find that the answer to Miltersen's big challenges consists of a small variation of what was already know.

Our trade-off also allows us to show an $\Omega(\lg n/\lg\lg\lg n)$ bound for partial sums in the bit-probe model, improving on Fredman's 1982 result [48]. Finally, it allows us to show an $\Omega\big((\frac{\lg n}{\lg\lg n})^2\big)$ lower bound in the bit-probe model, for dynamic connectivity. This gives a new record for lower bounds in the bit-probe model, addressing Miltersen's challenge to show an $\omega(\lg n)$ bound.

Intuitively, performance in the bit-probe model should typically be slower by a $\lg n$ factor compared to the cell-probe model. However, our $\widetilde{\Omega}(\lg^2 n)$ bound in the bit-probe world is far from an echo of an $\widetilde{\Omega}(\lg n)$ bound in the cell-probe world. Indeed, $\Omega(\frac{\lg n}{\lg\lg n})$ bounds in the cell-probe model have been known since 1989, but the bit-probe record has remained just the slightly higher $\Omega(\lg n)$. In fact, our bound is the first to show a quasi-optimal $\widetilde{\Omega}(\lg n)$ separation between bit-probe complexity and the cell-probe complexity, for superconstant cell-probe complexity.

2.2.4 Related Problems

List manipulation. In practice, one often seeks a cross between linked lists and array: maintaing a collection of items under typical linked-list operations, with the additional indexing operation (that is typical of array). We can consider the following natural operations, ignoring all details of what constitutes a "record" (all manipulation is done through pointers to black-box records):

INSERT(p, r)**:** insert record r immediately after record p in its list.

DELETE(r)**:** delete record r from its list.

LINK(h_1, h_2): concatenate two lists, identified by their head records h_1 and h_2.
CUT(h, p): split the list with head at h into two lists, the second beginning from record p.
INDEX(k, h): return a pointer to the k-th value in the list with header at h.
RANK(p): return the rank (position) of record p in its list.
FIND(p): return the head of the list that contains record p.

All these operations, and many variants thereof, can be solved in $O(\lg n)$ time per operation using augmented binary search trees. For INSERT/DELETE updates, the binary search tree algorithm must be able to maintain balance in a dynamic tree. Such algorithms abound (consider red-black trees, AVL trees, splay trees, etc). For LINK/CUT, the tree algorithm must support SPLIT and MERGE operations. Many dynamic trees support these operations in $O(\lg n)$ time, including 2-3-4 trees, AVL trees, splay trees, etc.

It turns out that there is a complexity gap between list manipulation with LINK/CUT operations, and list manipulation with INSERT/DELETE. In the former case, updates can make large topological changes to the lists, while in the latter, the updates only have a very local record-wise effect.

If the set of operations includes LINK, CUT, and any of the three queries (including the minimalist FIND), we can show an $\Omega(\lg n)$ lower bound for list manipulation. Thus, augmented binary search trees give an optimal solution. The lower bound is described in Chapter 3, and needs a few interesting ideas beyond the partial-sums lower bound.

If the set of updates is restricted to INSERT and DELETE (with all queries allowed), the problem admits an $O(\lg n / \lg\lg n)$ solution, as proved by Dietz [38]. This restricted case of list manipulation can be shown equivalent to the partial sums problem, in which every array element is $A[i] \in \{0, 1\}$. The original lower bound of Fredman and Saks [51], as well as our trade-off from Chapter 4 show a tight $\Omega(\lg n / \lg\lg n)$ bound for restricted partial sums problem. By reduction, this implies a tight bound for list manipulation with INSERT/DELETE.

Dynamic connectivity. Dynamic graph problems ask to maintain a graph under various operations (typically edge insertions and deletions), and queries for typical graph properties (connectivity of two vertices, MST, shortest path, etc). This has been a very active area of research in the past two decades, and many surprising results have appeared.

Dynamic connectivity is the most basic dynamic graph problem. It asks to maintain an undirected graph with a fixed set of n vertices subject to the following operations:
INSERT(u, v): insert an edge (u, v) into the graph.
DELETE(u, v): delete the edge (u, v) from the graph.
CONNECTED(u, v): test whether u and v lie in the same connected component.

If the graph is always a forest, Sleator and Tarjan's classic "dynamic trees" [97] achieve an $O(\lg n)$ upper bound. A simpler solution is given by Euler tour trees [55], which show that the problem is equivalent to list manipulation. If we maintain the Euler tour of each tree as a list, then an INSERT or DELETE can be translated to $O(1)$ LINK/CUT operations, and the query can be solved by comparing the results of two FIND queries.

For general graphs, the first to achieve polylogarithmic time per operation were Henzinger and King [55], who gave an $O(\lg^3 n)$ running time, using randomization and amortization. Henzinger and Thorup [56] improve the bound to $O(\lg^2 n)$. Holm, de Lichtenberg, and Thorup [57] gave a simple deterministic solution with the same amortized running time, $O(\lg^2 n)$. The best known result is by Thorup [99], achieving very close to logarithmic time: $O(\lg n \cdot (\lg \lg n)^3)$. For plane graphs, Eppstein et al. [40] gave an $O(\lg n)$ upper bound.

Our results from Chapter 3 imply an $\Omega(\lg n)$ lower bound for dynamic connectivity, even in forests and plane graphs. It is easy to show by reductions that our bounds hold for many dynamic graph problems, including dynamic MST, dynamic planarity testing, etc.

2.3 Range Queries

Range-query problems include some of the most natural and fundamental problems in computational geometry and databases. The goal is to represent a set of n objects in d dimensions, such that certain queries about objects that intersect a given *range* can be answered efficiently. In this line of research, the dimension d is a small constant (2, 3, 4, ...), and constants in the O-notation may depend on d.

Usual choices for the query include *counting* the number of points in the range, *reporting* all points in the range, and *existential* queries (test whether the range contains any point, or is empty).

By far, the most common instantiation of the problem is when the n objects are *points*, and the range is an axis-parallel rectangle $[a_1, b_1] \times \cdots \times [a_d, b_d]$. This problem is usually termed *orthogonal range queries*, though the choice is so common that talking about range queries without further qualification usually means orthogonal range queries. Such queries are some of the most natural examples of what computers might be queried for. The introductory lecture of any database course is virtually certain to contain an example like "find employees with a salary between 70000 and 95000, who have been hired between 1998 and 2001."

An important special case of orthogonal range queries consists of *dominance queries*, where query rectangles have the form $[0, b_1] \times \cdots \times [0, b_d]$. One may also consider ranges of a more geometric flavor, including half-spaces, simplices, balls etc.

A second common instantiation of range queries are the stabbing problems, where that query asks for the objects stabbed by a point. Typically, the objects are axis-aligned boxes.

Range queries have been studied intensively, and an overview is well beyond the scope of this work. Instead, we refer the reader to a survey by Agarwal [2]. Below, we discuss mainly the known lower bounds, as well as our new results.

General flavor. The general flavor of all upper bound results is given by that standard use of range trees. This idea can raise a d-dimensional solution to a solution in $d+1$ dimensions, paying a factor of roughly $O(\lg n)$ in time and space. It is generally believed that this cost for each additional the dimension is optimal. Unfortunately, we cannot prove optimal lower bounds for large d, since current lower bound techniques cannot show superlogarithmic bounds in the cell-probe model. Then, it remains to ask about optimal bounds for small dimension.

Another artifact of our slow progress on lower bounds is that we cannot separate space S from, say, space $S \cdot \lg n$ in the cell-probe model. Thus, while it would be interesting to show that the space needs to grow with the dimension, all work has concentrated on showing growth in the time bound. For static lower bounds, one just assumes that the space is an arbitrary $O(n \cdot \text{polylog}\, n)$.

There is also a question about the universe of the coordinate values. Using a predecessor structure for each of the d coordinates, it is possible to reduce all coordinates to *rank space*, $\{1, \dots, n\}$. Thus, the bounds only depend additively on the original universe.

On the other hand, colored predecessor search can be reduced to essentially any range query, including dominance range reporting in 2 dimensions. Thus, the complexity of range queries must depend additively on the predecessor complexity. Since the predecessor bound is well understood (see §2.1), we will eliminate this noise from the bounds, and assume all coordinate values are originally in the range $\{1, \dots, n\}$. In almost all cases, the predecessor bound is a asymptotically smaller, so the additional term is inconsequential anyway.

2.3.1 Orthogonal Range Counting

As mentioned before, these queries ask to count the points inside a range $[a_1, b_1] \times \cdots \times [a_d, b_d]$. As with the partial sums problem, range counting has been traditionally studied in algebraic models. In the group and semigroup models, each point has an associated weight from an arbitrary commutative (semi)group and the "counting" query asks for the sum of the weights of the points in the range. The data structure can only manipulate weights through the black-box addition (and, in the group model, subtraction), and must work for any choice of the (semi)group. The running time of a query is the number of algebraic operations performed. Any other computation, i.e. planning algebraic operations based on coordinates, is free.

The semigroup model has allowed for every strong lower bounds. As early as 1981, Fredman [47] showed that dynamic range reporting (with INSERT and DELETE operations) has a lower bound of $\Omega(\lg^d n)$ in the semigroup model. The DELETE operation is particularly convenient for lower bound in the semigroup model, because any memory cell storing a sum that contains the deleted element in now entirely useless (there is no way to subtract the deleted element). With only INSERTs allowed, Yao [107] showed a lower bound of $\Omega(\lg n / \lg\lg n)$ for dimension $d = 1$. This was significantly strengthened by Chazelle [31], who showed a dynamic lower bound of $\Omega\big((\lg n / \lg\lg n)^d\big)$ in d dimensions, and a static lower bound of $\Omega\big((\lg n / \lg\lg n)^{d-1}\big)$.

In recent years, different types of nonorthogonal ranges were analyzed in the semigroup model, and very strong lower bounds were shown. See, for instance, the survey of Agarwal [2].

By contrast, the lower bounds in the group model have remained quite weak. Not only do known lower bounds fail to grow appropriately with the dimension, but we cannot even get satisfactory bounds in, say, 2 dimensions. No static lower bound had been proved before our work, perhaps with the exception of a result by Chazelle [32]. This result states that for

n input points and n query ranges in 2D, the offline problem takes $\Omega(n \lg\lg n)$ time. This implies that any static data structure that can be *constructed* in $o(n \lg\lg n)$ time, requires query time $\Omega(\lg\lg n)$ — a result that is exponentially weaker than the upper bound.

The group/semigroup gap. Why is there such a rift between our lower-bound abilities in the semigroup and stronger models? In general, semigroup lower bounds hide arguments of a very geometric flavor. To see why, note than when a value is included in a sum, it can never be taken out again (no subtraction is available). In particular, if a the sum stored in one cell includes an input point, this immediately places a geometric restriction on the set of ranges that could use the cell (the range must include the input point). Thus, semigroup lower bounds are essentially bounds on a certain kind of "decomposability" of geometric shapes.

On the other hand, bounds in the group or cell-probe model require a different, information theoretic understanding of the problem. In the group model, the sum stored in a cell does not tell us anything, as terms could easily be subtracted out. The lower bound must now find certain bottlenecks in the manipulation of information that make the problem hard. On the bright side, when such bottlenecks were found, it was generally possible to use them both for group-model lower bounds (arguing about dimensionality in vector spaces), and for cell-probe lower bounds (arguing about entropy).

Philosophically speaking, the difference in the type of reasoning behind semigroup 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 semigroup model has been extremely slow.

As one might expect, proving good bounds outside the restrictive semigroup model has been recognized as an important challenge for a long time. As early as 1982, Fredman [48] asked for better bounds in the group model for dynamic range counting. In FOCS'86, Chazelle [31] echoed this, and also asked about the static case.

Our results. We address the challenges of Fredman and Chazelle, and obtain an almost perfect understanding the range counting in 2 dimensions. See Table 2.1 for a summary of the known and new results.

The upper bounds have been stated in a variety of sources; see e.g. [31]. The following theorems give formal statements for our lower bounds. We note that the trade-offs we obtain are very similar to the ones known in the semigroup model. The space/time trade-offs are known to be tight for space $\Omega(n \lg^{1+\varepsilon} n)$. The query/time trade-off is tight for update time $t_u = \Omega(\lg^{2+\varepsilon} n)$. Lower bounds are shown for a fixed set of input points in $[n]^2$, and *dominance* queries.

Theorem 2.2. *In the group model, a static data structure of size $n \cdot \sigma$ must take $\Omega(\frac{\lg n}{\lg \sigma + \lg\lg n})$ expected time for dominance counting queries.*

Theorem 2.3. *In the cell-probe model with w-bit cells, a deterministic static data structure of size $n \cdot \sigma$ must take $\Omega(\frac{\lg n}{\lg \sigma + \lg w})$ time for dominance counting queries.*

Problem	Model	Lower Bounds		Dimension
static	semigroup	$\Omega\big((\lg n/\lg\lg n)^{d-1}\big)$	[31]	
	group	$\Omega(\lg\lg n)$ $\star$	[32]	$d=2$
$O\big((\frac{\lg n}{\lg\lg n})^{d-1}\big)$		$\boldsymbol{\Omega(\lg n/\lg\lg n)}$	**new**	$d=2$
	cell-probe	$\Omega(\lg\lg n)$	[90]	$d=2$
		$\boldsymbol{\Omega(\lg n/\lg\lg n)}$	**new**	$d=2$
dynamic	semigroup, with DELETE	$\Omega(\lg^d n)$	[47]	
	semigroup, no DELETE	$\Omega(\lg n/\lg\lg n)$	[107]	$d=1$
$O(\lg^d n)$		$\Omega\big((\lg n/\lg\lg n)^d\big)$	[31]	
		$\Omega(\lg n)$	[54]	$d=1$
	group	$\Omega(\lg n/\lg\lg n)$	[51]	$d=1$
		$\Omega(\lg n)$	[86]	$d=1$
		$\boldsymbol{\Omega\big((\lg n/\lg\lg n)^2\big)}$	**new**	$d=2$

Table 2.1: Old and new results for orthogonal range counting. The upper bound for static data structures assumes $O(n \operatorname{polylog} n)$ space. ($\star$) The bound of [32], starred, says that for n input points and n queries, the offline problem takes $\Omega(n \lg\lg n)$ time.

Theorem 2.4. *In the group model, a dynamic data structure which supports updates in expected time t_u requires $t_q = \Omega\big((\frac{\lg n}{\lg t_u + \lg\lg n})^2\big)$ expected time for dominance counting queries.*

Unfortunately, our techniques cannot obtain better bounds in higher dimensions. Depending on mood, the reader may view this as a breakthrough (e.g. providing the first convincingly superconstant bounds for the static case), or as a lucky discovery that moves the borderline of the big open problem from $d=2$ to $d=3$. We believe there is truth in both views.

Another unfortunate limitation is that we cannot obtain an $\widetilde{\Omega}(\lg^2 n)$ bound in the cell-probe model.

2.3.2 Range Reporting

As mentioned before, range reporting is the problem of reporting the points in a box $[a_1, b_1] \times \cdots \times [a_d, b_d]$. Reporting k points must take $\Omega(k)$ time, so, to avoid this technicality, our lower bounds only deal with the existential range reporting problem (report whether there exists any point in the range).

Range reporting has enjoyed some attention recently. In FOCS'00, Alstrup, Brodal, and Rauhe [7] showed how to solve static 2D range reporting in $O(\lg\lg n)$ time and almost linear space.

Dynamic range reporting in 2D has a lower bound of $\Omega(\lg n/\lg\lg n)$, as shown by Alstrup, Husfeldt, and Rauhe [8] in FOCS'98. Their proof goes via the marked ancestor problem, and dynamic stabbing in 1D; see §2.3.3. Mortensen [75] showed how to obtain a tight $O(\lg n/\lg\lg n)$ upper bound in SODA'03.

Note that a (near-)constant running time for static 2D stabbing is natural in retrospect, since dominance reporting in 2D can be solved in constant time by the famous range minimum query (RMQ) results [24]. However, until recently, it seemed safe to conjecture that 3D range reporting would require $\widetilde{\Omega}(\lg n)$ query time for space $O(n \cdot \mathrm{polylog}\, n)$ — that is, that there is no special trick to play in 3D, and one just has to recurse to range trees to lift the 2D solution into 3D. This is even more reasonable a conjecture, given that the dynamic 2D problem has a lower bound of $\Omega(\lg n / \lg\lg n)$. Traditionally, dynamic d-dimensional problems tended to behave like static problems in $d + 1$ dimensions.

However, this conjecture was refuted by a recent result of Nekrich [79] from SoCG'07. It was shown that 3D range reporting can be done in doubly-logarithmic query time, specifically $t_q = O(\lg^2 \lg n)$. Without threatening the belief that *ultimately* the bounds should grow by $\Theta(\lg n / \lg\lg n)$ per dimension, this positive result raised the intriguing question whether further dimensions might also collapse to nearly constant time before this exponential growth begins.

Four dimensions. In Chapter 7, we show that the 4th dimension will not see a similar improvement:

Theorem 2.5. *A data structure for range reporting in 4 dimensions using space $n \cdot \sigma$ in the cell probe model with w-bit cells, requires query time $\Omega(\frac{\lg n}{\lg(w+\sigma)})$.*

For the main case $w = O(\lg n)$ and $S = O(n \cdot \mathrm{polylog}\, n)$, the query time must be $\Omega(\lg n / \lg\lg n)$. This is almost tight, since the result of Nekrich implies an upper bound of $O(\lg n \lg\lg n)$.

Reachability oracles in butterfly graphs. The natural question that our result stirs is: why would 4 dimensions be hard, if 3 dimensions turned out to be easy? The question has a simple, but fascinating answer: reachability oracles in butterfly graphs.

The following problem appears very hard: preprocess a sparse directed graph in less than n^2 space, such that reachability queries (can u be reached from v?) are answered efficiently. The problem seems to belong to folklore, and we are not aware of any nontrivial positive results. By contrast, for undirected graphs, many oracles are known.

We show the first lower bound supporting the apparent difficulty of the problem:

Theorem 2.6. *A reachability oracle using space $n\sigma$ in the cell probe model with w-bit cells, requires query time $\Omega(\frac{\lg n}{\lg(\sigma+w)})$.*

The bound holds even if the graph is a subgraph of a butterfly graph, and in fact it is tight for this special case. If constant time is desired, our bounds shows that the space needs to be $n^{1+\Omega(1)}$. This stands in contrast to undirected graphs, for which connectivity oracles are easy to implement with $O(n)$ space and $O(1)$ query time. Note however, that our lower bound is still very far from the conjectured hardness of the problem.

After showing this result, we give a reduction from reachability *on butterfly graphs* to static 4D range reporting, which proves the logarithmic complexity gap between 3 and 4 dimensions. These results are describe in Chapter 7.

Reporting in 2 dimensions. As mentioned already, in FOCS'00, Alstrup, Brodal, and Rauhe [7] showed how to solve static 2D range reporting in $O(\lg \lg n)$ time and almost linear space. This raises the question whether the query time can be reduced to constant. In the case of dominance queries, it can indeed by made $O(1)$ using range minimum queries. However, in Chapter 9, we show that:

Theorem 2.7. *A data structure for 2-dimensional range reporting in rank space* $[n]^2$*, using memory* $O(n \cdot \text{polylog}\, n)$ *in the cell-probe model with cells of* $O(\lg n)$ *cells, requires* $\Omega(\lg \lg n)$ *query time.*

To the best of our knowledge, this is the first separation between dominance queries and the general case. Our lower bound, in fact, hinges on some fairly deep developments in lower bounds for predecessor search: it relies on our lower bounds for the direct sum of predecessor problems.

2.3.3 Orthogonal Stabbing

A dual of range queries is *stabbing*: preprocess a set of n boxes of the form $[a_1, b_1] \times \cdots \times [a_d, b_d]$, such that we can quickly find the box(es) containing a query point.

Stabbing is a very important form of classification queries. For instance, network routers have rules (access control lists) applying to packets coming from some IP range, and heading to another IP range. A query is needed for every packet passing through the router, making this a critical problem. This application has motivated the following theoretically-minded papers [100, 44, 17, 41], as well as a significant body of practically-minded ones.

Another important application of stabbing is method dispatching, in experimental object oriented languages that (unlike, say, Java and C++) allow dynamic dispatching on more arguments than the class. This application has motivated the following theoretically-minded papers [78, 6, 45, 46], as well as a number of practically-minded ones.

Previous results. Static stabbing in one dimension can be solved easily by predecessor search (after locating the query among the interval end-points, you can determined the stabbed interval in constant time). For the sake of a lower bound, one can also obtain the inverse reduction: colored predecessor search reduces to stabbing (add an interval for each red point, and its next blue point; the query stabs an interval iff its predecessor is red).

Dynamic stabbing in one dimension is as hard as the marked ancestor problem of Alstrup, Husfeldt, and Rauhe [8] in FOCS'98. In this problem, we are to maintain a complete tree of degree b and depth d, in which vertices have a mark bit. The updates may mark or unmark a vertex. The query is given a leaf v, and must determine whether the path from the root to v contains any marked node.

Marked ancestor reduces to dynamic stabbing in 1D, by associating each vertex with an interval extending from the leftmost to the rightmost leaf in its subtree. Marking a node adds the interval to the set, and unmarking removes it. Then, an ancestor of a leaf is marked iff the leaf stabs an interval currently in the set.

Alstrup et al. [8] showed a lower bound trade-off for the marked ancestor problem, stating that $t_q = \Omega(\lg n / \lg(w + t_u))$. In particular, $\max\{t_q, t_u\} = \Omega(\lg n / \lg \lg n)$ for word size $w = O(\operatorname{polylog} n)$, and this bound carries over to dynamic range stabbing in 1D. A tight upper bound for range stabbing is obtained by Thorup [100] in STOC'03.

Our results. We show the first superconstant lower bounds for static stabbing in 2-dimensions, which is the application relevant to routers, and arguably the most frequently needed case of multimethod dispatching.

Theorem 2.8. *A data structure for orthogonal range stabbing in 2 dimensions using space $n \cdot \sigma$ in the cell probe model with w-bit cells, requires query time $\Omega(\frac{\lg n}{\lg(\sigma w)})$.*

Our bound is tight, matching the upper bound of [30]. Our bound holds by reduction from reachability oracles in butterfly graphs; see §2.3.2.

Reductions. It is easy to see that stabbing in d dimensions reduces to range reporting in $2d$ dimensions, since boxes can be expressed as $2d$-dimensional points. Thus, our lower bound for 2D stabbing immediately implies our lower bound for 4D reporting.

Existential range stabbing in 2D also reduces to (weighted) range counting in 2D by the following neat trick. We replace a rectangle $[a_1, b_1] \times [a_2, b_2]$ by 4 points: (a_1, b_1) and (a_2, b_2) with weight $+1$, and (a_1, b_2) and (a_2, b_1) with weight -1. To test whether (q_1, q_2) stabs a rectangle, query the sum in the range $[0, q_1] \times [0, q_2]$. If the query lies inside a rectangle, the lower-left corner contributes $+1$ to count. If the query point is outside, the corners cancel out.

Our lower bounds for stabbing can in fact guarantee that the query never stabs more than one rectangle. Then, in the above reduction, it suffices to count points mod 2. Thus, we obtain a lower bound even for the unweighted range counting problem.

2.4 Problems in High Dimensions

2.4.1 Partial Match

Formally, the problem asks to preprocess a data base of n strings in $\{0,1\}^d$, and support the following query: given a pattern in $\{0, 1, \star\}^d$, determine whether any string in the database matches this pattern (where $\star$ can match anything).

Since it is defined on the hypercube, the partial match problem is only interesting in very high dimensions. In fact, partial match is a stylized version of many practically important queries in high dimensions:

Range queries: Preprocess a set of n points in $\mathbb{R}^d$ to answer orthogonal range queries, such as reporting points in the range $[a_1, b_1] \times \cdots \times [a_d, b_d]$. Partial match can be seen as a dominance query on the hypercube: double the dimension and apply the following transformation: $0 \mapsto 01; 1 \mapsto 10; \star \mapsto 11$.

Near neighbor in ℓ_∞: A natural example of an orthogonal range in high dimensions is the hypercube, which has the nice feature that dimensions are treated symmetrically. This defines the near neighbor problem in ℓ_∞; see §2.4.3 for a discussion of this problem.

Intersection queries: Preprocess a family of sets $\mathcal{F} = \{S_1, \dots, S_d\}$, where $S_i \subseteq [n]$, to answer the query: given a subcollection of $\mathcal{F}$, is the intersection of those sets empty? This is one of the main queries in search engines: the set S_i is the list of documents containing word i, and the query asks for a document containing some set of words.

Searching with wild-cards: Preprocess a collection of strings over an alphabet Σ, to search for query strings in $\Sigma \cup \{\star\}$, where $\star$ is a wild card matching any letter. Partial match is the case $\Sigma = \{0, 1\}$.

It should be noted that these problems only degenerate into the partial match problem in high dimensions; in low dimensions, other considerations are more important. See, for example, the ample literature of range queries, discussed in §2.3.

The "low-dimensional" version of searching with wild-cards puts a bound k on the number of $\star$'s in the string. This problem is considerably less explored than range reporting. However, a recent break-through result of Cole, Gottlieb and Lewenstein [33] achieved results reminiscent of range reporting: they use space $O(n \lg^k n)$ and time $O(\lg^k n \cdot \lg\lg n)$, for any constant k.

Problem status. The first upper bounds for partial match was obtained by Rivest [93], who showed that the trivial 2^d space can be slightly improved when $d \le 2\lg n$. Charikar, Indyk, and Panigrahy [29] showed that query time $O(n/2^\tau)$ can be achieved with space $n \cdot 2^{O(d \lg^2 d / \sqrt{\tau / \lg n})}$. This is currently the best known theoretical result, though many heuristic solutions are used in practical applications.

It is generally conjectured that the problem suffers from the curse of dimensionality. A fairly plausible conjecture seems to be that there is no constant $\varepsilon > 0$, such that query time $O(n^{1-\varepsilon})$ can be supported with space $\mathrm{poly}(m) \cdot 2^{O(d^{1-\varepsilon})}$.

Partial match has been investigated in the asymmetric communication model (where Alice holds the query and Bob holds the database), in the hope of obtaining evidence for this curse of dimensionality. In STOC'95, Miltersen et al. [73] could show an $\Omega(\sqrt{\lg d})$ cell-probe lower bound via round elimination. In STOC'99, Borodin, Ostrovsky, and Rabani [25] used the richness method to show that either the querier sends $a = \Omega(\lg d \cdot \lg n)$ bits, or the database sends $b = \Omega(n^{1-\varepsilon})$ bits, for any $\varepsilon > 0$. In STOC'03, Jayram, Khot, Kumar, and Rabani [64] significantly strengthened the bound to prove that either Alice sends $a = \Omega(d/\lg n)$ bits, or Bob sends $b = \Omega(n^{1-\varepsilon})$ bits.

Our results. In our work [82], and Chapter 6, we give a reduction from lopsided set disjointness to partial match, showing that:

Theorem 2.9. *Let Alice hold a string in $\{0, 1, \star\}^d$, and Bob hold n strings in $\{0, 1\}^d$. In any bounded-error protocol answering the partial match query, either Alice sends $\Omega(d)$ bits or Bob sends $\Omega(n^{1-\varepsilon})$ bits, for any constant $\varepsilon > 0$.*

This improves the best previous bound for Alice's communication from $\Omega(d/\lg n)$ to the optimal $\Omega(d)$.

Our reduction is a simple exercise, and it seems surprising that the connection was not established before. Notably, Barkol and Rabani [20] gave a difficult lower bound for exact near neighbor in the Hamming cube, showing $a = \Omega(d)$ and $b = \Omega(n^{1/8-\varepsilon})$, though it was well known that partial match reduces to exact near neighbor. This suggests that partial match was viewed as a "nasty" problem. Our result greatly simplifies their proof, as well as improving Bob's communication to $\Omega(n^{1-\varepsilon})$.

Theorem 2.9 implies that a decision tree for the partial match problem with predicate size $O(n^{\varepsilon})$ must either have size $2^{\Omega(d)}$, or depth $\Omega(n^{1-2\varepsilon}/d)$. Thus, the course of dimensionality is true for decision trees in a very strong form.

By the standard relation between asymmetric communication and cell-probe complexity, Theorem 2.9 also implies that a data structure with query time t must use space $2^{\Omega(d/t)}$, assuming the word size is $O(n^{1-\varepsilon}/t)$. We normally assume the word size to be $O(d)$, or maybe $d^{O(1)}$, making the condition $w = O(n^{1-\varepsilon}/t)$ essentially trivial. As usual with such bounds, the cell-probe result is optimal for constant query time, but degrades quickly with t.

Our direct sum results for richness (see Chapter 6) slightly strengthen the implication of Theorem 2.9 to $t = \Omega(d/\lg \frac{S\cdot w}{n})$. This implies, for example, a time lower bound of $\Omega(d/\lg w)$ for space $S = O(n \cdot \mathrm{poly}(d \lg n))$, which is of course very far from the upper bounds.

2.4.2 Near Neighbor Search in ℓ_1, ℓ_2

Nearest neighbor search (NNS) is the problem of preprocessing a collection of n points, such that we can quickly find the database point closest to a given query point. This is a key algorithmic problem arising in several areas such as data compression, databases and data mining, information retrieval, image and video databases, machine learning, pattern recognition, statistics and data analysis.

The most natural examples of spaces in which NNS can be defined are the ℓ_p^d norms, denoting the space $\Re^d$ endowed with the distance $\|x-y\|_p = \left(\sum_{i=1}^d |x_i - y_i|^p\right)^{1/p}$. Significant attention has been devoted to NNS in the Euclidean norm ℓ_2 and the Manhattan norm ℓ_1. We refer the reader to surveys in [14, 96, 94]. Another very important, but less understood space is ℓ_∞, which we discuss in §2.4.3.

Exact NNS is conjectured to suffer from a "curse of dimensionality," which makes the bounds grow exponentially with the dimension. For example, two natural solutions for NNS are:

- answer queries by a linear scan, using linear space and linear query time.
- construct the d-dimensional Voronoi diagram, which uses space $n^{\Theta(d)}$, but allows query time $\mathrm{poly}(d \lg n)$.

The conjectured curse of dimensionality states that the transition between these two types of bounds is sharp: it is impossible to achieve $O(n^{1-\varepsilon})$ query time with space $n^{o(d)}$.

Though many heuristic solutions have been designed for practical instances, from a theoretical perspective, the curse of dimensionality has only been overcome by allowing approximation. In the c-approximate nearest neighbor problem, the goal is to return a point which is at most a factor c farther than the nearest neighbor.

A very clean setup for NNS is the Hamming cube $\{0,1\}^d$. In this case, $\|p-q\|_H = \|p-q\|_1 = \big(\|p-q\|_2\big)^2$, so a c approximation for ℓ_2 is equivalent to a c^2 approximation for ℓ_1. Essentially, it suffices to consider the problem in this particular case, since there exist efficient *embeddings* of other spaces into the Hamming cube.

Hardness of Exact NNS

Perhaps the strongest evidence for the curse of dimensionality of NNS is a simple reduction from partial match (see §2.4.1). If the query contains k wildcards, and we map translate these wildcards into a coordinate value of $\frac{1}{2}$, the nearest distance between the query and an integral point is $\frac{k}{2}$ is ℓ_1 and $\sqrt{k/4}$ in ℓ_2. If the query is matched by a database string, the string is precisely at this minimum distance. Otherwise, the nearest neighbor is farther away.

Before our work, communication lower bounds for partial match were not optimal: [64] only bounded the communication of the querier by $\Omega(d/\lg n)$. In STOC'00, Barkol and Rabani [20] circumvented the partial match problem, and considered exact NNS directly. They considered the problem in the Hamming cube, and showed that either Alice must communicate $a = \Omega(d)$ bits, or Bob must send $b = \Omega(n^{1/8-\delta})$ bits, for any $\delta > 0$. Our result for partial match (Theorem 2.9) supersedes this result: Bob's communication is improved to $\Omega(n^{1-\delta})$, and our proof is considerably simpler.

Upper Bounds for Approximate NNS

Solutions to the approximate nearest neighbor problem often go via the c-approximate *near* neighbor problem. In this problem, a data structure must be constructed for a given set of points S, approximation c, and radius r. Given a point q, the near-neighbor query returns:

- a point $p \in S$ at distance $\|p-q\| \le c \cdot r$; or
- NO, if all points $p \in S$ are at distance $\|p-q\| > r$.

Note that there is an overlap between the two cases. In the decision version of this problem, the query returns:

- YES, if there is a point $p \in S$ at distance $\|q-p\| \le r$.
- NO, if all points $p \in S$ are at distance $\|q-p\| > c \cdot r$.
- an unspecified value otherwise.

One can use approximate near neighbor to solve approximate nearest neighbor by constructing near-neighbor structures for all $r = c^i$, and doing a binary search for the correct r at query time.

Solutions for approximate NNS attack two ranges of parameters: approximation $1+\varepsilon$, and "large" approximation c (for instance, $c=2$).

In the $1+\varepsilon$ regime, the landmark papers of Kushilevitz, Ostrovsky, and Rabani [67] and Indyk and Motwani [63] solved the near neighbor problem with space $n^{O(1/\varepsilon^2)}$ and very efficient query time. These results demonstrate that exponential dependence on the dimension can be overcome for any constant approximation, though, of course, the space bound is only interesting from a theoretical perspective. In some sense, the curse of dimensionality is replaced by a "curse of approximation."

The main idea behind these data structures is the *dimensionality reduction* method. This can be exemplified by the Johnson-Lindenstrauss lemma, which states for any fixed points p, q in arbitrary dimension, a projection π on a "random" hyperplane of dimension $O(\frac{1}{\varepsilon^2}\lg\frac{1}{\delta})$ satisfies (ignoring normalization):

$$\Pr\left[(1-\varepsilon)\|p-q\|_2 \leq \|\pi(p)-\pi(q)\|_2 \leq (1+\varepsilon)\|p-q\|_2\right] \;\leq\; \delta$$

Proofs are given for various notions of a "random hyperplane".

Thus, the database can be projected to a space of dimension $d' = O(\frac{1}{\varepsilon^2}\lg n)$, and the distance from the query to the nearest neighbor does not change by more than $1+\varepsilon$ with high probability. After reducing the problem to low dimensions, one can essentially use complete tabulation to achieve space exponential in the dimension. The idea is to store "all" points in $\Re^{d'}$ within distance 1 from some $p \in S$. To do this, we impose a cubic grid on $\Re^{d'}$, with each cell having diameter ϵ. It can be shown [63] that each unit ball in $\Re^{d'}$ touches at most $(1/\epsilon)^{O(d')}$ grid cells. Therefore, we can afford to store all such cells within the given space bound.

To answer a query, we simply check if a grid cell containing the query point has been stored. The query algorithm is extremely efficient: it only needs to apply the projection to the query, and then look up the resulting point. Thus, the cell-probe complexity is constant.

The paper of Indyk and Motwani [63] also initiated the study of c-approximate near neighbor, for "large" c. They introduced *locality sensitive hashing* (LSH), a technique used in all subsequent work in this regime. By constructing a family of LSH functions with *parameter* ρ, one can obtain a data structure with space $\widetilde{O}(n^{1+\rho})$ and query time $\widetilde{O}(n^{\rho})$. They constructed such a family with $\rho = 1/c$ for the ℓ_1 and ℓ_2 metric, obtaining a near-neighbor data structures with space $\widetilde{O}(n^{1+1/c})$ and query time $\widetilde{O}(n^{1/c})$.

Panigrahy [80] showed how to use LSH functions to obtain data structures with very efficient query time *or* near-linear space. In particular, he obtained query time $\mathrm{poly}(d\lg n)$ with space $n^{1+O(\rho)} = n^{1+O(1/c)}$, as well as space $\widetilde{O}(n)$ with query time $n^{O(\rho)} = n^{O(1/c)}$.

Motwani, Naor, and Panigrahy [77] showed a geometric lower bound on the parameter of LSH functions: in ℓ_1, $\rho = \Omega(1/c)$, while in ℓ_2, $\rho = \Omega(1/c^2)$. Of course, this does not rule out data structures using other techniques.

Datar, Immorlica, Indyk, and Mirrokni [34] improved the LSH upper bound for ℓ_2 to some $\rho < \frac{1}{c}$. Finally, Andoni and Indyk [14] showed $\rho = \frac{1}{c^2} + o(1)$, asymptotically matching the lower bound.

Unlike the regime of $1+\varepsilon$ approximations, work in the regime of high approximation has produced practical data structures (see, for example, the popular E2LSH package of Andoni and Indyk). Usually, the solution for approximate near neighbor is used as a heuristic filter: the query scans all points at distance at most $c \cdot r$ from the query (hoping they are few), and finds the true nearest neighbor.

Lower Bounds for Approximate NNS

Prior to our work, lower bounds for approximate NNS have focused on the difference between the near-neighbor and the nearest-neighbor problems. If we want constant $1+\varepsilon$ approximation and we accept polynomial space as "efficient," then [67, 63] solve the near neighbor problem optimally, with constant query time. However, in the Hamming cube, the nearest neighbor problem will have an $O(\lg\lg d)$ query time: the query must binary search for $i \le \log_{1+\varepsilon} d$, such that the distance to the nearest neighbor is between $(1+\varepsilon)^i$ and $(1+\varepsilon)^{i+1}$. In STOC'99, Chakrabarti, Chazelle, Gum, and Lvov [26] used round elimination to show the first lower bound for approximate *nearest* neighbor. In FOCS'04, Chakrabarti and Regev [27] slightly improved the upper bound to $O(\lg\lg d/\lg\lg\lg d)$, and showed a matching lower bound.

In our joint work with Alex Andoni and Piotr Indyk [16], we turned to the main cause of impracticality for $1+\varepsilon$ approximation: the prohibitive space usage. We considered approximate near neighbor in the usual asymmetric communication setting, and showed:

Theorem 2.10. *Let Alice receive a query $q \in \{0,1\}^d$ and Bob receive a database S of n points from $\{0,1\}^d$, where $d = (\frac{1}{\varepsilon}\lg n)^{O(1)}$. In any randomized protocol solving the $(1+\varepsilon)$-approximate near neighbor problem, either Alice sends $\Omega(\frac{1}{\varepsilon^2}\lg n)$ bits, or Bob sends $\Omega(n^{1-\delta})$ bits, for any $\delta > 0$.*

This theorem is shown by reduction from lopsided set disjointness; a proof can be found in Chapter 6. As usual, this lower bound implies that for data structures with constant cell-probe complexity, and for decision trees, the space must be $n^{\Omega(1/\varepsilon^2)}$. Thus, dimensionality reduction method gives an optimal (but prohibitive) space bound.

Very recently, Talwar, Panigrahy, and Wieder [98] considered the space consumption in the regime of high approximation. They showed that a data structure with constant query time requires space $n^{1+\Omega(1/c)}$ in the ℓ_1 case, and $n^{1+\Omega(1/c^2)}$ in the ℓ_2 case.

Finally, Liu [69] considered the case of approximate NNS when randomization is disallowed. For any constant approximation, he showed that in a deterministic protocol, either the querier sends $a = \Omega(d)$ bits, or the database sends $b = \Omega(nd)$ bits. This suggests that, in absence of randomization, approximate NNS might also suffer from a curse of dimensionality.

2.4.3 Near Neighbor Search in ℓ_∞

The ℓ_∞ space is dictated by a very natural metric that measures the maximum distance over coordinates: $\|x-y\|_\infty = \max_{i=1}^d |x_i - y_i|$. Though recent years have seen a significant increase in our understanding of high-dimensional nearest neighbor search in ℓ_1 and ℓ_2 spaces, ℓ_∞ remains the odd-man out, with a much less understood, and intriguingly different structure.

In fact, there is precisely one data structure for with provable theoretical guarantees for NNS in ℓ_∞. In FOCS'98, Indyk [61] proved the following unorthodox result: there is a data structure (in fact, a decision tree) of size $O(n^\rho)$, for any $\rho > 1$, which achieves approximation $4\lceil \log_\rho \log 4d \rceil + 1$ for NNS in the d-dimensional ℓ_∞ metric. The query time is $O(d \cdot \operatorname{polylog} n)$. For 3-approximation, Indyk can achieve space $n^{\log d + 1}$. Note that in the important regime of polynomial space, Indyk's algorithm achieves an uncommon approximation factor of $O(\log \log d)$.

Partial match can be reduced to 3-approximate near neighbor in ℓ_∞, by applying the following transformation to each coordinate of the query: $0 \mapsto -\frac{1}{2}$; $\star \mapsto \frac{1}{2}$; $1 \mapsto \frac{3}{2}$. Thus, it is likely that efficient solutions are only possible for approximation $c \geq 3$.

Applications of ℓ_∞

For some applications, especially when coordinates are rather heterogeneous, ℓ_∞ may be a natural choice for a similarity metric. If the features represented by coordinates are hard to relate, it is hard to add up their differences numerically, in the sense of ℓ_1 or ℓ_2 (the "comparing apples to oranges" phenomenon). One popular proposal is to convert each coordinate to rank space, and use the maximum rank difference as an indication of similarity. See for example [42].

However, the most compelling reasons for studying ℓ_∞ are extroverted, stemming from its importance in a theoretical understanding of other problems. For example, many NNS problems under various metrics have been reduced to NNS under ℓ_∞ via *embeddings* (maps that preserve distances up to some distortion). A well-known result of Matoušek states that *any* metric on n points can be embedded into ℓ_∞ with dimension $d = O(cn^{1/c} \log n)$ and distortion $2c - 1$. In particular, if we allow dimension n, the embedding can be isometric (no distortion). Of course, this general guarantee on the dimension is too high for many applications, but it suggests that ℓ_∞ is a very good target space for trying to embed some particular metric more efficiently.

Early embeddings into ℓ_∞ with interesting dimension included various results for Hausdorff metrics [43], embedding tree metrics into dimension $O(\log n)$ [68], and planar graphs metrics into dimension $O(\log n)$ [66] (improving over [91]).

More recently, embeddings have been found into generalizations of ℓ_∞, namely product spaces. For a metric $\mathcal{M}$, the *max-product* over k copies of $\mathcal{M}$ is the space $\mathcal{M}^k$ with the distance function $d_{\infty,\mathcal{M}}(x, y) = \max_{i=1}^{k} d_{\mathcal{M}}(x_i, y_i)$, where $x, y \in \mathcal{M}^k$. Indyk [62] has extended his original NNS algorithm from ℓ_∞ to max-product spaces, thus an algorithm for the Frechet metric.

In current research, it was shown by Andoni, Indyk, and Krauthgamer [15] that algorithms for max-product spaces yield (via a detour through *sum-product* spaces) interesting upper bounds for the Ulam metric and the Earth-Mover Distance. By embedding these metrics into (iterated) sum-products, one can achieve approximations that are provably smaller than the best possible embeddings into ℓ_1 or ℓ_2.

Thus, the bottleneck in some of the best current algorithms for Frechet, Ulam and EMD metrics is the ℓ_∞ metric. In particular, Indyk's unusual log-logarithmic approximation for polynomial space carries over to these metrics.

Our Results

The unusual structure of ℓ_∞ NNS (as evidenced by an uncommon approximation result) and the current developments leading to interesting applications plead for a better understanding of the problem. The reduction from partial match to NNS with approximation better than 3 does not explain the most interesting feature of the problem, namely the space/approximation trade-off. It also leaves open the most interesting possibility: a constant factor approximation with polynomial space. (It appears, in fact, that researchers were optimistic about such an upper bound being achievable [60].)

In our joint work with Alex Andoni and Dorian Croitoru [13], presented in Chapter 8, we show the following lower bound for the asymmetric communication complexity of c-approximate NNS in ℓ_∞:

Theorem 2.11. *Assume Alice holds a point q, and Bob holds a database D of n points in d dimensions. Fix $\delta > 0$, and assume d satisfies $\Omega(\lg^{1+\delta} n) \leq d \leq o(n)$. For any $\rho > 10$, define $c = \Theta(O(\log_\rho \log_2 d) > 3$.*

In any deterministic protocol solving the c-approximate near neighbor problem in ℓ_∞, either Alice sends $\Omega(\rho \lg n)$ bits, or Bob sends $\Omega(n^{1-\delta})$ bits.

Our lower bound stems from a new information-theoretic understanding of Indyk's algorithm, also presented in Chapter 8. We feel that this conceptual change allows for a cleared explanation of the algorithm, which may be of independent interest.

Note that our lower bound is tight in the communication model, by Indyk's algorithm. As usual, the communication bound implies that for data structures with constant cell-probe complexity, as well as for decision trees of depth $O(n^{1-\delta})$, the space must be $n^{\Omega(\rho)}$. Since Indyk's result is a deterministic decision tree with depth $O(d \cdot \operatorname{polylog} n)$, we obtain an optimal trade-off between space and approximation, at least in the decision tree model.

This suggests that Indyk's unusual space/approximation trade-off, in particular the $\Theta(\lg \lg d)$ approximation with polynomial space, is in fact inherent to NNS in ℓ_∞.

Chapter 3

Dynamic $\Omega(\lg n)$ Bounds

In this chapter, we prove our first lower bounds: we show that the partial sums and dynamic connectivity problems require $\Omega(\lg n)$ time per operation. We introduce the proof technique using the partial sums problem; dynamic connectivity requires two additional tricks, described in §3.6 and §3.7. The proofs are surprisingly simple and clean, in contrast to the fact that proving any $\Omega(\lg n)$ bound was a well-known open problem for 15 years before our paper [86].

3.1 Partial Sums: The Hard Instance

It will pay to consider the partial sums problem in a more abstract setting, namely over an arbitrary group $\mathbb{G}$. Remember that the problem asked to maintain an array $A[1 \,..\, n]$, initialized to zeroes (the group identity), under the following operations:

UPDATE(k, Δ)**:** modify $A[k] \leftarrow \Delta$, where $\Delta \in \mathbb{G}$.

SUM(k)**:** returns the partial sum $\sum_{i=1}^{k} A[i]$.

Our proof works for any choice of $\mathbb{G}$. In the cell-probe model with w-bit cells, the most natural choice of $\mathbb{G}$ is $\mathbb{Z}/2^w\mathbb{Z}$, i.e. integer arithmetic modulo 2^w. In this case, the argument Δ of an update is a machine word. Letting $\delta = \lg|\mathbb{G}|$, our proof will show that any data structure requires an average running time of $\Omega(\frac{\delta}{w} \cdot n \lg n)$ to execute a sequence of n updates and n queries chosen from a particular distribution. If $\delta = w$, we obtain an amortized $\Omega(\lg n)$ bound per operation.

The hard instance is described by a permutation π of size n, and a sequence $\langle \Delta_1, \ldots, \Delta_n \rangle \in \mathbb{G}^n$. Each Δ_i is chosen independently and uniformly at random from $\mathbb{G}$; we defer the choice of π until later. For t from 1 to n, the hard instance issues two operations: the query SUM$(\pi(t))$, followed by UPDATE$(\pi(t), \Delta_t)$. We call t the "time," saying, for instance, that SUM$(\pi(t))$ occurs at time t.

A very useful visualization of an instance is as a two-dimensional chart, with time on one axis, and the index in A on the other axis. The answer to a query SUM$(\pi(t))$ is the sum of the update points in the rectangle $[0, t] \times [0, \pi(t)]$; these are the updates which have already occurred, and affect indices relevant to the partial sum. See Figure 3-1 (a).

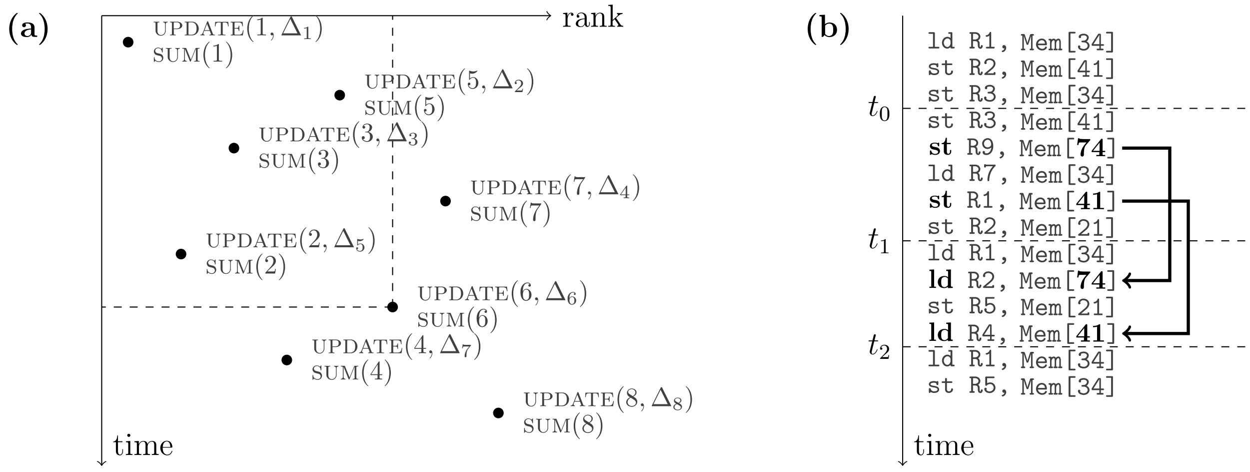


Figure 3-1: (a) An instance of the partial sums problem. The query $\text{SUM}(6)$ occurring at time 6 has the answer $\Delta_1 + \Delta_2 + \Delta_3 + \Delta_5$. (b) The execution of a hypothetical cell-probe algorithm. $IT(t_0, t_1, t_2)$ consists of cells 41 and 74.

3.2 Information Transfer

Let $t_0 < t_1 < t_2$, where t_0 and t_2 are valid time values, and t_1 is non-integral to avoid ties. These time stamps define two adjacent intervals of operations: the time intervals $[t_0, t_1]$ and $[t_1, t_2]$; we will be preoccupied by the interaction between these time intervals. Since the algorithm cannot maintain state between operations, such interaction can only be caused by the algorithm writing a cell during the first interval and reading it during the second.

Definition 3.1. *The* information transfer $IT(t_0, t_1, t_2)$ *is the set of memory locations which:*

- *were read at a time* $t_r \in [t_1, t_2]$.
- *were written at a time* $t_w \in [t_0, t_1]$*, and not overwritten during* $[t_w + 1, t_r]$.

The definition is illustrated in Figure 3-1 (b). Observe that the information transfer is a function of the algorithm, the permutation π, and the sequence Δ.

For now, let us concentrate on bounding $|IT(t_0, t_1, t_2)|$, ignoring the question of how this might be useful. Intuitively, any dependence of the queries from $[t_1, t_2]$ on updates from the interval $[t_0, t_1]$ must come from the information in the cells $IT(t_0, t_1, t_2)$. Indeed, $IT(t_0, t_1, t_2)$ captures the only possible information flow between the intervals: an update happening during $[t_0, t_1]$ cannot be reflected in a cell written before time t_0.

Let us formalize this intuition. We break the random sequence $\langle \Delta_1, \ldots, \Delta_n \rangle$ into the sequence $\Delta_{[t_0, t_1]} = \langle \Delta_{t_0}, \ldots, \Delta_{\lfloor t_1 \rfloor} \rangle$, and $\Delta^\star$ containing all other values. The values in $\Delta^\star$ are uninteresting to our analysis, so fix them to some arbitrary $\boldsymbol{\Delta}^\star$. Let A_t be the answer of the query $\text{SUM}(\pi(t))$ at time t. We write $A_{[t_1, t_2]} = \langle A_{\lceil t_1 \rceil}, \ldots, A_{t_2} \rangle$ for the answers to the queries in the second interval.

In information theoretic terms, the observation that all dependence of the interval $[t_1, t_2]$ on the interval $[t_0, t_1]$ is captured by the information transfer, can be reformulated as saying that the *entropy* of the observable outputs of interval $[t_1, t_2]$ (i.e., the query results $A_{[t_1,t_2]}$) is bounded by the information transfer:

Lemma 3.2. $H\big(A_{[t_1,t_2]} \mid \Delta^\star = \mathbf{\Delta}^\star\big) \;\le\; w \;+\; 2w \cdot \mathbf{E}\big[|IT(t_0,t_1,t_2)| \mid \Delta^\star = \mathbf{\Delta}^\star\big].$

Proof. The bound follows by proposing an *encoding* for $A_{[t_1,t_2]}$, since the entropy is upper bounded by the average length of any encoding. Our encoding is essentially the information transfer; formally, it stores:

- first, the cardinality $|IT(t_0,t_1,t_2)|$, in order to make the encoding prefix free.
- the *address* of each cell; an address is at most w bits in our model.
- the *contents* of each cell at time t_1, which takes w bits per cell.

The average length of the encoding is $w \;+\; 2w \cdot \mathbf{E}\big[|IT(t_0,t_1,t_2)| \mid \Delta^\star = \mathbf{\Delta}^\star\big]$ bits, as needed. To finish the proof, we must show that the information transfer actually encodes $A_{[t_1,t_2]}$; that is, we must give a *decoding algorithm* that recovers $A_{[t_1,t_2]}$ from $IT(t_0,t_1,t_2)$.

Our decoding algorithm begins by simulating the data structure during the time period $[1, t_0 - 1]$; this is possible because $\Delta^\star$ is fixed, so all operations before time t_0 are known. It then *skips* the time period $[t_0, t_1]$, and simulates the data structure again during the time period $[t_1, t_2]$. Of course, simulating the time period $[t_1, t_2]$ recovers the answers $A_{[t_1,t_2]}$, which is what we wanted to do.

To see why it is possible to simulate $[t_1, t_2]$, consider a read instruction executed by a data structure operation during $[t_1, t_2]$. Depending on the time t_w when the cell was last written, we have the following cases:

$t_w > t_1$: We can recognize this case by maintaining a list of memory locations written during the simulation; the data is immediately available.

$t_0 < t_w < t_1$: We can recognize this case by examining the set of addresses in the encoding; the cell contents can be read from the encoding.

$t_w < t_0$: This is the default case, if the cell doesn't satisfy the previous conditions. The contents of the cell is determined from the state of the memory upon finishing the first simulation up to time $t_0 - 1$. □

3.3 Interleaves

In the previous section, we showed an upper bound on the dependence of $[t_1, t_2]$ on $[t_0, t_1]$; we now aim to give a lower bound. Refer to the example in Figure 3-2 (a). The information that the queries in $[t_1, t_2]$ *need to know* about the updates in $[t_0, t_1]$ is the sequence $\big\langle \Delta_6,\ \Delta_6 + \Delta_3 + \Delta_4,\ \Delta_6 + \Delta_3 + \Delta_4 + \Delta_5 \big\rangle$. Equivalently, the queries need to know $\big\langle \Delta_6,\ \Delta_3 + \Delta_4,\ \Delta_5 \big\rangle$, which are three independent random variables, uniformly distributed in the group $\mathbb{G}$.

This required information comes from *interleaves* between the update indices in $[t_0, t_1]$, on the one hand, and the query indices in $[t_1, t_2]$, on the other. See Figure 3-2 (b).

Definition 3.3. *If one sorts the set $\{\pi(t_0), \ldots, \pi(t_2)\}$, the interleave number $IL(t_0, t_1, t_2)$ is defined as the number of transitions between a value $\pi(i)$ with $i < t_1$, and a consecutive value $\pi(j)$ with $j > t_1$.*

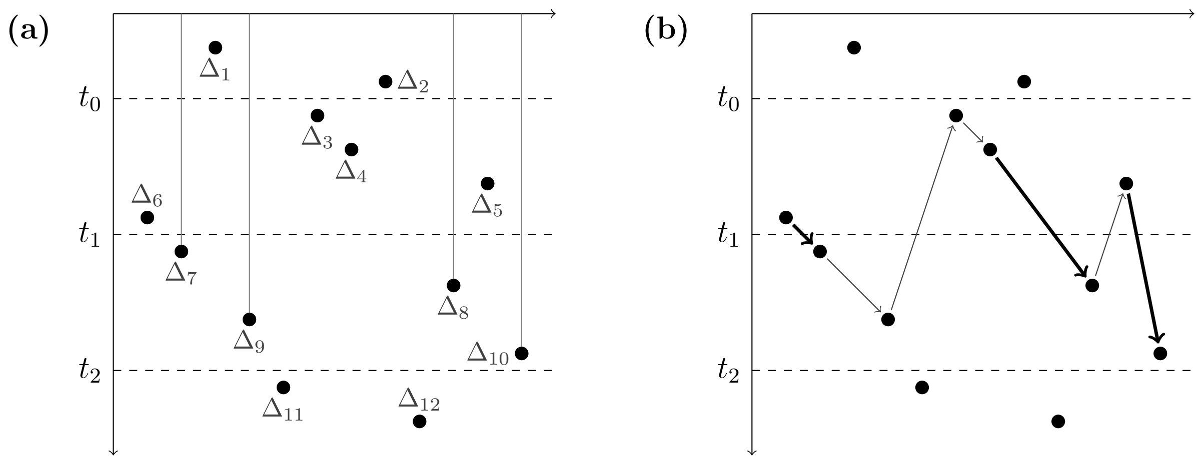


Figure 3-2: (a) The vertical lines describe the information that the queries in $[t_1, t_2]$ from the updates in $[t_0, t_1]$. (b) The interleave number $IL(t_0, t_1, t_2)$ is the number of down arrows crossing t_1, where arrows indicate left-to-right order.

The interleave number is only a function of π. Figure 3-2 suggests that interleaves between two intervals cause a large dependence of the queries $A_{[t_1,t_2]}$ on the updates $\Delta_{[t_1,t_2]}$, i.e. $A_{[t_1,t_2]}$ has large conditional entropy, even if all updates outside $\Delta_{[t_1,t_2]}$ are fixed:

Lemma 3.4. $H\big(A_{[t_1,t_2]} \mid \Delta^\star = \mathbf{\Delta}^\star\big) \;=\; \delta \cdot IL(t_0, t_1, t_2)$.

Proof. Each answer in $A_{[t_1,t_2]}$ is a sum of some random variables from $\Delta_{[t_0,t_1]}$, plus a constant that depends on the fixed $\Delta^\star$. Consider the indices $L = \{\pi(t_0), \dots, \pi(\lfloor t_1 \rfloor)\}$ from the first interval, and $R = \{\pi(\lceil t_1 \rceil), \dots, \pi(t_2)\}$ from the second interval. Relabel the indices of R as $r_1 < r_2 < \cdots$ and consider these r_i's in order:

- If $L \cap [r_{i-1}, r_i] = \emptyset$, the answer to $\textsc{sum}(r_i)$ is the same as for $\textsc{sum}(r_{i-1})$, except for a different constant term. The answer to $\textsc{sum}(r_i)$ contributes nothing to the entropy.
- Otherwise, the answer to $\textsc{sum}(r_i)$ is a random variable independent of all previous answers, due to the addition of random Δ's to indices $L \cap [r_{i-1}, r_i]$. This random variable is uniformly distributed in $\mathbb{G}$, so it contributes δ bits of entropy. □

Comparing Lemmas 3.4 and 3.2, we see that $\mathbf{E}\big[|IT(t_0, t_1, t_2)| \;\big|\; \Delta^\star = \mathbf{\Delta}^\star\big] \;\geq\; \frac{\delta}{2w} \cdot IL(t_0, t_1, t_2) - 1$ for any fixed $\mathbf{\Delta}^\star$. By taking expectation over $\Delta^\star$, we have:

Corollary 3.5. *For any fixed π, $t_0 < t_1 < t_2$, and any algorithm solving the partial sums problem, we have* $\mathbf{E}_\Delta\big[|IT(t_0, t_1, t_2)|\big] \;\geq\; \frac{\delta}{2w} \cdot IL(t_0, t_1, t_2) - 1$.

3.4 A Tree For The Lower Bound

The final step of the algorithm is to consider the information transfer between many pairs of intervals, and piece together the lower bounds from Corollary 3.5 into one lower bound

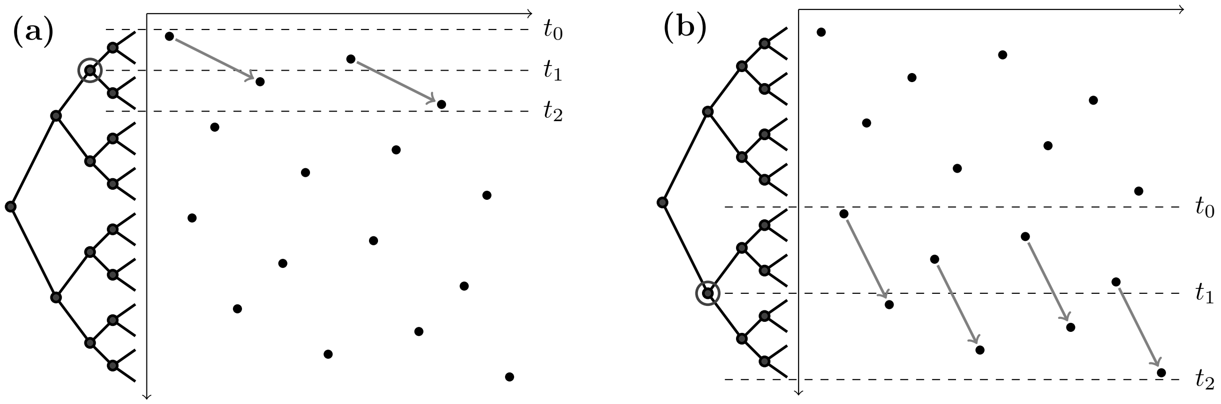


Figure 3-3: The bit-reversal permutation of size $n = 16$, and the lower-bound tree over the time axis. Each node has a maximal possible interleave: e.g. 2 in (a), and 4 in (b).

for the total running time of the data structure. The main trick for putting together these lower bounds is to consider a lower-bound tree $\mathcal{T}$: an arbitrary binary tree with n leaves, where each leaf denotes a time unit (a query and update pair). In other words, $\mathcal{T}$ is built "over the time axis," as in Figure 3-3.

For each internal node v of $\mathcal{T}$, we consider the time interval $[t_0, t_1]$ spanning the left subtree, and the interval $[t_1, t_2]$ spanning the right subtree. We then define:

- the information transfer *through* the node: $IT(v) = |IT(t_0, t_1, t_2)|$. Essentially, $IT(v)$ counts the cells written in the left subtree of v, and read in the right subtree.
- the interleave at the node: $IL(v) = IL(t_0, t_1, t_2)$.

Theorem 3.6. *For any algorithm and fixed π, the expected running time of the algorithm over a random sequence Δ is at least $\frac{\delta}{2w} \sum_{v \in \mathcal{T}} IL(v) \; - \; n$.*

Proof. First, observe that on any problem instance (any fixed Δ), the number of read instructions executed by the algorithm is at least $\sum_{v \in \mathcal{T}} IT(v)$. Indeed, for each read instruction, let t_r be the time it is executed, and $t_w \le t_r$ be the time when the cell was last written. If $t_r = t_w$, we can ignore this trivial read. Otherwise, this read instruction appears in the information transfer through exactly one node: the lowest common ancestor of t_w and t_r. Thus, $\sum_v IT(v)$ never double-counts a read instruction.

Now we apply Corollary 3.5 to each node, concluding that for each v, $\mathbf{E}_\Delta[IT(v)] \ge \frac{\delta}{2w} \cdot IL(v) - 1$. Thus, the total expected running time is at least $\frac{\delta}{2w} \sum_v IL(v) - (n-1)$. It is important to note that each lower bound for $|IT(v)|$ applies to the expected value under the same distribution (a uniformly random sequence Δ). Thus we may sum up these lower bounds to get a lower bound on the entire running time, using linearity of expectation. $\square$

To complete our lower bound, it remains to design an access sequence π that has high total interleave, $\sum_{v \in \mathcal{T}} IL(v) = \Omega(n \lg n)$, for some lower-bound tree $\mathcal{T}$. From now on, assume n is a power of 2, and let $\mathcal{T}$ be a perfect binary tree.

Claim 3.7. *If π is a uniformly random permutation, $\mathbf{E}_\pi\big[\sum_{v\in\mathcal{T}} IL(v)\big] = \Omega(n \lg n)$.*

Proof. Consider a node v with $2k$ leaves in its subtree, and let S be the set of indices touched in v's subtree, i.e. $S = \{\pi(t_0), \dots, \pi(t_2)\}$. The interleave at v is the number of down arrows crossing from the left subtree to the right subtree, when S is sorted; see Figure 3-2 (b) and Figure 3-3. For two indices $j_1 < j_2$ that are consecutive in S, the probability that j_1 is touched in the left subtree, and j_2 is touched in the right subtree will be $\frac{k}{2k} \cdot \frac{k}{2k-1} > \frac{1}{4}$. By linearity of expectation over the $2k-1$ arrows, $\mathbf{E}_\pi[IL(v)] = (2k-1) \cdot \frac{k}{2k} \cdot \frac{k}{2k-1} = \frac{k}{2}$. Summing up over all internal nodes v gives $\mathbf{E}_\pi\big[\sum_v IL(v)\big] = \frac{1}{4} n \log_2 n$. □

Thus, any algorithm requires $\Omega(\frac{\delta}{w} \cdot n \lg n)$ cell probes in expectation on problem instances given by random Δ and random π. This shows our $\Omega(\lg n)$ amortized lower bound for $\delta = w$.

3.5 The Bit-Reversal Permutation

An interesting alternative to choosing π randomly, is to design a worst-case π that *maximizes* the total interleave $\sum_{v\in\mathcal{T}} IL(v)$. We construct π recursively. Assume π' is the worst-case permutation of size n. Then, we *shuffle* two copies of π' to get a permutation π of size $2n$. Formally:

$$\pi = \big\langle 2\pi'(1) - 1, \ \cdots, \ 2\pi'(n) - 1, \quad 2\pi'(1), \ \cdots, \ 2\pi'(n) \big\rangle$$

The two halves interleave perfectly, giving an interleave at the root equal to n. The order in each half of π is the same as π'. Thus, by the recursive construction, each node with $2k$ leaves in its subtree has a perfect interleave of k. Summing over all internal nodes, $\sum_v IL(v) = \frac{1}{2} n \log_2 n$. Refer to Figure 3-3 for an example with $n = 16$, and an illustration of the perfect interleave at each node.

The permutation that we have just constructed is the rather famous *bit-reversal permutation.* Subtracting 1 from every index, we get a permutation of the elements $\{0, \dots, n-1\}$ which is easy to describe: the value $\pi(i)$ is the number formed by reversing the $\lg n$ bits of i. To see this connection, consider the recursive definition of π: the first half of the values (most significant bit of i is zero) are even (least significant bit of $\pi(i)$ is zero); the second half (most significant bit of i is one) are odd (least significant bit of $\pi(i)$ is one). Recursively, all bits of i except the most significant one appear in π' in reverse order.

Duality of upper and lower bounds. An important theme of this thesis is the idea that a good lower bound should be a natural dual of the best upper bound. The standard upper bound for the partial-sums problem is a balanced binary search tree with the array $A[1 \mathinner{..} n]$ in its leaves. Every internal node is augmented to store the sum of all leaves in its subtree. An update recomputes all values on the leaf-to-root path, while a query sums left children hanging from the root-to-leaf path.

Thinking from a lower bound perspective, we can ask when an update (a cell write) to some node v is "actually useful." If v is a left child, the value it stores is used for any query that lies in the subtree of its right sibling. A write to v is useful if a query to the sibling's subtree occurs before another update recomputes v's value. In other words, a write

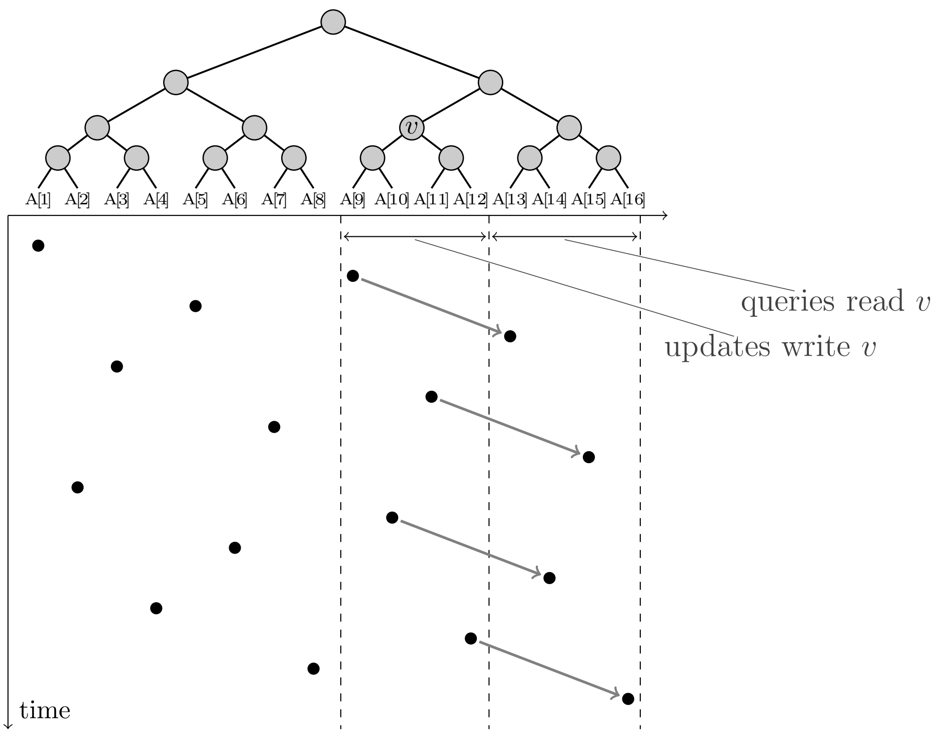


Figure 3-4: An augmented binary search tree solving a partial-sums instance. The writes to some node v are "useful" when interleaves occur in the time-sorted order.

instruction is useful whenever there is an interleave between v's left and right subtrees, sorting operations *by time*. See Figure 3-4.

Intuitively, the amount of "useful work" that the algorithm does is the sum of the interleaves at every node of the binary search tree. To maximize this sum, we can use a bit-reversal permutation again. Note the bit-reversal permutation is equal to its own inverse (reversing the bits twice gives the identity); in other words, the permutation is invariant under 90-degree rotations. Thus, the lower-bound tree sitting on the time axis counts exactly the same interleaves that generate work in the upper bound.

3.6 Dynamic Connectivity: The Hard Instance

We now switch gears to dynamic connectivity. Remember that this problem asks to maintain an undirected graph with a fixed set V of vertices, subject to the following operations:

INSERT(u, v): insert an edge (u, v) into the graph.

DELETE(u, v): delete the edge (u, v) from the graph.

CONNECTED(u, v): test whether u and v lie in the same connected component.

We aim to show an amortized lower bound of $\Omega(\lg |V|)$ per operation.

It turns out that the problem can be dressed up as an instance of the partial sums problem. Let $n = \sqrt{V} - 1$, and consider the partial sums problem over an array $A[1 \mathinner{..} n]$, where each element comes from the group $\mathbb{G} = S_{\sqrt{V}}$, the permutation group on $\sqrt{V}$ elements. We consider a graph whose vertices form an integer grid of size $\sqrt{V}$ by $\sqrt{V}$; see Figure 3-5.

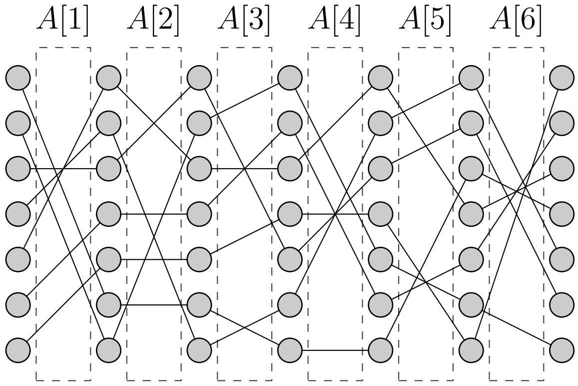


Figure 3-5: Our hard instance of dynamic connectivity implements the partial-sums problem over the group $S_{\sqrt{V}}$.

Edges only connect vertices from adjacent columns; the edges between column i and $i+1$ describe the permutation $A[i]$ from the partial sums problem.

In other words, the graph is a disjoint union of $\sqrt{V}$ paths. Each path stretches from column 1 to column $\sqrt{V}$, and the paths are permuted arbitrarily between columns. This has a strong partial-sums flavor: a node at coordinates $(1, y_1)$ on the first column is connected to a node (k, y_2) on the kth column, if and only if the partial sum permutation $A[1] \circ \cdots \circ A[k-1]$ has y_1 going to y_2.

Given our choice of the group $\mathbb{G}$, we have $\delta = \lg\big((\sqrt{V})!\big) = \Theta\big(\sqrt{V} \cdot \lg V\big)$. For dynamic connectivity, we concentrate on the natural word size $w = \Theta(\lg V)$, so this group is represented by $\Theta(\sqrt{V})$ memory words. Even though these huge group elements may seem worrisome (compared to the previous setting where each $A[i]$ was a word), notice that nothing in our proof depended on the relation between δ and w. Our lower bound still holds, and it implies that a partial-sums operation requires $\Omega(\frac{\delta}{w} \cdot \lg n) = \Omega(\frac{\sqrt{V} \lg V}{\lg V} \cdot \lg \sqrt{V}) = \Omega(\sqrt{V} \cdot \lg V)$ cell probes on average.

Observe that a partial sums UPDATE can be implemented by $O(\sqrt{V})$ dynamic connectivity updates: when some $A[i]$ changes, we run $\sqrt{V}$ DELETE's of the old edges between columns i and $i+1$, followed by $\sqrt{V}$ INSERT's of the new edges. If we could implement SUM using $O(\sqrt{V})$ CONNECTED queries, we would deduce that the dynamic connectivity problem requires a running time of $\Omega(\frac{\sqrt{V} \cdot \lg V}{\sqrt{V}}) = \Omega(\lg V)$ per operation.

Unfortunately, it is not clear how to implement SUM through few CONNECTED queries, since connectivity queries have boolean output, whereas SUM needs to return a permutation with $\Theta\big(\sqrt{V} \cdot \lg V\big)$ bits of entropy. To deal with this issue, we introduce a conceptual change to the partial sums problem, considering a different type of query:

VERIFY-SUM(k, σ): test whether SUM$(k) = \sigma$.

This query is easy to implement via $\sqrt{V}$ connectivity queries: for $i = 1$ to $\sqrt{V}$, these queries test whether the point $(1, i)$ from the first column is connected to point $(k, \sigma(k))$ from the kth column. This runs a pointwise test of the permutation equality $\sigma = A[1] \circ \cdots \circ A[k-1]$.

Below, we extend our lower bound to partial sums with VERIFY-SUM queries:

Theorem 3.8. *In the cell-probe model with w-bit cells, any data structure requires $\Omega(\frac{\delta}{w} n \lg n)$ expected time to support a sequence of n UPDATE and n VERIFY-SUM operations, drawn from a certain probability distribution.*

By our construction, this immediately implies that, in the cell-probe model with cells of $\Theta(\lg V)$ bits, dynamic connectivity requires $\Omega(\lg V)$ time per operation.

3.7 The Main Trick: Nondeterminism

Our lower bound technique thus far depends crucially on the query answers having high entropy, which lower bounds the information transfer, by Lemma 3.2. High entropy is natural for SUM queries, but impossible for VERIFY-SUM. To deal with this problem, we augment our model of computation with nondeterminism, and argue that SUM and VERIFY-SUM are equivalent in this stronger model.

Technically, a nondeterministic cell-probe algorithm is defined as follows. In the beginning of each query, an arbitrary number of *threads* are created. Each thread proceeds independently according to the following rules:

1. first, the thread may read some cells.
2. the thread decides whether the *accept* or *reject*. Exactly one thread must accept.
3. the accepting thread may now write some cells, and must output the answer.

An alternative view of our model is that an all-powerful prover reveals the query answer, and then probes a minimal set of cells sufficient to certify that the answer is correct. We define the *running time* of the query as the number of cell reads and writes executed by the accepting[1] thread.

A deterministic algorithm for VERIFY-SUM running in time t immediately implies a nondeterministic algorithm for SUM, also running in time t. The algorithm for SUM starts by guessing the correct sum (trying all possibilities in separate threads), and verifying the guess using VERIFY-SUM. If the guess was wrong, the thread rejects; otherwise, it returns the correct answer.

3.8 Proof of the Nondeterministic Bound

We will now show that the lower bound for partial sums holds even for nondeterministic data structures, implying the same bound for VERIFY-SUM queries. The only missing part of the proof is a new version of Lemma 3.2, which bounds the entropy of (nondeterministic) queries in terms of the information transfer.

Remember that in Definition 3.1, we let the information transfer $IT(t_0, t_1, t_2)$ be the set of cells that were: (1) read at a time $t_r \in [t_1, t_2]$; and (2) written at a time $t_w \in [t_0, t_1]$, and

[1]There is no need to consider rejecting threads here. If we have a bound t on the running time of the accepting thread, the algorithm may immediately reject after running for time $t + 1$, since it knows it must be running a rejecting thread.

not overwritten during $[t_w + 1, t_r]$. Condition 2. remains well defined for nondeterministic algorithms, since only the unique accepting thread may write memory cells. For condition 1., we will only look at the reads made by the accepting threads, and ignore the rejecting threads.

More formally, let the *accepting execution* of a problem instance be the sequence of cell reads and writes executed by the updates and the *accepting* threads of each query. Define:

$W(t_0, t_1)$: the set of cells written in the accepting execution during time interval $[t_0, t_1]$.

$R(t_0, t_1)$: the set of cells read in the accepting execution during time interval $[t_0, t_1]$, which had last been written at some time $t_w < t_0$.

Then, $IT(t_0, t_1, t_2) = W(t_0, t_1) \cap R(t_1, t_2)$. Our replacement for Lemma 3.2 states that:

Lemma 3.9. $H\big(A_{[t_1,t_2]} \mid \Delta^\star = \mathbf{\Delta}^\star\big) \;\le\; O\big(\mathbf{E}\big[w\cdot|IT(t_0,t_1,t_2)| + |R(t_0,t_2)| + |W(t_1,t_2)| \;\big|\; \Delta^\star = \mathbf{\Delta}^\star\big]\big)$

Note that this lemma is weaker than the original Lemma 3.2 due to the additional terms depending on $W(t_0, t_1)$ and $R(t_1, t_2)$. However, these terms are fairly small, adding $O(1)$ bits of entropy per cell, as opposed to $O(w)$ bits for each cell in the information transfer. This property will prove crucial.

Before we prove the lemma, we redo the analysis of §3.4, showing that we obtain the same bounds for nondeterministic data structures. As before, we consider a lower-bound tree $\mathcal{T}$, whose n leaves represent time units (query and update pairs). For each internal node v of $\mathcal{T}$, let $[t_0, t_1]$ span the left subtree, and $[t_1, t_2]$ span the right subtree. We then define $IT(v) = |IT(t_0, t_1, t_2)|$, $W(v) = |W(t_0, t_1)|$, and $R(v) = |R(t_1, t_2)|$.

Let T be the total running time of the data structure on a particular instance. As before, observe that each cell read in the accepting execution is counted in exactly one $IT(v)$, at the lowest common ancestor of the read and write times. Thus, $T \ge \sum_{v\in\mathcal{T}} IT(v)$.

For each node, we compare Lemmas 3.9 and 3.4 to obtain a lower bound in terms of the interleave at the node:

$$\mathbf{E}[w \cdot IT(v) + W(v) + R(v)] = \Omega(\delta \cdot IL(v)) \tag{3.1}$$

Note that summing up $R(v) + W(v)$ over the nodes on a single level of the tree gives at most T, because each instruction is counted in at most one node. Thus, summing (3.1) over all $v \in \mathcal{T}$ yields: $\mathbf{E}[w\cdot T + T\cdot \mathrm{depth}(\mathcal{T})] = \Omega(\delta \sum_v IL(v))$. By using the bit-reversal permutation and letting $\mathcal{T}$ be a perfect binary tree, we have $\sum_v IL(v) = \Omega(n \lg n)$, and $\mathrm{depth}(\mathcal{T}) = \lg n$. Since $w = \Omega(\lg n)$ in our model, the lower bound becomes $\mathbf{E}[2w\cdot T] = \Omega(\delta\cdot n \lg n)$, as desired.

Proof of Lemma 3.9. The proof is an encoding argument similar to Lemma 3.2, with one additional complication: during decoding, we do not know which thread will accept, and we must simulate all of them. Note, however, that the cells read by the rejecting threads are not included in the information transfer, and thus we cannot afford to include them in the encoding. But without these cells, it is not clear how to decode correctly: when simulating a rejecting thread, we may think incorrectly that a cell was *not* written during $[t_0, t_1]$. If we give the algorithm a stale version of the cell (from before time t_0), a rejecting thread might now turn into an accepting thread, giving us an incorrect answer.

To fix this, our encoding will contain two components:

C1: for each cell in $IT(t_0, t_1, t_2)$, store the cell address, and the contents at time t_1.

C2: a dictionary for cells in $\big(W(t_0, t_1) \cup R(t_1, t_2)\big) \setminus IT(t_0, t_1, t_2)$, with one bit of associated information: "W" if the cell comes from $W(t_0, t_1)$, and "R" if it comes from $R(t_1, t_2)$.

Component C2 will allow us to stop the execution of rejecting threads that try to read a "dangerous" cell: a cell written in $[t_0, t_1]$, but which is not in the information transfer (and thus, its contents in unknown). The presence of C2 in the encoding accounts for a covert information transfer: the fact that a cell was *not* written during $[t_0, t_1]$ is a type of information that the algorithm can learn during $[t_1, t_2]$.

The immediate concern is that the dictionary of C2 is too large. Indeed, a dictionary storing a set S from some universe U, with some r-bit data associated to each element, requires at least $\lg \binom{|U|}{|S|} + |S| \cdot r$ bits of space. Assuming that the space of cells is $[2^w]$, C2 will use roughly $(|W(t_0, t_1)| + |R(t_1, t_2)|) \cdot w$ bits of space, an unacceptable bound that dominates the information transfer.

We address this concern by pulling an interesting rabbit out of the hat: a retrieval-only dictionary (also known as a "Bloomier filter"). The idea is that the membership (is some $x \in S$?) and retrieval (return the data associated with some $x \in S$) functionalities of a dictionary don't need to be bundled together. If we only need the retrieval query and never run membership tests, we do not actually need to store the set S, and we can avoid the lower bound of $\lg \binom{|U|}{|S|}$ bits:

Lemma 3.10. *Consider a set S from a universe U, where each element of S has r bits of associated data. We can construct a data structure occupying $O(|S| \cdot r + \lg\lg |U|)$ bits of memory that answers the following query:*
RETRIEVE(x) : *if $x \in S$, return x's associated data; if $x \notin S$, return an arbitrary value.*

Proof. To the reader familiar with the field, this is a simple application of perfect hash functions. However, for the sake of completeness, we choose to include a simple proof based on the the probabilistic method.

Let $n = |S|, u = |U|$. Consider a hash function $h : U \to [2n]$. If the function is injective on S, we can use an array with $2n$ locations of r bits each, storing the data associated to each $x \in S$ at $h(x)$. For retrieval, injectivity of S guarantees that the answer is correct whenever $x \in S$.

There are $\binom{2n}{n} \cdot n! \cdot (2n)^{u-n}$ choices of h that are injective on S, out of $(2n)^u$ possibilities. Thus, if h is chosen uniformly at random, it works for any fixed S with probability $\binom{2n}{n} \cdot n!/(2n)^n \geq 2^{-O(n)}$. Pick a family $\mathcal{H}$ of $2^{O(n)} \cdot \lg \binom{u}{n}$ independently random h. For any fixed S, the probability that no $h \in \mathcal{H}$ is injective on S is $(1 - \frac{1}{2^{O(n)}})^{|\mathcal{H}|} = \exp\big(\Theta\big(\lg \binom{u}{n}\big)\big) < 1/\binom{u}{n}$. By a union bound over all $\binom{u}{n}$ choices of S, there exists a family $\mathcal{H}$ such that for any S, there exists $h \in \mathcal{H}$ injective on S.

Since our lemma does not promise anything about the time efficiency of the dictionary, we can simply construct $\mathcal{H}$ by iterating over all possibilities. The space will be $2n \cdot r$ bits for the array of values, plus $\lg|\mathcal{H}| = O(n + \lg\lg u)$ bits to specify a hash function from $\mathcal{H}$ that is injective on S. $\square$

We will implement C2 using a retrieval-only dictionary, requiring $O\big(|W(t_0,t_1)|+|R(t_1,t_2)|\big)$ bits of space. Component C1 requires $O(w)\cdot|IT(t_0,t_1,t_2)|$ bits. It only remains to show that the query answers $A_{[t_1,t_2]}$ can be recovered from this encoding, thus giving an upper bound on the entropy of the answers.

To recover the answers $A_{[t_1,t_2]}$, we simulate the execution of the data structure during $[t_1,t_2]$. Updates, which do not use nondeterminism, are simulated as in Lemma 3.2. For a query happening at time $t \in [t_1,t_2]$, we simulate all possible threads. A cell read by one of these threads falls into one of the following cases:

$W(t_1,t)$: We can recognize this case by maintaining a list of memory locations written during the simulation; the contents of the cell is immediately available.

$IT(t_0,t_1,t_2)$: We can recognize this case by examining the addresses in C1; the cell contents can be read from the encoding.

$W(t_0,t_1) \setminus IT(t_0,t_1,t_2)$: We can recognize this case by querying the dictionary C2. If the retrieval query returns "W," we know that the answer *cannot* be "R" (the correct answer may be "W," or the cell may be outside the dictionary set, in which case an arbitrary value is returned). But if the answer cannot be "R," the cell cannot be in $R(t_1,t_2)$. Thus, this thread is certainly not the accepting thread, and the simulation may reject immediately.

$W(1,t_0-1) \setminus W(t_0,t)$: This is the default case, if the cell doesn't satisfy previous conditions. The contents of the cell is known, because the operations before time t_0 are fixed, as part of $\Delta^\star$.

It should be noted that C2 allows us to handle an arbitrary number of rejecting threads. All such threads are either simulated correctly until they reject, or the simulation rejects earlier, when the algorithm tries to read a cell in $W(t_0,t_1) \setminus IT(t_0,t_1,t_2)$.

3.9 Bibliographical Notes

In our paper [86], we generalize the argument presented here to prove lower bound trade-offs between the update time t_u and the query time t_q. We omit these proofs from the current thesis, since our improved epoch arguments from the next chapter will yield slightly better trade-offs than the ones obtained in [86].

Dictionaries supporting only retrieval have found another beautiful application to the range reporting problem in one dimension. See our paper [76] for the most recent work on 1-dimensional range reporting. *Dynamic* dictionaries with retrieval were investigated in our paper [37], which gives tight upper and lower bounds.

Chapter 4

Epoch-Based Lower Bounds

This chapter presents a subtle improvement to the classic chronogram technique of Fredman and Saks [51], which enables it to prove logarithmic lower bounds in the cell-probe model.. To fully appreciate this development, one must remember that the chronogram technique was the only known approach for proving dynamic lower bounds from 1989 until our work in 2004 [86]. At the same time, obtaining a logarithmic bound in the cell-probe model was viewed as one of the most important problems in data-structure lower bounds. It is now quite surprising to find that the answer has always been this close.

Formally, our result is the following:

Theorem 4.1. *Consider an implementation of the partial-sums problem over a group $\mathbb{G}$ with $|\mathbb{G}| \geq 2^\delta$, in the cell-probe model with b-bit cells. Let t_u denote the expected amortized running time of an update, and t_q the expected running time of a query. Then, in the average case of an input distribution, the following lower bounds hold:*

$$\begin{aligned} t_q \lg \left(\frac{t_u}{\lg n} \cdot \frac{b + \lg\lg n}{\delta} \right) &= \Omega\left(\left\lceil \frac{\delta}{b + \lg\lg n} \right\rceil \cdot \lg n \right) \\ t_u \lg \left(\frac{t_q}{\lg n} \Big/ \left\lceil \frac{\delta}{b + \lg\lg n} \right\rceil \right) &= \Omega\left(\frac{\delta}{b + \lg(t_q / \lceil \frac{\delta}{b} \rceil)} \cdot \lg n \right) \end{aligned}$$

Note that the theorem does not assume $\delta \leq b$, so it also gives interesting results in the bit-prove model, where group element are larger than a single cell. Another strengthening of the chronogram technique apparent in this theorem is that it is now possible to derive lower bound trade-offs in the regime of fast updates and slow queries. This implies almost matching the bounds achieved by buffer trees, which constitute one of the most important tools for external-memory algorithms.

Before we prove the theorem, we first apply it in some interesting interesting setups, and compare with the best previously known results.

4.1 Trade-offs and Higher Word Sizes

Assuming $b = \Omega(\lg n)$ and $\delta \le b$, our bounds simplify to:

$$t_q \left(\lg \frac{t_u}{\lg n} + \lg \frac{b}{\delta} \right) = \Omega(\lg n) \qquad\qquad t_u \lg \frac{t_q}{\lg n} = \Omega\left(\frac{\delta}{b} \cdot \lg n \right)$$

The first trade-off, which is optimal [86], represents a strengthening of the normal trade-offs obtained by the chronogram technique. Note in particular that our trade-off implies $\max\{t_u, t_q\} = \Omega(\lg n)$, which had been a major open problem since [51].

The second trade-off for fast updates is fundamentally new; all previous technique are helpless in the regime $t_q \ge t_u$.

Buffer trees [18] are a general algorithmic paradigm for obtaining fast updates, given a higher cell size. For our problem, this yields a *cell-probe* upper bound of $t_u = O(\left\lceil \frac{\delta + \lg n}{b} \cdot \lg_{t_q/\lg n} n \right\rceil)$, for any $t_q = \Omega(\lg n)$. Thus, we obtain tight bounds when $\delta = \Omega(\lg n)$. (Note that in the cell-probe model, we have a trivial lower bound of $t_u \ge 1$, matching the ceiling in the upper bound.)

To appreciate these bounds in a natural setup, let us consider the external memory model, which is the main motivation for looking at a higher cell size. In this model, the unit for memory access is a *page*, which is modeled by a cell in the cell-probe model. A page contains B *words*, which are generally assumed to have $\Omega(\lg n)$ bits. The model also provides for a *cache*, a set of cells which the algorithm can access at zero cost. We assume that the cache is *not* preserved between operations (algorithmic literature is ambivalent in this regard). This matches the assumption of the cell-probe model, where each operation can only learn information by probing the memory. Note that the nonuniformity in the cell-probe model allows unbounded internal state for an operation, so any restriction on the size of the cache cannot be captured by cell-probe lower bounds.

Under the natural assumption that δ matches the size of the word, we see that our lower bound becomes $t_u = \Omega(\frac{1}{B} \lg_{t_q/\lg n} n)$. Buffer trees offer a matching upper bound, if the update algorithm is afforded a cache of $\Omega(t_q/\lg n)$ pages. As mentioned before, we cannot expect cell-probe lower bounds to be sensitive to cache size.

4.2 Bit-Probe Complexity

The bit-probe model is an instantiation of the cell-probe model with one-bit cells. Bit-probe complexity can be considered a fundamental measure of computation. Though a cell size of $\Theta(\lg n)$ bits is more realistic when comparing to real-world computers, the bit-probe measure posses an appeal of its own, due to the exceedingly clean mathematical setup.

Setting $b = \delta = 1$, which is the most natural interpretation of partial sums in the bit-probe model, our lower bounds simplify to:

$$t_q \lg\left(\frac{t_u}{\lg n/\lg\lg n}\right) = \Omega(\lg n) \qquad\qquad t_u \cdot \lg\left(\frac{t_q}{\lg n}\right)\cdot \lg t_q = \Omega(\lg n)$$

The folklore solution to the problem achieves the following trade-offs:

$$t_q \lg\frac{t_u}{\lg n} = \Omega(\lg n) \qquad\qquad t_u \cdot \lg\frac{t_q}{\lg n} = \Omega(\lg n)$$

It can be seen that our lower bounds come close, but do not exactly match the upper bounds. In the most interesting point of balanced running times, the upper bound is $\max\{t_u, t_q\} = O(\lg n)$, while our lower bound implies $\max\{t_u, t_q\} = \Omega(\frac{\lg n}{\lg\lg\lg n})$. Thus, our lower bound is off by just a triply logarithmic factor.

Previously, the best known lower bound was $\max\{t_u, t_q\} = \Omega(\frac{\lg n}{\lg\lg n})$ achieved by Fredman in 1982 [48]. This was by a reduction to the greater-than problem, which Fredman introduced specifically for this purpose. As we showed in [87], there is an $O(\frac{\lg n}{\lg\lg n})$ upper bound for this problem, so Fredman's technique cannot yield a better result for partial sums.

Dynamic connectivity and a record bit-probe bound. With $b = 1$ and superconstant δ, Theorem 4.1 easily implies a nominally superlogarithmic bound on $\max\{t_u, t_q\}$. For instance, for partial sums in $\mathbb{Z}/n\mathbb{Z}$ (i.e. $\delta = \lg n$), we obtain $\max\{t_u, t_q\} = \Omega(\frac{\lg^2 n}{\lg\lg n \cdot \lg\lg\lg n})$. This is a modest improvement over the $\Omega(\frac{\lg^2 n}{(\lg\lg n)^2})$ bound of Fredman and Saks [51].

However, it is not particularly relevant to judge the magnitude of such bounds, as we are only proving a hardness of $\widetilde{\Omega}(\lg n)$ per bit in the query output and update input, and we can obtain arbitrarily high nominal bounds. As advocated by Miltersen [72], the proper way to gauge the power of lower bound techniques is to consider problems with a minimal set of operations, and, in particular, decision queries. Specifically, for a language L, we look at the dynamic language membership problem, defined as follows. For any fixed n (the problem size), maintain a string $w \in \{0,1\}^n$ under two operations: flip the i-th bit of w, and report whether $w \in L$.

Based on our partial sums lower bound, we prove a lower bound of $\Omega((\frac{\lg n}{\lg\lg n})^2)$ for dynamic connectivity, which is a dynamic language membership problem. This has has an important complexity-theoretic significance, as it is the highest known bound for an explicit dynamic language membership problem. The previous record was $\Omega(\lg n)$, shown in [74]. Miltersen's survey of cell-probe complexity [72] lists improving this bound as the first open problem among three major challenges for future research.

It should be noted that our $\widetilde{\Omega}(\lg^2 n)$ bound is far from a mere echo of a $\widetilde{\Omega}(\lg n)$ bound in the cell-probe model. Indeed, $\Omega(\frac{\lg n}{\lg\lg n})$ bounds in the cell-probe model have been known since 1989 (including for dynamic connectivity), but the bit-probe record has remained just the slightly higher $\Omega(\lg n)$. Our bound is the first to show a quasi-optimal $\widetilde{\Omega}(\lg n)$ separation between bit-probe complexity and the cell-probe complexity with cells of $\Theta(\lg n)$ bits, when the cell-probe complexity is superconstant.

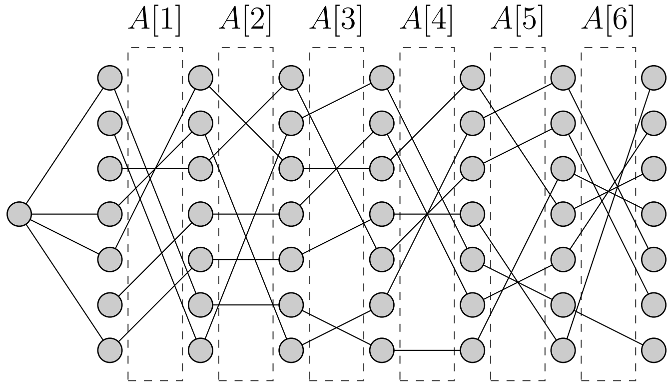


Figure 4-1: Our graphs can be viewed as a sequence of $\sqrt{n}$ permutation boxes.

The main trick for obtaining this lower bound is to use the trade-offs for slow queries and fast updates, a regime in which we give the first known lower bounds. It is not hard to convert a decision query into one returning a large output, at the price of an appropriate slow down. This is the second time, after the analysis of buffer trees, when our extension of the chronogram technique for the regime of slow queries turns out to be very relevant.

Theorem 4.2. *Consider a bit-probe implementation for dynamic connectivity, in which updates take expected amortized time t_u, and queries take expected time t_q. Then, in the average case of an input distribution, $t_u = \Omega\left(\frac{\lg^2 n}{\lg^2(t_u+t_q)}\right)$. In particular $\max\{t_u, t_q\} = \Omega\left((\frac{\lg n}{\lg\lg n})^2\right)$.*

Proof. We first describe the shape of the graphs used in the reduction to Theorem 4.1; refer to Figure 4-1. The vertex set is roughly given by an integer grid of size $\sqrt{n} \times \sqrt{n}$. The edge set is given by a series of permutation boxes. A permutation box connects the nodes in a column to the nodes in the next column arbitrarily, according to a given permutation in $S_{\sqrt{n}}$. Notice that the permutations decompose the graph into a collection of $\sqrt{n}$ paths. As the paths evolve horizontally, the y coordinates change arbitrarily at each point due to the permutations. In addition to this, there is a special *test vertex* to the left, which is connected to some vertices in the first column.

We now describe how to implement the partial sums macro-operations in terms of the connectivity operations:

UPDATE(i, π)**:** sets $\pi_i = \pi$. This is done by removing all edges in permutation box i and inserting new edges corresponding to the new permutation π. Thus, the running time is $O(t_u \cdot \sqrt{n})$.

SUM(i)**:** returns $\sigma = \pi_1 \circ \cdots \circ \pi_i$. We use $O(\lg n)$ phases, each one guessing a bit of $\sigma(j)$ for all j. Phase k begins by removing all edges incident to the test node. Then, we add edges from the test vertex to all vertices in the first column, whose row number has a one in the k-th bit. Then, we test connectivity of all vertices from the i-th column and the test node, respectively. This determines the k-th bit of $\sigma(j)$ for all j. In total, SUM takes time $O((t_u + t_q)\sqrt{n} \cdot \lg n)$.

Finally, we interpret the lower bounds of Theorem 4.1 for these operations. We have $b = 1$ and $\delta = \Theta(\sqrt{n} \cdot \lg n)$. The first trade-off is less interesting, as we have slowed down queries by a factor of $\lg n$. The second trade-off becomes:

$$t_u\sqrt{n}\cdot\lg\left(\frac{(t_u+t_q)\sqrt{n}\cdot\lg n}{\sqrt{n}\cdot\lg^2 n/\lg\lg n}\right) = \Omega\left(\frac{\sqrt{n}\cdot\lg n}{\lg(t_u+t_q)}\cdot\lg n\right) \quad\Rightarrow\quad t_u\lg\left(\frac{t_u+t_q}{\lg n/\lg\lg n}\right) = \Omega\left(\frac{\lg^2 n}{\lg(t_u+t_q)}\right)$$

Since the lower bound implies $\max\{t_u, t_q\} = \Omega((\frac{\lg n}{\lg\lg n})^2)$, we have $\lg(\frac{t_u+t_q}{\lg n/\lg\lg n}) = \Theta(\lg(t_u + t_q))$, so the bound simplifies to $t_u = \Omega(\frac{\lg^2 n}{\lg^2(t_u+t_q)})$. □

4.3 Lower Bounds for Partial Sums

We begin by reviewing the chronogram method, at an intuitive level. One first generates a sequence of random updates, ended by one random query. Looking back in time from the query, one partitions the updates into exponentially growing epochs: for a certain r, epoch i contains the r^i updates immediately *before* epoch $i-1$. One then argues that for all i, the query needs to read at least one cell from epoch i with constant probability. This is done as follows. Clearly, information about epoch i cannot be reflected in earlier epochs (those occurred back in time). On the other hand, the latest $i-1$ epochs contain only $O(r^{i-1})$ updates. Assume the cell-probe complexity of each update is bounded by t_u. Then, during the latest $i-1$ epochs, only $O(t^{i-1}t_u b)$ bits are written. If $r = C \cdot t_u \frac{b}{\delta}$ for a sufficiently large constant C, this number is at most, say, $\frac{1}{10}r^i\delta$. On the other hand, updates in epoch i contain $r^i\delta$ bits of entropy, so all information known outside epoch i can only fix a constant fraction of these updates. If a random query is forced to learn information about a random update from epoch i, it is forced to read a cell from epoch i with constant probability, because the information is not available outside the epoch. This means a query must make $\Omega(1)$ probes in expectation into every epoch, so the lower bound on the query time is given by the number of epochs that one can construct, i.e. $t_q = \Omega(\log_r n) = \Omega(\frac{\lg n}{\lg(t_u b/\delta)})$. A trade-off of this form was indeed obtained by [8], and is the highest trade-off obtained by the chronogram method. Unfortunately, even for $\delta = b$, this only implies $\max\{t_u, t_q\} = \Omega(\frac{\lg n}{\lg\lg n})$.

We now describe the new ideas that we use to improve this result. Intuitively, the analysis done by the chronogram technique is overly pessimistic, in that it assumes all cells written in the latest $i-1$ epochs concentrate on epoch i, encoding a maximum amount of information about it. In the setup from above, this may actually be tight, up to constant factors, because the data structure knows the division into epochs, and can build a strategy based on it. However, we can randomize the construction of epochs to foil such strategies. We generate a random number of updates, followed by one query; since the data structure cannot anticipate the number of updates, it cannot base its decisions on a known epoch pattern. Due to this randomization, we intuitively expect each update to write $O(\frac{t_u b}{\log_r n})$ bits "about" a random epoch, as there are $\Theta(\lg_r n)$ epochs in total. In this case, it would suffice to pick r satisfying $r = \Theta(\frac{t_u b}{\delta \lg_r n})$, i.e. $\lg r = \Theta(\lg \frac{b \cdot t_u}{\delta \lg n})$. This yields $t_q = \Omega(\log_r n) = \Omega(\frac{\lg n}{\lg(t_u/\lg n)+\lg(b/\delta)})$, which means $\max\{t_u, t_q\} = \Omega(\lg n)$ when $\delta = b$.

Unfortunately, formalizing the intuition that the information written by updates "splits" between epochs seems to lead to elusive information theoretic arguments. To circumvent this, we need a second very important idea: we can look at cell reads, as opposed to cell writes. Indeed, regardless of how many cells epochs 1 through $i-1$ write, the information recorded about epoch i is bounded by the information that was read out of epoch i in the first place. On the other hand, the information theoretic value of a read is more easily graspable, as it is dictated by combinatorial properties, like the time when the read occurs and the time when the cell was last written. We can actually show that in expectation, $O(\frac{t_u}{\log_r n})$ of the reads made by each update obtain information about a random epoch. Then, regardless of how many cells are written, subsequent epochs can only encode little information about epoch i, because very little information was read by the updates in the first place.

Once we have this machinery set up, there is a potential for applying a different epoch construction. Assume t_u is already "small". Then, since we don't need to divide t_u by too much to get few probes into each epoch, we can define epochs to grow less than exponentially fast. In particular, we will define epochs to grow by a factor of r *every* r *times*, which means we can obtain a higher lower bound on t_q (in particular, $t_q = \omega(\lg n)$ is possible). Such a result is inherently impossible to obtain using the classic chronogram technique, which decides on the epoch partition in advance. As discussed in the introduction, this is a crucial contribution of our paper, since it leads both to an understanding of buffer trees, and a $\omega(\lg n)$ bit-probe lower bound.

4.3.1 Formal Framework

We first formalize the overall construction. We consider $2M-1$ random updates, and insert a random query at a uniformly random position after the M-th update. Now we divide the last M operations before the query into k epochs. Denote the lengths of the epochs by $\ell_1, \dots, \ell_k$, with ℓ_1 being the closest to the query. For convenience, we define $s_i = \sum_{j=1}^{i} \ell_j$.

Our analysis will mainly be concerned with two random variables. Let T_i^{u} be the number of probes made during epochs $\{1, \dots, i-1\}$ that read a cell written during epoch i. Also let T_i^{q} be the number of probes made by the query that read a cell written during epoch i.

All chronogram lower bounds have relied on an information theoretic argument showing that if epochs 1 up to $i-1$ write too few cells, T_i^{q} must be bounded from below (usually by a constant). As explained above, we instead want to argue that if T_i^{u} is too small, T_i^{q} must be large. Though crucial, this change is very subtle, and the information theoretic analysis follows the same general principles. The following lemma, the proof of which is deferred to Section 4.3.4, summarizes the results of this analysis:

Lemma 4.3. *For any i such that $s_i \le \sqrt[3]{n}$, the following holds in expectation over a random instance of the problem:*

$$\frac{\mathbf{E}[T_i^{\mathrm{u}}]}{\ell_i}\left(b + \lg \frac{t_u s_{i-1}}{\mathbf{E}[T_i^{\mathrm{u}}]}\right) + \mathbf{E}[T_i^{\mathrm{q}}] \cdot \min\left\{\delta, b + \lg \frac{t_q}{\mathbf{E}[T_i^{\mathrm{q}}]}\right\} = \Omega(\delta)$$

We will set $M = \sqrt[3]{n}$ so that the lemma applies to all epochs i. The lower bound of the lemma is reasonably easy to grasp intuitively. The first term measures the average information future updates learn about each of the ℓ_i updates in epoch i. There are T_i^{u} future probes into epoch i. In principle, each one gathers b bits. However, there is also information hidden in the choice of *which* future probes hit epoch i. This amounts to $O(\lg \frac{t_u s_{i-1}}{\mathbf{E}[T_i^{\mathrm{u}}]})$ bits per probe, since the total number of future probes is in expectation $t_u s_{i-1}$ (there are s_{i-1} updates in future epochs). The second term in the expression quantifies the information learned by the query about epoch i. If the query makes T_i^{q} probes into epoch i, each one extracts b bits of information directly, and another $O(\lg \frac{t_q}{\mathbf{E}[T_i^{\mathrm{q}}]})$ bits indirectly, by the choice of which probes hit epoch i. However, there is also another way to bound the information (hence the min). If $\mathbf{E}[T_i^{\mathrm{q}}] \leq 1$, we have probability at most T_i^{q} that the query reads *any* cell from epoch i. If no cell is read, the information is zero. Otherwise, the relevant information is at most δ, since the answer of the query is δ bits. Finally, the lower bound on the total information gathered (the right hand side of the expression) is $\Omega(\delta)$ because a random query needs a random prefix sum of the updates happening in epoch i, which has $\Omega(\delta)$ bits of entropy.

Apart from relating to T_i^{u} instead of cell writes, the essential idea of this lemma is not novel. However, our version is particularly general, presenting several important features. For example, we achieve meaningful results for $\mathbf{E}[T_i^{\mathrm{q}}] > 1$, which is essential to analyzing the case $\delta > b$. We also get a finer bound on the "hidden information" gathered by a cell probe, such as the $O(\lg \frac{t_u s_{i-1}}{\mathbf{E}[T_i^{\mathrm{u}}]})$ term. In contrast, previous results could only bound this by $O(\lg n)$, which is irrelevant when $b = \Omega(\lg n)$, but limits the lower bounds for the bit-probe model.

It is easy and instructive to apply Lemma 4.3 using the ideas of the classic chronogram technique. Define epochs to grow exponentially with rate $r \geq 2$, i.e. $\ell_i = r^i$ and $s_i = O(r^i)$. Assume for simplicity that t_u and t_q are worst-case bounds per operation. Then $T_i^{\mathrm{u}} \leq t_u \cdot s_{i-1}$, since the number of probes into epoch i is clearly bounded by the total number of probes made after epoch i. By Lemma 4.3 we can write $O(\frac{s_{i-1}}{\ell_i} t_u b) + \mathbf{E}[T_i^{\mathrm{q}}]\delta = \Omega(\delta)$, which means $O(\frac{t_u b}{r}) + \mathbf{E}[T_i^{\mathrm{q}}]\delta = \Omega(\delta)$. Setting $r = C t_u \frac{b}{\delta}$ for a sufficiently large constant C, we obtain $\mathbf{E}[T_i^{\mathrm{q}}] = \Omega(1)$. Then $t_q \geq \sum_i \mathbf{E}[T_i^{\mathrm{q}}] = \Omega(\log_r M) = \Omega(\frac{\lg n}{\lg(t_u b/\delta)})$.

As explained before, the key to improving this bound is to obtain a better bound on $\mathbf{E}[T_i^{\mathrm{u}}]$. The next section gives an analysis leading to such a result. Then, Section 4.3.3 uses this analysis to derive our lower bounds.

4.3.2 Bounding Probes into an Epoch

Since we will employ two different epoch constructions, our analysis needs to talk about general ℓ_i and s_i. However, we will need to relate to a certain exponential behavior of the epoch sizes. This property is captured by defining a parameter $\beta = \max_{i^*} \left(\sum_{i \geq i^*} \frac{\min\{\ell_i, s_{i-1}, s_{i^*}\}}{\ell_i} \right)$.

Lemma 4.4. *In expectation over a random instance of the problem and a uniformly random $i \in \{1, \ldots, k\}$, we have $\mathbf{E}[\frac{T_i^{\mathrm{u}}}{\ell_i}] = O(\frac{\beta}{k} \cdot t_u)$.*

Proof. Fix the sequence of updates arbitrarily, which fixes all cell probes. Let T be the total number of cell probes made by updates. Now consider an arbitrary cell probe, and analyze the probability it will be counted towards T_i^{u}. Let r be the time when the probe is executed, and w the time when the cell was last written, where "time" is given by the index of the update. Let i^* be the unique value satisfying $s_{i^*-1} \le r - w < s_{i^*}$.

Note that if $i < i^*$, for any choice of the query position after r, epoch i will begin after w. In this case, the probe cannot contribute to T_i^{u}.

Now assume $i \ge i^*$, and consider the positions for the query such that the cell probe contributes to T_i^{u}. Since w must fall between the beginning of epoch i and its end, there are at most ℓ_i good query positions. In addition, epoch $i-1$ must begin between $w+1$ and r, so there are at most $r - w < s_{i^*}$ good query positions. Finally, epoch $i-1$ must begin between $r - s_{i-1} + 1$ and r, so there are at most s_{i-1} good query positions. Since there are M possible choices for the query position, the probability the cell probe contributes to T_i^{u} is at most $\frac{\min\{\ell_i, s_{i^*}, s_{i-1}\}}{M}$.

We now consider the expectation of $\frac{T_i^{\mathrm{u}}}{\ell_i}$ over the choice of i and the position of the query. We apply linearity of expectation over the T cell probes. A probe with a certain value i^* contributes to the terms $\frac{\min\{\ell_i, s_{i^*}, s_{i-1}\}}{Mk\ell_i}$ for any $i \ge i^*$. The sum of all terms for one cell probe is bounded by $\frac{\beta}{Mk}$, so the expectation of $\frac{T_i^{\mathrm{u}}}{\ell_i}$ is bounded by $\frac{\beta T}{kM}$. Finally, we also take the expectation over random updates. By definition of t_u, $\mathbf{E}[T] \le (2M-1)t_u$. Then $\mathbf{E}[\frac{T_i^{\mathrm{u}}}{\ell_i}] = O(\frac{\beta}{k} t_u)$. □

We now analyze the two epoch constructions that we intend to use. In the first case, epochs grow exponentially at a rate of $r \ge 2$, i.e. $\ell_i = r^i$. Then, $s_i \le 2r^i$, so:

$$\sum_{i \ge i^*} \frac{\min\{\ell_i, s_{i-1}, s_{i^*}\}}{\ell_i} \le \frac{s_{i^*-1}}{\ell_{i^*}} + \sum_{i > i^*} \frac{s_{i^*}}{\ell_i} \le \frac{2}{r} + \sum_{j=1}^{\infty} \frac{2}{r^j} = O\left(\frac{1}{r}\right)$$

Then, $\beta = O(\frac{1}{r})$, and $k = \Theta(\log_r M) = \Theta(\log_r n)$, so $\frac{\beta}{k} = O(\frac{1}{r \log_r n})$.

In the second case, assume $r \le \sqrt{M}$ and construct r epochs of size r^j, for all $j \ge 1$. Then $k = \Theta(r \log_r \frac{M}{r}) = \Theta(r \log_r n)$. Note that $s_i \le (r+2)\ell_i$, since s_i includes at most r terms equal to ℓ_i, while the smaller terms represent r copies of an exponentially decreasing sum with the highest term $\frac{\ell_i}{r}$. Now we have:

$$\sum_{i \ge i^*} \frac{\min\{\ell_i, s_{i-1}, s_{i^*}\}}{\ell_i} \le \sum_{i \ge i^*} \min\{1, \frac{s_{i^*}}{\ell_i}\} \le \sum_{i \ge i^*} \min\{1, \frac{(r+2)\ell_{i^*}}{\ell_i}\} \le r \cdot 1 + r(r+2) \sum_{j=1}^{\infty} \frac{1}{r^j} = O(r)$$

This means $\beta = O(r)$ and $\frac{\beta}{k} = O(\frac{r}{r \log_r n}) = O(\frac{1}{\log_r n})$.

Comparing the two constructions, we see that the second one has r times more epochs, but also r times more probes per epoch. Intuitively, the first construction is useful for large t_u, since it can still guarantee few probes into each epoch. The second one is useful when t_u is already small, because it can construct more epochs, and thus prove a higher lower bound on t_q.

4.3.3 Deriving the Trade-offs of Theorem 4.1

We now put together Lemma 4.3 with the analysis of the previous section to derive our lower bound trade-offs. In the previous section, we derived bounds of the form $\mathbf{E}[\frac{T_i^{\mathrm{u}}}{\ell_i}] = O(\frac{\beta}{k} \cdot t_u)$, where the expectation is also over a random i. By the Markov bound, for at least $\frac{2k}{3}$ choices of i, the bound holds with the constant in the O-notation tripled. Also note that $t_q \geq \sum_i E[T_i^{\mathrm{q}}]$, so for at least $\frac{2k}{3}$ choices of i, we have $\mathbf{E}[T_i^{\mathrm{q}}] \leq \frac{3t_q}{k}$. Then for at least $\frac{k}{3}$ choices of i the above bounds of T_i^{u} and T_i^{q} hold simultaneously. These are the i for which we apply Lemma 4.3.

Since the expression of Lemma 4.3 is increasing in $\mathbf{E}[\frac{T_i^{\mathrm{u}}}{\ell_i}]$ and $\mathbf{E}[T_i^{\mathrm{q}}]$, we can substitute upper bounds for these, obtaining:

$$\begin{aligned}
& \frac{\beta}{k} t_u \left(b + \lg \frac{t_u s_{i-1}/\ell_i}{(\beta/k) t_u} \right) \;+\; \frac{t_q}{k} \cdot \min \left\{ \delta, b + \lg \frac{t_q}{3t_q/k} \right\} = \Omega(\delta) \\
\Rightarrow\; & \frac{\beta}{k} t_u \left(b + \lg \frac{s_{i-1}/\ell_i}{\beta/k} \right) \;+\; \frac{t_q}{k} \Big/ \max \left\{ \frac{1}{\delta}, \frac{1}{b + \lg k} \right\} = \Omega(\delta) \\
\Rightarrow\; & \frac{\beta}{k} t_u \cdot \frac{b + \lg(s_{i-1}k/(\ell_i \beta))}{\delta} \;+\; \frac{t_q}{k} \Big/ \left\lceil \frac{\delta}{b + \lg k} \right\rceil = \Omega(1)
\end{aligned} \tag{4.1}$$

Since the left hand side is increasing in $\frac{\beta}{k}$, we can again substitute an upper bound. This bound is $\frac{\Theta(1)}{r \log_r n}$ for the first epoch construction, and $\frac{\Theta(1)}{\log_r n}$ for the second one. Also note that $\frac{s_{i-1}}{\ell_i} = O(\frac{1}{r})$ in the first construction and $O(r)$ in the second one. Then $\lg \frac{s_{i-1}k}{\ell_i \beta}$ becomes $O(\lg k)$.

Now let us analyze the trade-off implied by the first epoch construction. Note that it is valid to substitute the upper bound $\lg k \leq \lg \lg n$ in (4.1). Also, we use the calculated values for k and $\frac{\beta}{k}$:

$$\frac{t_u}{r \log_r n} \cdot \frac{b + \lg \lg n}{\delta} \;+\; \frac{t_q}{\log_r n} \Big/ \left\lceil \frac{\delta}{b + \lg \lg n} \right\rceil = \Omega(1) \tag{4.2}$$

We can choose r large enough to make the first term smaller than any constant $\varepsilon > 0$. This is true for r satisfying $\varepsilon \frac{r}{\lg r} > \frac{t_u}{\lg n} \cdot \frac{b + \lg \lg n}{\delta}$, which holds for $\lg r = \Theta(\lg(\frac{t_u}{\lg n} \cdot \frac{b + \lg \lg n}{\delta}))$. For a small enough constant ε, the second term in (4.2) must be $\Omega(1)$, which implies our tradeoff:

$$t_q \lg \left(\frac{t_u}{\lg n} \cdot \frac{b + \lg \lg n}{\delta} \right) = \Omega \left(\left\lceil \frac{\delta}{b + \lg \lg n} \right\rceil \cdot \lg n \right)$$

Now we move to the second epoch construction. Remember that $k = \Theta(r \log_r n)$. We can choose r such that the second term of (4.1) is $\Theta(\varepsilon)$, i.e. bounded both from above and from below by small constants. For small enough ε, the $O(\varepsilon)$ upper bound implies that the first term of (4.1) is $\Omega(1)$:

$$\frac{t_u}{\log_r n} \cdot \frac{b + \lg(r \log_r n)}{\delta} = \Omega(1) \quad \Rightarrow \quad t_u \lg r = \Omega \left(\frac{\delta}{b + \lg(r \log_r n)} \cdot \lg n \right) \tag{4.3}$$

To understand this expression, we need the following upper bounds:

$$\frac{t_q}{r \log_r n} \ \Big/ \ \left\lceil \frac{\delta}{b + \lg(r \log_r n)} \right\rceil = \Omega(\varepsilon)$$
$$\Rightarrow \begin{cases} \frac{t_q}{r \log_r n} \Big/ \left(\left\lceil \frac{\delta}{b + \lg \lg n} \right\rceil \cdot \frac{1}{\lg r} \right) = \Omega(1) \Rightarrow \lg r = O\left(\lg \left(\frac{t_q}{\lg n} \Big/ \left\lceil \frac{\delta}{b + \lg \lg n} \right\rceil \right) \right) \\ \frac{t_q}{r \log_r n} \Big/ \left(\left\lceil \frac{\delta}{b} \right\rceil \cdot \frac{1}{\lg(r \log_r n)} \right) = \Omega(1) \Rightarrow \lg(r \log_r n) = O\left(\lg \left(t_q \Big/ \left\lceil \frac{\delta}{b} \right\rceil \right) \right) \end{cases}$$

Plugging into (4.3), we obtain our final tradeoff:

$$t_u \lg \left(\frac{t_q}{\lg n} \Big/ \left\lceil \frac{\delta}{b + \lg \lg n} \right\rceil \right) = \Omega \left(\frac{\delta}{b + \lg(t_q / \lceil \frac{\delta}{b} \rceil)} \cdot \lg n \right)$$

4.3.4 Proof of Lemma 4.3

Remember that our goal is to prove that for any epoch i with $s_i \leq \sqrt[3]{n}$, the following holds in expectation over a random instance of the problem:

$$\frac{\mathbf{E}[T_i^{\mathrm{u}}]}{\ell_i} \left(b + \lg \frac{t_u s_{i-1}}{\mathbf{E}[T_i^{\mathrm{u}}]} \right) + \mathbf{E}[T_i^{\mathrm{q}}] \cdot \min \left\{ \delta, b + \lg \frac{t_q}{\mathbf{E}[T_i^{\mathrm{q}}]} \right\} = \Omega(\delta) \tag{4.4}$$

Pick ℓ_i queries independently at random, and imagine that each is run as the query in our hard instance. That is, each of these queries operates on its own copy of the data structure, all of which are in the same state. Now we define the following random variables:

$Q^I =$ the indices of the ℓ_i queries.

$Q^A =$ the correct answers of the ℓ_i queries.

$U_i^I =$ the indices of the updates in epoch i.

$U_i^\Delta =$ the Δ parameters of the updates in epoch i.

$U_{\neg i}^{I\Delta} =$ the indices and Δ parameters of the updates in all epochs except i.

By [86, Lemma 5.3], $H(Q^A \mid Q^I, U_i^I, U_{\neg i}^{I\Delta}) = \Omega(\ell_i \delta)$, where H denotes conditional binary entropy. This result is very intuitive. We expect the set of query indices Q^I to interleave with the set of update indices U_i^I in $\Omega(\ell_i)$ places. Each interleaving gives a query that extracts δ bits of information about U_i^Δ (it extract a partial sum linearly independent from the rest). Thus, the set of query answers has $\Omega(\ell_i \delta)$ bits of entropy. The cited lemma assumes our condition $s_i \leq \sqrt[3]{n}$, because we do not want updates after epoch i to overwrite updates from epoch i. If there are at most $\sqrt[3]{n}$ updates in epoch i and later, they all touch distinct indices with probability $1 - o(1)$.

We now propose an encoding for Q^A given Q^I and $U^{I\Delta}_{\neg i}$. Comparing the size of this encoding with the previous information lower bound, we will obtain the conclusion of Lemma 4.3. Consider the following random variables:

$T^{\mathrm{u}}_{<i} =$ the number of cell probes made during epochs $\{1, \dots, i-1\}$.

$T^{\mathrm{u}}_{i} =$ as defined previously, the number of cell probes made during epochs $\{1, \dots, i-1\}$ that read a cell written during epoch i.

$T^{\mathrm{Q}} =$ the total number of cell probes made by all ℓ_i queries.

$T^{\mathrm{Q}}_{i} =$ the total number of cell probes made by all ℓ_i queries that read a cell written during epoch i.

Lemma 4.5. *There exists an encoding for Q^A given Q^I and $U^{I\Delta}_{\neg i}$ whose size in bits is:*

$$O\left(T^{\mathrm{u}}_i \cdot b + \lg \binom{T^{\mathrm{u}}_{<i}}{T^{\mathrm{u}}_i} \;+\; \min\left\{T^{\mathrm{Q}}_i \cdot \delta + \lg \binom{\ell_i}{T^{\mathrm{Q}}_i},\;\; T^{\mathrm{Q}}_i \cdot b + \lg \binom{T^{\mathrm{Q}}}{T^{\mathrm{Q}}_i}\right\}\right)$$

Proof. The encoding begins by describing the cell probes made during epochs $\{1, \dots, i-1\}$ into epoch i. First, we specify the subset of probes reading a cell from epoch i in the set of all probes made by future epochs. This takes $O\left(\lg \binom{T^{\mathrm{u}}_{<i}}{T^{\mathrm{u}}_i}\right)$ bits, where the O notation accounts for lower order terms from encoding the integers $T^{\mathrm{u}}_{<i}$ and T^{u}_i using $O(\lg T^{\mathrm{u}}_{<i})$ and $O(\lg T^{\mathrm{u}}_i)$ bits respectively. Second, for all probes into epoch i, we specify the contents of the cell, taking $T^{\mathrm{u}}_i \cdot b$ bits.

We now claim that based on the previous information, one can deduce the contents of all cells that were not last written during epoch i. We can of course simulate the data structure before epoch i, because we know the updates from $U^{I\Delta}_{\neg i}$. Also, we can simulate the data structure after epoch i, because we know which probes read a cell from epoch i, and we have the cell contents in the encoding.

We now choose among two strategies for dealing with the ℓ_i queries. In the first strategy, the encoding specifies all queries which make at least one cell-probe into epoch i. Obviously, there are at most T^{Q}_i such queries, so this takes $O\left(\lg \binom{\ell_i}{T^{\mathrm{Q}}_i}\right)$ bits. For each query making at least one cell probe into epoch i, we simply encode its answer using at most $T^{\mathrm{Q}}_i \cdot \delta$ bits in total. Otherwise, we can simulate the query and find the answer: we know the queried index from Q^I, and we know the contents of all cells that were last written in an epoch other than i.

In the second strategy, the encoding describes all cell probes made by the queries into epoch i. This is done by specifying which is the subset of such cell probes, and giving the cell contents for each one. Thus, in the second strategy we use $T^{\mathrm{Q}}_i \cdot b + O\left(\lg \binom{T^{\mathrm{Q}}}{T^{\mathrm{Q}}_i}\right)$ bits. Given this information, we can simulate all queries and obtain the answers.

It is important to point out that we actually need to know *which* probes touch a cell written during epoch i. Otherwise, we would have no way to know whether a cell has been updated during epoch i, or it still has the old value from the simulation before epoch i. □

We now aim to analyze the expected size of the encoding. By linearity of expectation over the ℓ_i random queries, $\mathbf{E}[T^{\mathrm{Q}}] = t_q \ell_i$ and $\mathbf{E}[T_i^{\mathrm{Q}}] = \mathbf{E}[T_i^{\mathrm{q}}]\ell_i$. Using convexity of $x \mapsto x \lg \frac{y}{x}$, we have:

$$\mathbf{E}\left[\lg \binom{T^{\mathrm{Q}}}{T_i^{\mathrm{Q}}}\right] = O\left(\mathbf{E}\left[T_i^{\mathrm{Q}} \cdot \lg \frac{T^{\mathrm{Q}}}{T_i^{\mathrm{Q}}}\right]\right) = O\left(\mathbf{E}[T_i^{\mathrm{Q}}] \cdot \lg \frac{\mathbf{E}[T^{\mathrm{Q}}]}{\mathbf{E}[T_i^{\mathrm{Q}}]}\right) = O\left(\mathbf{E}[T_i^{\mathrm{q}}]\ell_i \cdot \lg \frac{t_q}{\mathbf{E}[T_i^{\mathrm{q}}]}\right)$$

Similarly, $\mathbf{E}[\lg \binom{\ell_i}{T_i^{\mathrm{Q}}}] = O(\mathbf{E}[T_i^{\mathrm{q}}]\ell_i \cdot \lg \frac{1}{\mathbf{E}[T_i^{\mathrm{q}}]})$.

To bound $T_{<i}^{\mathrm{u}}$, note that it is the sum of s_{i-1} random variables X_j, each giving the number of probes made by the j-th update before the query. By definition of t_u, the total number of probes made by all $2M-1$ updates is in expectation at most $(2M-1)t_u$. Our query is inserted randomly in one of M possible positions, so the update described by X_j is chosen randomly among M possibilities. Then, $\mathbf{E}[X_j] \le \frac{(2M-1)t_u}{M} < 2t_u$, and by linearity of expectation $\mathbf{E}[T_{<i}^{\mathrm{u}}] = O(t_u s_{i-1})$. Then, using convexity as before, we can bound:

$$\mathbf{E}\left[\lg \binom{T_{<i}^{\mathrm{u}}}{T_i^{\mathrm{u}}}\right] = O\left(\mathbf{E}\left[T_i^{\mathrm{u}} \cdot \lg \frac{T_{<i}^{\mathrm{u}}}{T_i^{\mathrm{u}}}\right]\right) = O\left(\mathbf{E}[T_i^{\mathrm{u}}] \cdot \lg \frac{\mathbf{E}[T_{<i}^{\mathrm{u}}]}{\mathbf{E}[T_i^{\mathrm{u}}]}\right) = O\left(\mathbf{E}[T_i^{\mathrm{u}}] \cdot \lg \frac{t_u s_{i-1}}{\mathbf{E}[T_i^{\mathrm{u}}]}\right)$$

We now use the previous calculations and the fact $\mathbf{E}[\min\{a,b\}] \le \min\{\mathbf{E}[a], \mathbf{E}[b]\}$ to bound the expected size of the encoding. Comparing with the entropy lower bound of $\Omega(\delta \ell_i)$, we obtain:

$$\frac{\mathbf{E}[T_i^{\mathrm{u}}]}{\ell_i}\left(b + \lg \frac{t_u s_{i-1}}{\mathbf{E}[T_i^{\mathrm{u}}]}\right) + \mathbf{E}[T_i^{\mathrm{q}}] \cdot \min\left\{\delta + \lg \frac{1}{\mathbf{E}[T_i^{\mathrm{q}}]},\ b + \lg \frac{t_q}{\mathbf{E}[T_i^{\mathrm{q}}]}\right\} \ge c\delta$$

Here c is a positive constant. This is the desired (4.4), except that the first term in the min is $\delta + \lg \frac{1}{T_i^{\mathrm{q}}}$ instead of δ. We now show that this makes no difference up to constant factors. First of all, when the second term in the min is smaller, the expressions are obviously identical. Otherwise, pick a constant $c' > 0$ such that $c' \lg \frac{1}{c'} \le \frac{c}{2}$. If $\mathbf{E}[T_i^{\mathrm{q}}] \le c'$, we have $\mathbf{E}[T_i^{\mathrm{q}} \lg \frac{1}{T_i^{\mathrm{q}}}] \le \frac{c}{2}$. Then, moving the offending term to the right hand side, we obtain a lower bound of $c\delta - \frac{c}{2} = \Omega(\delta)$. Finally, assume $\mathbf{E}[T_i^{\mathrm{q}}] > c'$. Then (4.4) is trivially true if the constant in the Ω notation is at most c', because just the term $\mathbf{E}[T_i^{\mathrm{q}}\delta]$ is larger than the lower bound.

Chapter 5

Communication Complexity

A key tool in showing lower bounds for data structures is asymmetric communication complexity. In this chapter, we introduce communication games and prove some useful lower bounds on their complexity. The connections to data structures are sometimes rather subtle, and will only be apparent in later chapters. While following this chapter, we hope the reader will be motivated by the intrinsic appeal of communication problems, and the promise of future applications.

5.1 Definitions

We consider communication games between two players, traditionally named *Alice* and *Bob*. Alice receives an input x from some set X of allowed inputs, while Bob receives an input $y \in Y$. The goal of the players is to compute some function $f(x, y)$, by communicating to share knowledge about their respective inputs.

The communication between Alice and Bob proceeds according to a predetermined protocol. Players take alternate turns to send messages; each message is a sequence of bits, whose length and contents are dictated by the protocol. Messages must be self-delimiting, i.e. the receiving party must know when the message is over; in virtually all protocols that we discuss, the messages of each player have fixed length, so this property tends to be trivial. At the end of the protocol, both Alice and Bob must know the answer (the output of the function that they want to compute). Since most functions we deal with are boolean, a protocol that fails this criterion can just be augmented with a final 1-bit message in which the answer is communicated to the other player.

5.1.1 Set Disjointness

The most important communication problem that we will consider is the *set disjointness* function: Alice receives a set S, Bob receives a set T, and they want to compute a bit indicating whether $S \cap T = \emptyset$. To specify the problem entirely, assume S and T are subsets of some universe U. The problem is parameterized by $u = |U|$, and quantities n and m

bounding the set sizes: $|S| \leq n$, $|T| \leq m$. In other words, the set X of Alice's inputs consists of all subsets of U with at most n elements; Y consists of all subsets of U with at most m elements.

In standard complexity theory, "set disjointness" usually refers to the symmetric case $n = m = u$. Unfortunately, this problem is not useful for data structures, and we will concentrate on *lopsided set disjointness* (LSD), where $n \ll m$.

As two trivial examples of communication protocols, consider the following:

- Alice sends a message of $O\big(\log_2 \binom{u}{n}\big)$ bits, specifying her set. Bob now knows S, so he knows whether $S \cap T = \emptyset$. He replies with a one-bit message, giving this answer.
- Bob sends a message of $O\big(\log_2 \binom{u}{m}\big)$ bits, specifying T. Alice replies with one bit, giving the answer.

5.1.2 Complexity Measures

The *protocol* solving a communication problem is the equivalent of an *algorithm* solving a computational problem. Given a communication problem, our goal is to design a protocol that is as "efficient" as possible, or prove a lower bound on how efficient the best protocol can be. The two common efficiency measures that we will use are:

- the pair (A, B), where A is the total number of bits communicated by Alice during the protocol, and B the total number of bits communicated by Bob. These quantities are measured in the worst case, i.e. we look at maximum number of bits communicated for any problem instance.
- the number of rounds of communication. Sometimes, we impose the restriction that messages from Alice have some fixed length m_A and messages from Bob some fixed length m_B. Then, the number of rounds is the only parameter of the protocol.

Both of our LSD protocols from above have one round of communication. In the first, $A = \lceil \log_2 \binom{u}{n} \rceil$ and $B = 1$. In the second, $A = 1$ and $B = \lceil \log_2 \binom{u}{m} \rceil$. These suggest a trade-off between A and B; below, we will investigate this trade-off, proving matching upper and lower bounds.

5.2 Richness Lower Bounds

To introduce our first communication lower bound, we concentrate on a very simple communication problem: Indexing. In this problem, Alice receives an index $x \in [m]$, Bob receives a vector $y[1 \dots m]$ of bits, and they want to compute $y[x]$. Indexing can be seen as a special case of LSD with $n = 1$ and $m = u$, where y is the characteristic vector of $U \setminus T$.

Thinking of upper bounds, what trade-off between A and B can we come up with? If Alice is allowed to communicate A bits, she can begin by sending the most significant $A - 1$ bits of x. Bob can reply with the values of Y at every index x that is possible given Alice's message. In other words, he sends a subvector from Y of length $m/2^{A-1}$, as the lower order

$\log_2 m \ - \ A + 1$ bits of x are unknown to him. Now, Alice can just send a final bit with the correct answer.

We thus obtained an upper bound trade-off of $B = \lceil m/2^{A-1} \rceil$. Intuitively, this upper bound seems the best possible. Whatever Alice sends in A bits of communication, there will be an uncertainty of $m/2^A$ about her input. Then, it seems Bob is forced to send the plain values in his vector for all the plausible indices, or otherwise the protocol may make a mistake. Below, we formalize this intuition into a proof showing that $B \geq m/2^{O(A)}$.

5.2.1 Rectangles

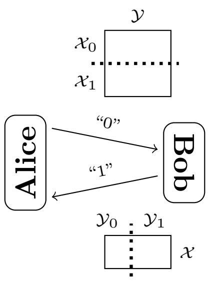
Consider a communication problem $f : X \times Y \to \{0, 1\}$. A *combinatorial rectangle* of f is a matrix minor of the truth table, i.e. a set $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X} \subseteq X$ and $\mathcal{Y} \subseteq Y$. While we call $\mathcal{X} \times \mathcal{Y}$ a rectangle, the reader should note that this is not a rectangle in our usual geometric sense, because the rows in $\mathcal{X}$ and the columns in $\mathcal{Y}$ may not be consecutive.

For a moment, suppose you are an outside observer who doesn't know the inputs of either Alice or Bob, and you watch the communication taking place, trying to guess their inputs. After seeing some transcript of the communication, what have you learned about the input?

Claim 5.1. *The possible problem instances that lead to a fixed communication transcript are always a combinatorial rectangle.*

Proof. This can be seen by induction on the bits sent by the protocol. Before any communication, all inputs are plausible to the outside observer, giving the entire truth table $X \times Y$. Say the next bit is sent by Alice. The value of the bit breaks the plausible inputs of Alice in two disjoint classes: the inputs $\mathcal{X}_0$ for which she would send a "zero," and the inputs $\mathcal{X}_1$ for which she would send a "one." Observing the bit she sent, the observer's belief about the input changes to $\mathcal{X}_i \times Y$. Thus, the belief changes to a subrectangle that drops some of the rows of the old rectangle. Similarly, when Bob sends a bit, the belief changes to a subrectangle dropping some columns. □

Figure 5-1:



Now assume that the observer has watched the communication until the end. What can we say about the resulting rectangle?

Claim 5.2. *At the end of a correct deterministic protocol, one always arrives at a monochromatic rectangle (consisting entirely of zeros, or entirely of ones).*

Proof. At the end of the protocol, both Alice and Bob must know the answer to the problem. But if the rectangle is not monochromatic, there exists a row or a column that is not monochromatic. If, for instance, some row x is bichromatic, Alice sometimes makes a mistake on input x. There are inputs of Bob leading both to zero and one answers, and these inputs are indistinguishable to Alice because they yield the same communication transcript. □

Though this may not be obvious at first, our intuition about the optimality of the protocol for Indexing was an intuitive argument based on rectangles. We reasoned that no matter what Alice communicates in a total of A bits, she cannot (in the worst case) reduce her side of the rectangle to $|\mathcal{X}| < |X|/2^A = m/2^A$. Suppose by symmetry that our final rectangle is monochromatically one. Then, all of Bob's inputs from $\mathcal{Y}$ must have a one at positions in $\mathcal{X}$, so we have $|\mathcal{Y}| \le |Y|/2^{|\mathcal{X}|}$. Since a bit of communication from Bob can at most *halve* $\mathcal{Y}$ on average, Bob must communicate $\Omega(|\mathcal{X}|) = \Omega(m/2^A)$ bits.

5.2.2 Richness

The next step to formalizing our intuition is to break it into two claims:

- If Alice sends A bits and Bob B bits, in the worst case they will obtain a 1-rectangle of size roughly $|X|/2^A \times |Y|/2^B$.
- Any large enough rectangle of the problem at hand contains some zeroes. The quantitative meaning of "large enough" dictates the lower bounds that can be proved.

We now formalize the first claim. Clearly, some minimal assumption about the function is needed (if, say, f were identically zero, one could never arrive at a 1-rectangle).

Definition 5.3. *A function $f : X \times Y \to \{0, 1\}$ is called* $[u, v]$-rich *if its truth table contains at least v columns that have at least u one-entries.*

Lemma 5.4. *Let f be a $[u, v]$-rich problem. If f has a deterministic protocol in which Alice sends A bits and Bob sends B bits, then f contains a 1-rectangle of size at least $u/2^A \times v/2^{A+B}$.*

Proof. By induction on the length of the protocol. Let's say that we are currently in a rectangle $\mathcal{X} \times \mathcal{Y}$ that is $[u, v]$-rich. We have two cases:

- Bob communicates the next bit. Let's say $\mathcal{Y}_0 \subset \mathcal{Y}$ is the set of columns for which he sends zero, and $\mathcal{Y}_1 \subset \mathcal{Y}$ is the set for which he sends one. Since $\mathcal{X} \times \mathcal{Y}$ contains v columns with at least u ones, either $\mathcal{X} \times \mathcal{Y}_0$ or $\mathcal{X} \times \mathcal{Y}_1$ contain $v/2$ columns with at least u ones. We continue the induction in the $[u, \frac{v}{2}]$-rich rectangle.
- Alice communicates the next bit, breaking $\mathcal{X}$ into $\mathcal{X}_0 \cup \mathcal{X}_1$. For an arbitrary column among the v columns that made $\mathcal{X} \times \mathcal{Y}$ rich, we can say that it either has $u/2$ ones in $\mathcal{X}_0$, or $u/2$ ones in $\mathcal{X}_1$. Thus, in either $\mathcal{X}_0 \times \mathcal{Y}$ or $\mathcal{X}_1 \times \mathcal{Y}$, there are at least $v/2$ columns that have at least $u/2$ ones. We continue in a rectangle that is $[\frac{u}{2}, \frac{v}{2}]$-rich.

At the end of the protocol, we reach a monochromatic rectangle that is $[u/2^a,\ v/2^{a+b}]$-rich. Since the rectangle has *some* ones (it has nonzero richness), it must be monochromatically one. Furthermore, it must have size at least $u/2^a$ by $v/2^{a+b}$ to accommodate the richness. $\square$

5.2.3 Application to Indexing

To complete the analysis of INDEXING, note that the problem is $[\frac{m}{2}, 2^{m-1}]$-rich. Indeed, half of the vector settings (i.e. 2^{m-1} columns) have at least $\frac{m}{2}$ ones, because either a vector or its negation have this property. By Lemma 5.4, we obtain a 1-rectangle of size $m/2^{A+1}$ by $2^m/2^{A+B+1}$.

If the rectangle is $\mathcal{X} \times \mathcal{Y} = \{x_1, x_2, \dots\} \times \{y_1, y_2, \dots\}$, every $y_i[x_j]$ must be equal to one. There are only $2^{m-|\mathcal{X}'|}$ distinct y_i's which have all x_j coordinates equal to one. Thus, $|\mathcal{Y}| \le 2^{m-|\mathcal{X}|}$, so $A + B + 1 \ge |\mathcal{X}| = m/2^{A+1}$. We obtain the trade-off lower bound $B \ge m/2^{A+1} - A - 1$, which implies $B \ge m/2^{O(A)}$.

5.3 Direct Sum for Richness

We now return to our original goal of understanding the complexity of LSD. As with indexing, we begin by considering the upper bounds. Armed with a good understanding of the upper bound, the lower bound will become very intuitive.

Our protocol is a simple generalization of the protocol for Indexing, in which Alice sent as many high-order bits of her value as she could afford. Formally, let $k \ge u$ be a parameter. We break the universe $[u]$ into k blocks of size $\Theta(u/k)$. The protocol proceeds as follows:

1. Alice sends the set of blocks in which her elements lie. This takes $A = O\big(\lg \binom{k}{n}\big) = O(n \lg \frac{k}{n})$ bits.
2. For every block containing an element of Alice, Bob sends a vector of $\Theta(u/k)$ bits, indicating which elements are in T. This takes $B = n \cdot \frac{u}{k}$ bits.
3. Alice replies with one more bit giving the answer.

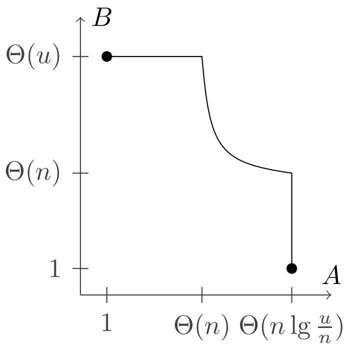


Figure 5-2:

To compute the trade-off, eliminate the parameter k between A and B: we have $k = n \cdot 2^{O(A/n)}$ and thus $B = u/2^{O(A/n)}$. Values of $A = o(n)$ are ineffective: Bob has to send $\Theta(u)$ bits, just as in the trivial protocol in which he describes his entire set. Similarly, to achieve any $B = o(n)$, Alice has to send $\Theta(n \lg \frac{u}{n})$ bits, which allows her to describe her set entirely. The trade-off curve is plotted symbolically in Figure 5-2.

5.3.1 A Direct Sum of Indexing Problems

We now aim to prove a lower bound matching the trade-off from above. This can be done by the elementary richness argument we used for Indexing, but the proof requires some careful bounding of binomial coefficients that describe the rectangle size. Instead, we choose to analyze LSD in an indirect but clean way, which also allows us to introduce an interesting topic in communication complexity: direct sum problems.

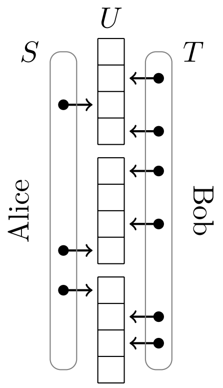


Figure 5-3:

Thinking back of our LSD protocol, the difficult case is when the values in Alice's set fall into different blocks (if two values are in the same block, Bob saves some communication because he only needs to describe one block instead of two). This suggests that we should construct a hard instance by breaking the universe into n

blocks, and placing one of Alice's value in each block (Figure 5-3). Intuitively, this LSD problem consists of n independent copies of INDEXING: each of Alice's values indexes into a vector of u/n bits, describing a block of Bob's set. The sets S and T are disjoint if and only if *all* of the n indices hit elements outside Bob's set (which we can indicate by a "one" in the vector being indexed). Thus, in our family of instances, the LSD query has become the logical AND of n INDEXING queries.

Definition 5.5. *Given a communication problem* $f : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$, *let the communication problem* $\bigwedge^n f : X^n \times Y^n \to \{0, 1\}$ *be defined by* $\big(\bigwedge^n f\big)(\vec{x}, \vec{y}) = \prod_i f_i(x_i, y_i)$.

This is an example of a *direct sum problem*, in which Alice and Bob each receive n independent inputs; the players want to output an aggregate (in this case, the logical AND) of the function f applied to each pair of corresponding inputs. As we have explained above $\bigwedge^n$ INDEXING is a special case of LSD.

Intuitively, a n-wise direct sum problem should have a lower bound that is n times larger than the original. While it can be shown that this property is not always true, we can prove that any *richness* lower bound gets amplified k-fold, in the following sense:

Theorem 5.6. *Let* $f : X \times Y \to \{0, 1\}$ *be* $[\rho|X|, v]$*-rich, and assume* $\bigwedge^n f$ *has a communication protocol in which Alice sends* $A = n \cdot a$ *bits and Bob sends* $B = n \cdot b$ *bits. Then* f *has a 1-rectangle of size* $\rho^{O(1)}|X|/2^{O(a)} \times v/2^{O(a+b)}$.

Before we prove the theorem, let us see that it implies an optimal lower bound for LSD. We showed that INDEXING is $[\frac{|X|}{2}, \frac{|Y|}{2}]$-rich. Then, if LSD has a protocol in which Alice sends A bits and Bob sends B bits, the theorem finds a 1-rectangle of INDEXING of size $|X|/2^{O(A/n)}$ by $|Y|/2^{O(A+B)/n}$. But we showed that any 1-rectangle $\mathcal{X} \times \mathcal{Y}$ must have $|\mathcal{Y}| \le |Y|/2^{|\mathcal{X}|}$, so $2^{O(A+B)/n} \ge |\mathcal{X}| = |X|/2^{O(A/n)}$. The instance of INDEXING are on blocks of size u/n, thus $|X| = \frac{u}{n}$, and we obtain the trade-off:

Theorem 5.7. *Fix* $\delta > 0$. *In a deterministic protocol for* LSD, *either Alice sends* $\Omega(n \lg \frac{m}{n})$ *bits, or Bob sends* $n \cdot \left(\frac{m}{n}\right)^{1-\delta}$ *bits.*

5.3.2 Proof of Theorem 5.6

We begin by restricting Y to the v columns with u one entries. This maintains richness, and doesn't affect anything about the protocol.

Claim 5.8. $\bigwedge^n f$ *is* $[(\rho|X|)^n, v^n]$*-rich.*

Proof. Since $\bigwedge^n f$ only has v^n columns, we want to show that *all* columns contain enough ones. Let $\vec{y} \in Y^n$ be arbitrary. The set of $\vec{x} \in X^n$ with $\big(\bigwedge^n f\big)(\vec{x}, \vec{y}) = 1$ is just the n-wise Cartesian product of the sets $\{x \in X \mid f(x, y_i) = 1\}$. But each set in the product has at least $\rho|X|$ elements by richness of f. □

Now we apply Lemma 5.4 to find a 1-rectangle of $\bigwedge^n f$ of size $(\rho|X|)^n/2^A \times v^n/2^{A+B}$, which can be rewritten as $(\frac{\rho}{2^a}|X|)^n \times (\frac{1}{2^{a+b}}|Y|)^n$. Then, we complete the proof of the theorem by applying the following claim:

Claim 5.9. *If $\bigwedge^n f$ contains a 1-rectangle of dimensions $(\alpha|X|)^n \times (\beta|Y|)^n$, then f contains a 1-rectangle of dimensions $\alpha^3|X| \times \beta^3|Y|$.*

Proof. Let $\mathcal{X} \times \mathcal{Y}$ be the 1-rectangle of $\bigwedge^n f$. Also let $\mathcal{X}_i$ and $\mathcal{Y}_i$ be the projections of $\mathcal{X}$ and $\mathcal{Y}$ on the i-th coordinate, i.e. $\mathcal{X}_i = \{x_i \mid \vec{x} \in \mathcal{X}\}$. Note that for all i, $\mathcal{X}_i \times \mathcal{Y}_i$ is a 1-rectangle for f_i. Indeed, for any $(x, y) \in \mathcal{X}_i \times \mathcal{Y}_i$, there must exists some $(\vec{x}, \vec{y}) \in \mathcal{X} \times \mathcal{Y}$ with $\vec{x}[i] = x$ and $\vec{y}[i] = y$. But $\left(\bigwedge^n f\right)(\vec{x}, \vec{y}) = \prod_j f(x_j, y_j) = 1$ by assumption, so $f(x, y) = 1$.

Now note that there must be at least $\frac{2}{3}n$ dimensions with $|\mathcal{X}_i| \geq \alpha^3|X|$. Otherwise, we would have $|\mathcal{X}| \leq \prod_i |\mathcal{X}_i| < (\alpha^3|X|)^{k/3} \cdot |X|^{2k/3} = (\alpha|X|)^k = |\mathcal{X}|$, contradiction. Similarly, there must be at least $\frac{2}{3}k$ dimensions with $|\mathcal{Y}_i| \geq \beta^3|Y|$. Consequently, there must be an overlap of these good dimensions, satisfying the statement of the lemma. □

This completes the proof of Theorem 5.6.

5.4 Randomized Lower Bounds

5.4.1 Warm-Up

We first prove a slightly weaker randomized lower bound for LSD:

Theorem 5.10. *Assume Alice receives a set $S, |S| = m$ and Bob receives a set $T, |T| = n$, both sets coming from a universe of size $2mn$, for $m < n^\gamma$, where $\gamma < 1/3$ is a constant. In any randomized, two-sided error communication protocol deciding disjointness of S and T, either Alice sends $\Omega(\frac{m}{\log m} \lg n)$ bits or Bob sends $\Omega(n^{1-\delta}/m^2)$ bits, for any $\delta > 0$.*

First we define the hard instance. The elements of our sets come from the universe $[2m] \times [n]$. Alice receives $S = \{(i, s_i) \mid i \in [m]\}$, for $s_1, \dots, s_m$ chosen independently at random from $[n]$. Bob receives $T = \{(t_j, j) \mid j \in [n]$, for $t_1, \dots, t_n$ chosen independently from $[2m]$. The output should be 1 iff the sets are disjoint. Note that the number of choices is n^m for S and $(2m)^n$ for T, and that S and T are chosen independently.

The lower bound follows from the following variant of the richness lemma, based on [73, Lemma 6]. The only change is that we make the dependence on ε explicit, because we will use $\varepsilon = o(1)$.

Lemma 5.11. *Consider a problem $f : X \times Y \to \{0, 1\}$, such that the density of $\{(x, y) \mid f(x, y) = 1\}$ in $X \times Y$ is $\Omega(1)$. If f has a randomized two-sided error $[a, b]$-protocol, then there is a rectangle of f of dimensions at least $|X|/2^{O(a \lg(1/\varepsilon))} \times |Y|/2^{O((a+b) \lg(1/\varepsilon))}$ in which the density of zeros is at most ε.*

To apply the lemma, we first show the disjointness function is 1 with constant probability.

Lemma 5.12. *As S and T are chosen randomly as described above,* $\Pr[S \cap T = \emptyset] = \Omega(1)$.

Proof. Note that $S \cap T \subset [n] \times [m]$. We have $\Pr[(i, j) \in S \cap T] = \frac{1}{n(2m)}$ when $i \in [n], j \in [m]$. Then by linearity of expectation $\mathbf{E}[|S \cap T|] = \frac{1}{2}$. Since $|S \cap T| \in \{0, 1, 2, \dots\}$, we must have $\Pr[|S \cap T| = 0] \geq \frac{1}{2}$. □

Thus, it remains to show that no big enough rectangle has a small density of zeros. Specifically, we show the following:

Lemma 5.13. *Let $\delta > 0$ be arbitrary. If we choose $S \in \mathcal{S}, T \in \mathcal{T}$ uniformly and independently at random, where $|\mathcal{S}| > 2n^{(1-\delta)m}$ and $\mathcal{T} \geq (2m)^n \cdot 2/e^{n^{1-\delta}/(8m^2)}$, then the probability $S \cap T \neq \emptyset$ is at least $\frac{1}{16m^2}$.*

We use the richness lemma with $\varepsilon = \frac{1}{32m^2}$. If there exists an $[a, b]$ protocol for our problem, we can find a rectangle of size $\left(n^m/2^{O(a \lg m)}\right) \times \left((2m)^n/2^{O((a+b)\lg m)}\right)$, in which the fraction of zeros is at most ε. To avoid contradicting Lemma 5.13, we must either have $2^{O(a \lg m)} > n^{\delta m}/2$, or $2^{O((a+b)\lg m)} > e^{n^{1-\delta}/(8m^2)}/2$. This means either $a = \Omega(\frac{m}{\lg m} \lg n)$ or $a + b = \Omega(n^{1-\delta}/(m^2 \lg m))$. If $m < n^\gamma$, for constant $\gamma < \frac{1}{3}$, this implies that $a = \Omega(\frac{m}{\lg m} \lg n)$ or $b = \Omega(n^{1-\delta}/m^2)$, for any $\delta > 0$.

Proof. (of Lemma 5.13) Choosing S at random from $\mathcal{S}$ induces a marginal distribution on $[n]$. Now consider the heaviest $n^{1-\delta}$ elements in this distribution. If the total probability mass of these elements is at most $1 - \frac{1}{2m}$, we call i a *well-spread coordinate.*

Lemma 5.14. *If $|\mathcal{S}| > 2n^{(1-\delta)m}$, there exists a well-spread coordinate.*

Proof. Assume for contradiction that no coordinate is well-spread. Consider the set $\mathcal{S}'$ formed by $S \in \mathcal{S}$ such that no s_i is outside the heaviest $n^{1-\delta}$ elements in S_i. By a union bound, the probability over $S \in \mathcal{S}$ that some s_i is not among the heavy elements is at most $m\frac{1}{2m} = \frac{1}{2}$. Then, $|\mathcal{S}'| \geq |\mathcal{S}|/2$. On the other hand $|\mathcal{S}'| \leq (n^{1-\delta})^m$, since for each coordinate we have at most $n^{1-\delta}$ choices. This contradicts the lower bound on $|\mathcal{S}|$. $\square$

Let i be a well-spread coordinate. We now lower bound the probability of $S \cap T \neq \emptyset$ by the probability of $S \cap T$ containing an element on coordinate i. Furthermore, we ignore the $n^{1-\delta}$ heaviest elements of S_i. Let the remaining elements be W, and $p(j) = \Pr[s_i = j]$ when $j \in W$. Note that $p(j) \leq 1/n^{1-\delta}$, and $\sum_{j \in W} p(j) \geq \frac{1}{2m}$.

Define $\sigma(T) = \sum_{j \in W : t_j = i} p(j)$. For some choice of T, $\sigma(T)$ gives exactly the probability of an interesting intersection, over the choice of $S \in \mathcal{S}$. Thus, we want to lower bound $\mathbf{E}_T[\sigma(T) \mid T \in \mathcal{T}]$.

Assume for now that T is uniformly distributed in the original space (not in the subspace $\mathcal{T}$). Note that $\sigma(T) = \sum_{j \in W} X_j$, where X_j is a variable equal to $p(j)$ when $t_j = i$ and 0 otherwise. By linearity of expectation, $\mathbf{E}_T[\sigma(T)] = \sum_{j \in W} \frac{p(j)}{2m} \geq 1/(2m)^2$. Since X_j's are independent (t_j's are independent when T is not restricted), we can use a Chernoff bound to deduce $\sigma(T)$ is close to this expectation with very high probability over the choice of T. Indeed, $\Pr[\sigma(T) < \frac{1}{2} \cdot \frac{1}{(2m)^2}] < e^{-n^{1-\delta}/(8m^2)}$.

Now we can restrict ourselves to $T \in \mathcal{T}$. The probability $\sigma(T) < \frac{1}{8m^2}$ is so small, that it remains small even in this restricted subspace. Specifically, this probability is at most $\Pr[\sigma(T) < \frac{1}{8m^2}] / \Pr[T \in \mathcal{T}] \leq \exp(-n^{1-\delta}/(8m^2))/(2\exp(-n^{1-\delta}/(8m^2))) = \frac{1}{2}$. Since $\sigma(T) \geq 0, (\forall)T$, we conclude that $\mathbf{E}_T[\sigma(T) \mid T \in \mathcal{T}] \geq \frac{1}{2} \cdot \frac{1}{8m^2} = \frac{1}{16m^2}$. $\square$

5.4.2 A Strong Lower Bound

We now strengthen Theorem 5.10 to the following:

Theorem 5.15. *Assume Alice receives a set $S, |S| = m$ and Bob receives a set $T, |T| = n$, both sets coming from a universe of size $2mn$, for $m < n^\gamma$, where $\gamma < 1$ is a constant. In any randomized, two-sided error communication protocol deciding disjointness of S and T, either Alice sends $\Omega(m \lg n)$ bits or Bob sends $\Omega(n^{1-\delta})$ bits, for any $\delta > 0$.*

Proof. We have a quadratic universe of size $m \cdot n$, which we view as n blocks of size m. Alice's set contains a random point in each block, and Bob's set contains m/n points in the each block. Note that this is a product distribution and the function is balanced ($\Pr[S \cap T = \emptyset] = \Omega(1)$ and $\Pr[S \cap T \neq \emptyset] = \Omega(1)$). This suggests that we should use randomized richness.

By Lemma 5.11, the problem boils down to proving that in a big enough rectangle $\mathcal{S} \times \mathcal{T}$, $\Pr[S \cap T \neq \emptyset \mid S \in \mathcal{S}, T \in \mathcal{T}] = \Omega(1)$.

For values in our universe $i \in [mn]$, we let $p(i)$ be the probability that a set $S \in \mathcal{S}$ contains i. Note that S contains one element in each block, so $p(\cdot)$ is a probability density function on each block. We have $H(S \in \mathcal{S}) \geq n \lg m - o(n \lg \frac{m}{n})$ by assuming that the rectangle is large (Alice communicate $o(n \lg \frac{m}{n})$ bits). Each S is a vector of n choices, each from a block of size m. The entropy of almost all coordinates must be $\lg m - o(\lg \frac{m}{n})$. This means that the distribution $p(\cdot)$ has weight $1 - o(1)$ on values with $p(i) < \frac{(m/n)^{o(1)}}{m}$. (If too many points have high probability, the entropy must be far from the maximum $\lg m$.)

Having fixed $p(\cdot)$ (depending on $\mathcal{S}$), let us calculate $f(T) = \mathbf{E}_{S \in \mathcal{S}}[|S \cap T|]$. This is obviously $\sum_{i \in T} p(i)$. Now we are going to pick T from the original marginal distribution (ignore $\mathcal{T}$ for now), and we're going to look at the distribution induced on $f(T)$. The mean is $m \cdot \frac{1}{m} = 1$, because the $p(i)$'s on each block summed up to one. Now note that each element of T is chosen independently and picks some $p(i)$ to be added to the sum. Thus we have a Chernoff bound kicking in, and $f(T)$ is concentrated around its mean. The probability of deviating a constant from the mean is given by the upper bounds on $p(i)$. If $\max p(i) = \alpha \cdot \text{mean } p(i) = \frac{\alpha}{m}$, the Chernoff bound is exponential in $m/\operatorname{poly}(\alpha)$. But we showed that $1 - o(1)$ of the weight is on values with $p(i) < \frac{(m/n)^{o(1)}}{m}$. So we can discard the big values, still keeping the expectation of $f(T) = 1 - o(1)$, and apply a Chernoff bound exponential in $m/(m/n)^{o(1)} = n \cdot (\frac{m}{n})^{1-o(1)}$.

Now if Bob communicated $n \cdot (\frac{m}{n})^{1-\varepsilon}$ for $\varepsilon > 0$, then $\mathcal{T}$ is so large, that even if all the deviation from the Chernoff bound is inside $\mathcal{T}$, it cannot change the average of $f(T)$ by more than a constant. (Formally $\Pr[\mathcal{S}]$ is ω of the Chernoff probability.) Then the average of $f(T)$ even on $\mathcal{T}$ is $\Omega(1)$.

We have just shown that $\mathbf{E}_{\mathcal{S} \times \mathcal{T}}[|S \cap T|] = \Omega(1)$. But how do we get $\mathbf{E}[|S \cap T|] = \Omega(1)$ to mean $\Pr[S \cap T \neq \emptyset] = \Omega(1)$? We are going to consider a bicriterion error function: $\widetilde{\varepsilon}(S, T) = \varepsilon(S, T) + \alpha \cdot |S \cap T|^2$. Here ε was the error measure of the protocol, and α is a small enough constant. By concentration $\operatorname{Var}[|S \cap T|] = O(1)$ so $\widetilde{\varepsilon}$ is bounded by a small constant: amplify the original protocol to reduce ε, and choose α small.

Randomized richness gives us a large rectangle for which the protocol answers "disjoint", and in which the error is a constant bigger than the original. This means that $\mathbf{E}_{\mathcal{S} \times \mathcal{T}}[|S \cap$

$T|^2] = O(1)$. But since $|S \cap T|$ is positive, have expectation $\Omega(1)$, and has a bounded second moment, it must be that $\Pr[|S \cap T| > 0] = \Omega(1)$. Thus $\Pr[S \cap T \neq \emptyset] = \Omega(1)$, contradicting the fact that the error was a very small constant. $\square$

5.4.3 Direct Sum for Randomized Richness

It will be useful to have a version of our direct-sum result for randomized richness. We now describe such a result, which comes from our paper [88]. Consider a vector of problems $\vec{f} = (f_1, \ldots, f_k)$, where $f_i : X \times Y \to \{0,1\}$. We define another data structure problem $\bigoplus^k \vec{f} : ([k] \times X) \times Y^k \to \{0,1\}$ as follows. The data structure receives a vector of inputs $(y_1, \ldots, y_k) \in Y^k$. The representation depends arbitrarily on all of these inputs. The query is the index of a subproblem $i \in [k]$, and an element $x \in X$. The output of $\bigoplus^k \vec{f}$ is $f_i(x, y_i)$.

Let us first recapitulate how randomized richness is normally applied to communication games. We say problem f is *α-dense* if $\mathbf{E}_{x \in X, y \in Y}[f(x,y)] \geq \alpha$, i.e. at least an α fraction of the truth table of f contains ones. Then, one applies Lemma 5.11, showing that, in order to prove a communication lower bound, one has to prove that every large rectangle contains $\Omega(1)$ zeros. Unfortunately, we cannot use this lemma directly because we do not know how to convert k outputs, some of which may contain errors, into a single meaningful boolean output. Instead, we need a new lemma, which reuses ideas of the old lemma in a more subtle way. A technical difference is that our new lemma will talk directly about data structures, instead of going through communication complexity.

Define $\rho_i : X \times Y \times \{0,1\} \to \{0,1\}$ by $\rho_i(x,y,z) = 1$ if $f_i(x,y) \neq z$, and 0 otherwise. Also let $\rho : X^k \times Y^k \times \{0,1\}^k \to [0,1]$ be $\rho(x,y,z) = \frac{1}{k} \sum_i \rho_i(x_i, y_i, z_i)$. In other words, ρ measures the fraction of the outputs from z which are wrong.

Lemma 5.16. *Let $\varepsilon > \frac{99}{k}$ be arbitrary, and $f_1, \ldots, f_k$ be ε-dense. Assume $\bigoplus^k \vec{f}$ can be solved in the cell-probe model with w-bit cells, using space S, cell-probe complexity T, and error rate $\leq \varepsilon$. Then there exists a canonical rectangle $\mathcal{X} \times \mathcal{Y} \subset X^k \times Y^k$ for some output $z \in \{0,1\}^k$ satisfying:*

$$|\mathcal{X}| \geq |X|^k / 2^{O(Tk \lg \frac{S}{k})}, \qquad |\mathcal{Y}| \geq |Y|^k / 2^{O(Tkw)}$$
$$\sum_i z_i \geq \frac{\varepsilon}{3} k, \qquad \mathbf{E}_{x \in \mathcal{X}, y \in \mathcal{Y}}[\rho(x,y,z)] \leq \varepsilon^2.$$

Proof. First we decrease the error probability of the data structure to $\frac{\varepsilon^2}{9}$. This requires $O(1)$ repetitions, so it only changes constant factors in S and T. Now we use the easy direction of Yao's minimax principle to fix the coins of the data structure (nonuniformly) and maintain the same error over the uniform distribution on the inputs.

We now convert the data structure to a communication protocol. We simulate one query to each of the k subproblems in parallel. In each round, Alice sends the subset of k cells probed, and Bob replies with the contents of the cells. Alice sends a total of $O(Tk \lg \frac{S}{k})$ bits, and Bob a total of $O(Tkw)$ bits. At the end, the protocol outputs the vector of k answers.

Let $P_i(x_i, y)$ be the output of the data structure when running query (i, x_i) on input y. Note that this may depend arbitrarily on the entire input y, but depends only on one query (since the query algorithm cannot consider parallel queries). When the communication protocol receives x and y as inputs, it will output $P(x, y) = (P_1(x_1, y), \dots, P_k(x_k, y))$. Note that some values $P_i(x_i, y)$ may be wrong (different from $f_i(x_i, y_i)$), hence some coordinates of $P(x, y)$ will contain erroneous answers. To quantify that, note $\mathbf{E}_{x,y}[\rho(x, y, P(x, y))] = \mathbf{E}_{i,x_i,y}[\rho_i(x_i, y_i, P_i(x_i, y))] \leq \frac{\varepsilon^2}{9}$, i.e. the average fraction of wrong answers is precisely the error probability of the data structure.

We now wish to show that the set $W = \{(x, y) \mid \sum_i P_i(x_i, y) \geq \frac{\varepsilon}{3}k\}$ has density $\Omega(1)$ in $X^k \times Y^k$. First consider the set $W_1 = \{(x, y) \mid \sum_i f_i(x_i, y_i) \geq \frac{2\varepsilon}{3}k\}$. As (x, y) is chosen uniformly from $X^k \times Y^k$, $f_i(x_i, y_i)$ are independent random variables with expectation $\geq \varepsilon$. Then, by the Chernoff bound, $\Pr_{x,y}[(x, y) \in W_1] \geq 1 - e^{k\varepsilon/18} \geq 1 - e^{-99/18} \geq \frac{2}{3}$. Now consider $W_2 = \{(x, y) \mid \rho(x, y, P(x, y)) \geq \frac{\varepsilon}{3}\}$. Since $\mathbf{E}_{x,y}[\rho(x, y, P(x, y))] = \frac{\varepsilon^2}{9}$, the Markov bound shows that the density of W_2 is at most $\frac{\varepsilon}{3}$. Finally, observe that $W_1 \setminus W_2 \subseteq W$, so W has density $\geq \frac{1}{3}$.

The communication protocol breaks $X^k \times Y^k$ into disjoint canonical rectangles, over which $P(x, y)$ is constant. Consider all rectangles for which $P(x, y)$ has at least $\frac{\varepsilon}{3}k$ one entries. The union of these rectangles is W. Now eliminate all rectangles R with $\mathbf{E}_{(x,y)\in R}[\rho(x, y, P(x, y))] \geq \varepsilon^2$, and let W' be the union of the remaining ones. Since the average of $\rho(x, y, P(x, y))$ over $X^k \times Y^k$ is $\frac{\varepsilon^2}{9}$, a Markov bound shows the total density of the eliminated rectangles is at most $\frac{1}{9}$. Then, $|W'| \geq \frac{2}{3}|W|$.

Now observe that membership in W' is $[\Omega(|X|^k), \Omega(|Y|^k)]$-rich. Indeed, since $|W'| = \Omega(|X|^k|Y|^k)$, a constant fraction of the rows must contain $\Omega(|Y|^k)$ elements from W'. Now note that the communication protocol can be used to decide membership in W', so we apply Lemma 5.4. This shows that one of the rectangles reached at the end of the protocol must contain only elements of W', and have size $\Omega(|X|^k)/2^{O(Tk \lg(S/k))} \times \Omega(|Y|^k)/2^{O(Tkw)}$. In fact, because Lemma 5.4 finds a large canonical rectangle, this must be one of the rectangles composing W', so we know the answer corresponding to this rectangle has at least $\frac{\varepsilon}{3}k$ ones, and the average $\rho(x, y, P(x, y))$ over the rectangle is at most ε^2. □

The direct-sum result that we want will rely on the following key combinatorial lemma, whose proof is deferred to §5.4.4:

Lemma 5.17. *For $i \in [d]$, consider a family of functions $\phi_i : X \times Y \to \{0, 1\}$, and define $\phi : X^d \times Y^d \to [0, 1]$ by $\phi(x, y) = \frac{1}{d}\sum_i \phi_i(x_i, y_i)$. Let $\mathcal{X} \subset X^d, \mathcal{Y} \subset Y^d$ with $|\mathcal{X}| \geq (|X|/\alpha)^d, |\mathcal{Y}| \geq (|Y|/\beta)^d$, where $\alpha, \beta \geq 2$. Then there exists $i \in [d]$ and a rectangle $A \times B \subset X \times Y$ with $|A| \geq |X|/\alpha^{O(1)}, |B| \geq |Y|/\beta^{O(1)}$, such that $\mathbf{E}_{a\in A, b\in B}[\phi_i(a, b)] = O(\mathbf{E}_{x\in\mathcal{X}, y\in\mathcal{Y}}[\phi(x, y)])$.*

Using this technical result, we can show our main direct-sum property:

Theorem 5.18. *Let $\varepsilon > \frac{99}{k}$ be arbitrary, and $f_1, \dots, f_k$ be ε-dense. Assume $\bigoplus^k \vec{f}$ can be solved in the cell-probe model with w-bit cells, using space S, cell-probe complexity T, and error ε. Then some f_i has a rectangle of dimensions $|X|/2^{O(T \lg(S/k))} \times |Y|/2^{O(Tw)}$ in which the density of zeros is at most ε.*

Proof. First we apply Lemma 5.16, yielding a rectangle $\mathcal{X} \times \mathcal{Y}$. By reordering coordinates, assume the first $d = \frac{\varepsilon}{3}k$ elements of z are ones. We now wish to fix $x_{d+1}, \dots, x_k$ and $y_{d+1}, \dots, y_k$ such that the remaining d-dimensional rectangle is still large, and the average of $\rho(x, y, z)$ over it is small. There are at most $|X|^{k-d}$ choices for fixing the x elements. We can eliminate all choices which would reduce the rectangle by a factor of at least $3|X|^{k-d}$. In doing so, we have lost a $\frac{1}{3}$ fraction of the density. Similarly, we eliminate all choices for the y elements which would reduce the rectangle by a factor of $3|Y|^{k-d}$.

We still have a third of the mass remaining, so the average of $\rho(x, y, z)$ can only have increased by a factor of 3. That means $\mathbf{E}_{i\in[k]}[\rho_i(x_i, y_i, z_i)] \leq 3\varepsilon^2$, which implies $\mathbf{E}_{i\in[d]}[\rho_i(x_i, y_i, z_i)] \leq 3\varepsilon^2 \cdot \frac{k}{d} = 9\varepsilon$. We now fix $x_{d+1}, \dots, x_k$ and $y_{d+1}, \dots, y_k$ among the remaining choices, such that this expected error is preserved. Thus, we have found a rectangle $\mathcal{X}' \times \mathcal{Y}' \subset X^d \times Y^d$ with $|\mathcal{X}'| \geq |X|^d/2^{O(Tk \lg(S/k))}$ and $|\mathcal{Y}'| \geq |Y|^d/2^{O(Tkw)}$. Since $d = \Theta(k)$, we can freely substitute d for k in these exponents. Besides largeness, the rectangle satisfies $\mathbf{E}_{i\in[d], x\in\mathcal{X}', y\in\mathcal{Y}'}[\rho_i(x_i, y_i, 1)] \leq 9\varepsilon$.

We now apply Lemma 5.17 on the rectangle $\mathcal{X}' \times \mathcal{Y}'$, with $\alpha = 2^{O(T \lg(S/k))}$, $\beta = 2^{O(Tw)}$ and $\phi_i(x, y) = \rho_i(x, y, 1)$. We obtain a rectangle $A \times B \subset X \times Y$ of dimensions $|A| \geq |X|/2^{O(T \lg(S/k))}$, $|B| \geq |Y|/2^{O(Tw)}$, which has the property $\mathbf{E}_{a\in A, b\in B}[\rho_i(a, b, 1)] = O(\varepsilon)$, i.e. $\Pr_{a\in A, b\in B}[f_i(a, b) = 0] = O(\varepsilon)$. $\square$

5.4.4 Proof of Lemma 5.17

Define $\mathcal{X}_i$ to be the weighted projection of $\mathcal{X}$ on dimension i (i.e. a distribution giving the frequency of every value on coordinate i). Thus, $\mathcal{X}_i$ is a distribution on X with density function $\wp_{\mathcal{X}_i}(z) = \frac{|\{x\in\mathcal{X} | x_i = z\}|}{|\mathcal{X}|}$.

We identify sets like $\mathcal{X}$ and $\mathcal{Y}$ with the *uniform distributions* on the sets. Treating ϕ and ϕ_i as random variables (measuring some error to be minimized), let $\varepsilon = \mathbf{E}_{\mathcal{X}\times\mathcal{Y}}[\phi] = \frac{1}{d}\sum_i \mathbf{E}_{\mathcal{X}_i\times\mathcal{Y}_i}[\phi_i]$.

We now interpret the lower bound on the size of $\mathcal{X}$ as bounding the entropy, and use submodularity of the Shannon entropy H to write:

$$\sum_i H(\mathcal{X}_i) \;\geq\; H(\mathcal{X}) \;\geq\; d\cdot(\lg|X| - \lg\alpha) \quad\Rightarrow\quad \frac{1}{d}\cdot\sum_i \big(\lg|X| - H(\mathcal{X}_i)\big) \;\leq\; \lg\alpha$$

Observe that each term in the sum is positive, since $H(\mathcal{X}_i) \leq \lg|X|$. We can conclude that:

$$(\exists)i: \qquad \lg|X| - H(\mathcal{X}_i) \;\leq\; 3\lg\alpha; \qquad \lg|Y| - H(\mathcal{Y}_i) \;\leq\; 3\lg\beta; \qquad \mathbf{E}_{\mathcal{X}_i\times\mathcal{Y}_i}[\phi_i] \leq 3\varepsilon,$$

because there are strictly less than $\frac{d}{3}$ coordinates that violate each of these three constraints. For the remainder of the proof, fix some i satisfying these constraints.

Let A' be the set of elements $z \in X$ with $\wp_{\mathcal{X}_i}(z) \leq \alpha^8/|X|$, where $\wp_{\mathcal{X}_i}$ is the density function of the distribution $\mathcal{X}_i$. In the probability space on which distribution $\mathcal{X}_i$ is observed,

A' is an event. We have:

$$\begin{aligned} H(\mathcal{X}_i) &= H(A') + \Pr[A'] \cdot H(\mathcal{X}_i \mid A') + (1 - \Pr[A']) \cdot H(\mathcal{X}_i \mid \neg A') \\ &\leq 1 + \Pr[A'] \cdot \lg|X| + \big(1 - \Pr[A']\big) \cdot \lg \frac{|X|}{\alpha^8} = \lg|X| + 1 - \big(1 - \Pr[A']\big) \cdot 8 \lg \alpha \end{aligned}$$

We claim that $\Pr[A'] \geq \frac{1}{2}$. Otherwise, we would have $H(\mathcal{X}_i) \leq \lg|X| + 1 - 4\lg\alpha$, contradicting the lower bound $H(\mathcal{X}_i) \geq \lg|X| - 3\lg\alpha$, given $\alpha \geq 2$.

Now let $\mathcal{X}'$ be the distribution $\mathcal{X}_i$ conditioned on A' (equivalently, the distribution restricted to the support A'). Performing an analogous analysis on $\mathcal{Y}_i$, we define a support B' and restricted distribution $\mathcal{Y}'$. Observe that:

$$\mathbf{E}_{\mathcal{X}' \times \mathcal{Y}'}[\phi_i] = \mathbf{E}_{\mathcal{X}_i \times \mathcal{Y}_i}[\phi_i \mid A' \wedge B'] \leq \frac{\mathbf{E}_{\mathcal{X}_i \times \mathcal{Y}_i}[\phi_i]}{\Pr[A' \wedge B']} \leq 4 \cdot \mathbf{E}_{\mathcal{X}_i \times \mathcal{Y}_i}[\phi_i] \leq 12\varepsilon$$

We now want to conclude that $\mathbf{E}_{A' \times B'}[\phi_i]$ is small. This is not immediately true, because changing from some distribution $\mathcal{X}'$ on support A' to the uniform distribution on A' may increase the average error. To fix this, we consider a subset $A \subseteq A'$, discarding from A' every value x with $\mathbf{E}_{\{x\} \times \mathcal{Y}'}[\phi_i] > 24\varepsilon$. Since the expectation over x is 12ε, a Markov bound implies that $\Pr[A] \geq \frac{1}{2}\Pr[A'] \geq \frac{1}{4}$. We now have a bound for every $x \in A$, and thus $\mathbf{E}_{A \times \mathcal{Y}'}[\phi_i] \leq 24\varepsilon$. Now perform a similar pruning of B, concluding that $\mathbf{E}_{A \times B}[\phi_i] \leq 48\varepsilon$.

Finally, we must show that $|A| \geq |X|/\alpha^{O(1)}$. This follows because $\Pr_{\mathcal{X}_i}[A] \geq \frac{1}{4}$, and for any $x \in A$ we had $\wp_{\mathcal{X}_i}(x) \leq \alpha^8/|X|$. The same analysis holds for $|B|$.

5.5 Bibliographical Notes

Communication complexity is a major topic of research in complexity theory, with numerous interesting applications. However, "traditional" communication complexity has focused on symmetric problems, and usually studied the total communication of both players, i.e. $A+B$. This measure of complexity does not seem useful for data structures.

Asymmetric communication complexity was introduced by Miltersen, Nisan, Safra, and Wigderson [73] in STOC'95, though it was implicit in previous work by Ajtai [3] and Miltersen [71]. The richness technique, and the first lower bounds for Indexing and LSD date back to this seminal paper [73]. The direct-sum property for richness lower bounds was shown in our paper [88] from FOCS'06.

The randomized lower bound for symmetric set disjointness, originally due to [65, 92], is probably the most used result from traditional communication complexity. Bar-Yossef, Jayram, Kumar, and Sivakumar [19] provide the most accessible (though by no means simple) proof of this result, based on a direct sum idea similar to that of §5.3. The randomized LSD lower bounds date back to our paper [16] from FOCS'06.

Chapter 6

Static Lower Bounds

In this chapter, we prove our first lower bounds for static data structures. The standard approach for such bounds is to convert a cell-probe algorithm into an asymmetric communication protocol. In the communication model, Alice holds a query, and Bob holds the database. The two players communicate to answer the query on the database. Each round of communication simulates a cell probe: the querier sends $\lg S$ bits, where S is the space (the number of cells used by the data structure), and the database responds with w bits, where w is the cell size.

As in the previous chapter, we can analyze this communication problem by the richness method, and show lower bounds of the following form: either Alice must send a bits, or Bob must send b bits. If the data structure makes t cell probes to answer the query, in the communication protocol, Alice sends $t \lg S$ bits, and Bob sends tw bits. Comparing with the lower bounds, one concludes that the cell-probe complexity must be at least $t \geq \min\{\frac{a}{\lg S}, \frac{b}{w}\}$. In general, b is prohibitively large, so the first bound dominates for reasonable word size. Thus, the time-space trade-off can be rewritten as $S \geq 2^{\Omega(a/t)}$.

We analyze two problems in this framework: partial match (in §6.1), and $(1+\varepsilon)$-approximate near neighbor search (in §6.2). These results appeared in our papers [16, 82], and both follow by reduction from the communication complexity of set disjointness. See Chapter 2 for background on these problems.

Decision trees. Intuitively, we do not expect the relation between cell-probe and communication complexity to be tight. In the communication model, Bob can remember past communication, and answer new queries based on this. Needless to say, if Bob is just a table of cells, he cannot remember anything, and his responses must be a function of Alice's last message (i.e. the address of the cell probe).

This reduction to a much stronger model has its limitations. The bounds that we obtain, which look like $S \geq 2^{\Omega(a/t)}$, are tight for constant query time (demonstrating interesting phenomena about the problem), but they degrade too quickly with t, and become uninteresting even for small t.

On the other hand, asymmetric communication complexity can be used to obtain very good lower bounds in a more restricted model of computation: decision trees. In this model, we can typically show that the decision tree must have size $2^{\Omega(a)}$, unless its depth (query time) is prohibitively large.

The reduction between decision trees and asymmetric communication is explored in §6.3.

Beating communication complexity. As mentioned above, the implications of communication lower bounds for cell-probe data structures are far from satisfactory. For example, the entire strategy is, by design, insensitive to polynomial changes in the space (up to constants in the lower bound). But this is unrealistic for data structures, where the difference between space $n \lg^{O(1)} n$ and (say) space $O(n^3)$ is plainly the difference between an interesting solution and an uninteresting one.

To put this in a different light, note that a communication complexity of $O(d)$ bits from the querier equates data structures of size $2^{O(d)}$ which solve the problem in constant time, and data structures of size $O(n)$ which solve the problem in a mere $O(d/\lg n)$ time. Needless to say, this equivalence appears unlikely. Thus, we need new approaches which can understand the time/space trade-offs in the cell-probe model at a finer granularity than direct reduction to communication. We make progress in this direction, in the case when the space is $n^{1+o(1)}$.

Interestingly, we do not need to throw out the old work in the communication model. We can take any lower bound shown by the richness method, for problems with a certain compositional structure, and obtain a better lower bound for small-space data structures by *black-box* use of the old result. Thus, we can boost old bounds for polynomial space, in the case of near-linear space.

Let S be the space in cells used by the data structure. If one uses richness to show a lower bound of $\Omega(d)$ bits for the communication of the querier, the standard approach would imply a cell-probe lower bound of $\Omega(d/\lg S)$. In contrast, we can show a lower bound of $\Omega(d/\lg \frac{Sd}{n})$, which is an improvement for $S = n^{1+o(1)}$. In the most interesting case of near-linear space $S = n(d \lg n)^{O(1)}$, the bound becomes $\Omega(d/\lg d)$. Compared to $\Omega(d/\lg n)$, this is a significant improvement for natural values of d, such as $d = \lg^{O(1)} n$. In particular, for $d = O(\lg n)$, previous lower bounds could not exceed a constant, whereas we obtain $\Omega(\lg n/\lg\lg n)$. Note that for $d = O(\lg n)$ we have constant upper bounds via tabulation, given large enough *polynomial* space.

Our technique is introduced in §6.4, where it is illustrated with the familiar partial match problem. In the next chapter, we give more exciting application to range queries, where our idea yields many tight bounds.

6.1 Partial Match

First, we must clarify the notion of reduction from the communication complexity of LSD to a data-structure problem. In such a reduction, Bob constructs a database based on his set T, and Alice constructs a query based on S. It is then shown that LSD can be solved based on the answer to the k queries on Bob's database. If the data structure has size S and query time t, this reduction in fact gives a communication protocol for LSD, in which Alice communicates $t \lg S$ bits, and Bob communicates tw bits. This is done by simulating the query algorithm: for each cell probe, Alice sends the address, and Bob sends the content from his constructed database. At the end, the answer to LSD is determined from the answer of the query.

In the previous chapter, we showed a lower bound for LSD via a direct sum argument for indexing problems. This gives a special case of LSD, that we call BLOCKED-LSD. In this problem, the universe is interpreted as $[N] \times [B]$, and elements as pairs (u, v). It is guaranteed that $(\forall)x \in [N]$, S contains a single element of the form $(x, \star)$.

Reduction 6.1. BLOCKED-LSD *reduces to one partial match query over $n = N \cdot B$ strings in dimension $d = O(N \lg B)$.*

Proof. Consider a constant weight code ϕ mapping the universe $[B]$ to $\{0,1\}^b$. If we use weight $b/2$, we have $\binom{b}{b/2} = 2^{\Omega(b)}$ codewords. Thus, we may set $b = O(\lg B)$.

If $S = \{(1, s_1), \ldots, (N, s_N)\}$, Alice constructs the query string $\phi(s_1)\phi(s_2)\cdots$, i.e. the concatenation of the codewords of each s_i. We have dimension $d = N \cdot b = O(N \lg B)$.

For each point $(x, y) \in T$, Bob places the string $0^{(x-1)b}\,\phi(y)\,0^{(N-x)b}$ in the database. Now, if $(i, s_i) \in T$, the database contains a string with $\phi(s_i)$ at position $(i-1)b$, and the rest zeros. This string is dominated by the query, which also has $\phi(s_i)$ at that position. On the other hand, if a query dominates some string in the database, then for some $(i, s_i) \in S$ and $(i, y) \in T$, $\phi(s_i)$ dominates $\phi(y)$. But this means $s_i = y$ because in a constant weight code, no codeword can dominate another. $\square$

From the lower bound on BLOCKED-LSD, we know that in a communication protocol solving the problem, either Alice sends $\Omega(N \lg B)$ bits, or Bob sends $N \cdot B^{1-\delta} \geq n^{1-\delta}$ bits. Rewriting this bound in terms of n and d, either Alice sends $\Omega(d)$ bits, or Bob sends $n^{1-\delta}$ bits, for constant $\delta > 0$.

This implies that a data structure with query time t requires space $2^{\Omega(d/t)}$, as long as the word size is $w \leq n^{1-\delta}/t$.

6.2 Approximate Near Neighbor

We consider the decision version of approximate near neighbor over the Hamming space. Given a set $P \subset \{0,1\}^d$ of n points and a distance r, build a data structure which given $q \in \{0,1\}^d$ does the following, with probability at least, say, 2/3:

- If there is $p \in P$ such that $\|q - p\| \leq r$, answer YES;
- If there is no $p \in P$ such that $\|q - p\| \leq (1+\varepsilon)r$, answer NO.

Here we use $\|\cdot\|$ for the Hamming norm. It is standard to assume cells have $\Theta(d)$ bits, i.e. a point can be stored in one cell. The lower bound holds for the Euclidean space as well.

To prove the lower bound, we consider the asymmetric communication complexity of the problem for dimension $d = (\frac{1}{\varepsilon}\lg n)^{O(1)}$. We assume that Alice holds q, while Bob holds P. We show that to solve the problem, either Alice sends $\Omega(\frac{1}{\varepsilon^2}\lg n)$ bits, or Bob sends $\Omega(n^{1-\delta})$ bits, for any constant $\delta > 0$.

By the standard relation to cell-probe complexity, this implies that any data structure with constant cell-probe complexity must use space $n^{\Omega(1/\varepsilon^2)}$, with is optimal (see Chapter 2). Note that the cell size w is usually much smaller than $n^{1-\delta}$, typically $b = d\log^{O(1)} n$, so that bound on Bob's communication is very permissive.

Thus, our result establishes a tight quantitative dependence between the approximation factor and the exponent in the space bound (for the constant query time case). Given that the exponent must be quadratic in $1/\varepsilon$, our results indicate a fundamental difficulty in designing practical data structures which are very accurate *and* very fast.

Theorem 6.2. *Consider the communication complexity of* $(1+\varepsilon)$*-approximate near neighbor in* $\{0,1\}^d$*, where* $d = O(\frac{\log^2 n}{\varepsilon^5})$*. For any* $\varepsilon = \Omega(n^{-\gamma})$*,* $\gamma < 1/2$*, in any randomized protocol deciding the* $(1+\varepsilon)$*-NN problem, either Alice sends* $\Omega(\frac{\log n}{\varepsilon^2})$ *bits or Bob sends* $\Omega(n^{1-\delta})$ *bits, for any* $\delta > 0$*.*

Proof. We show how to map an instance of lopsided set disjointness, given by T and S, into an instance of $(1+\varepsilon)$-approximate near neighbor, given by respectively the dataset $P \subset \{0,1\}^d$ and the query $q \in \{0,1\}^d$. For this purpose, first, Alice and Bob map their sets S and T into query $\tilde{q} \in \Re^U$ and dataset $\tilde{P} \subset \Re^U$, i.e., an $(1+\varepsilon)$-NN instance in Euclidean U-dimensional space, l_2^U. Then, Alice and Bob map their points from the l_2^U metric to Hamming cube $\{0,1\}^{O(\log^2 n/\varepsilon^5)}$, essentially preserving the distances among all the points $\tilde{q}$ and $\tilde{P}$.

For the set $T \subset [U]$, we define $\tilde{P} \triangleq \{e_u \mid u \in T\}$, where e_u is a standard $\Re^d$ basis vector, with 1 in the coordinate u, and 0 everywhere else. For the set S, we set the query $\tilde{q} \triangleq 3\varepsilon \cdot \sum_{u\in S} e_u$; note that $\|\tilde{q}\|_2^2 = m \cdot (3\varepsilon)^2 = 1$.

We show that if $S \cap T = \emptyset$, then $\|\tilde{q} - \tilde{p}\|_2 = \sqrt{2}$ for all $\tilde{p} \in \tilde{P}$, and, if $S \cap T \neq \emptyset$, then there exists a point $\tilde{p} \in \tilde{P}$ such that $\|\tilde{q} - \tilde{p}\|_2 \leq (1 - \frac{4\varepsilon}{3})\sqrt{2}$. Indeed, we have that

- if $S \cap T = \emptyset$, then for any $\tilde{p} \in \tilde{P}$, we have that $\|\tilde{q} - \tilde{p}\|_2^2 = \|\tilde{q}\|_2^2 + \|\tilde{p}\|_2^2 - 2\tilde{q}\cdot\tilde{p} = 2$;
- if $S \cap P \neq \emptyset$, then for $u^* \in S \cap P$ and for $\tilde{p} = e_{u^*} \in P$, we have $\|\tilde{q} - \tilde{p}\|_2^2 = \|\tilde{q}\|_2^2 + \|\tilde{p}\|_2^2 - 2\tilde{q}\cdot\tilde{p} = 2 - 2(3\varepsilon e_{u^*}) \cdot e_{u^*} = 2(1-3\varepsilon)$.

To construct $P \subset \{0,1\}^d$ and $q \in \{0,1\}^d$, Alice and Bob perform a randomized mapping of l_2^U into $\{0,1\}^d$ for $d = O(\log^2 n/\varepsilon^5)$, such that the distances are only insignificantly distorted, with high probability. Alice and Bob use a source of public random coins to construct the same randomized mapping. First, they construct a randomized embedding f_1 mapping l_2^U into $l_1^{O(\log n/\varepsilon^2)}$ with distortion less than $(1+\varepsilon/16)$. Then, they construct the standard embedding f_2 mapping $l_1^{O(\log n/\varepsilon^2)}$ into $\{0,1\}^{O(\log^2 n/\varepsilon^5)}$. The embedding f_2 first scales up all coordinates by $D = O(\frac{\log n}{\varepsilon^3})$, then rounds the coordinates, and finally transforms each coordinate into its unary representation. We set the constants such that the resulting approximation of f_2 is an additive term $O(\frac{\log n}{\varepsilon^2}) < \frac{D\varepsilon\sqrt{2}}{16}$.

Next, Alice and Bob construct $q = f_2(f_1(\tilde{q})) \in \{0,1\}^d$ and $P = \{f_2(f_1(\tilde{p})) \mid \tilde{p} \in \tilde{P}\} \subset \{0,1\}^d$. Notice that for any $p = f_2(f_1(\tilde{p})) \in P$, if $\|\tilde{q} - \tilde{p}\|_2 \geq \sqrt{2}$, then $\|q-p\|_H \geq D\sqrt{2}(1-\varepsilon/16) - \frac{D\varepsilon\sqrt{2}}{16} = D\sqrt{2}(1-\frac{\varepsilon}{8})$, and if $\|\tilde{q} - \tilde{p}\|_2 \leq \sqrt{2}(1-\frac{4\varepsilon}{3})$, then $\|q-p\|_H \leq D\sqrt{2}(1-\frac{4\varepsilon}{3})(1+\varepsilon/16) + \frac{D\varepsilon\sqrt{2}}{16} \leq D\sqrt{2}(1-\varepsilon-\frac{5\varepsilon}{24})$.

Finally, Alice and Bob can run the $(1+\varepsilon)$-NN communication protocol with $r = D\sqrt{2}(1-\varepsilon-\frac{5\varepsilon}{24})$ to decide whether $S \cap T = \emptyset$. Note that the error probability of the resulting set disjointness protocol is bounded away from $1/2$ since $(1+\varepsilon)$-NN communication protocol has error probability bounded away from $1/2$, and the embedding $f_2 \circ f_1$ fails with probability at most $n^{-\Omega(1)}$. □

6.3 Decision Trees

We formally define what we mean by a decision tree for a data structure problem. Consider a partial problem $F : \mathcal{I} \to \{0,1\}$ with $\mathcal{I} \subset X \times Y$, where X is the set of "queries" and Y is the set of "datasets".

For $y \in Y$, a *decision tree* T_y is a complete binary tree in which:

- each internal node v is labeled with a predicate function $f_v : X \to \{0,1\}$. We assume f_v comes from some set $\mathcal{F}$ of *allowed predicates*.
- each edge is labeled with 0 or 1, indicating the answer to the parent's predicate.
- each leaf is labeled with 0 or 1, indicating the outcome of the computation.

Evaluating T_y on x is done by computing the root's predicate on x, following the corresponding edge, computing the next node's predicate, and so on until a leaf is reached. The label of the leaf is the output, denoted $T_y(x)$.

We let the *size* s of the tree to be the total number of the nodes. The *depth* d of the tree is the longest path from the root to a leaf. The *predicate size* is $w = \lceil \log_2 \mathcal{F} \rceil$.

We say that problem F can be solved by a decision tree of size s, depth d, and predicate size w iff, for any y, there exists some tree T_y of size at most s, depth at most d, and node size at most w, such that $T_y(x) = F(x,y)$ whenever $(x,y) \in \mathcal{I}$.

Our result on the decision tree lower bound follows from the following folklore lemma, which converts an efficient decision tree solving a problem F into an efficient communication protocol.

Lemma 6.3. *Consider any (promise) problem $F : \mathcal{I} \to \{0,1\}$, where $\mathcal{I} \subset X \times Y$. Suppose there exists a decision tree of size s, depth d, and node size w.*

If Alice receives $x \in X$ and Bob receives $y \in Y$, there exists a communication protocol solving the problem F, in which Alice sends a total of $a = O(\log s)$ bits and Bob sends $b = O(dw \log s)$ bits.

Proof. Before the protocol, Bob constructs his decision tree T_y. Suppose, for a moment, that the decision tree is balanced, that is $d = O(\log s)$. Then, Alice and Bob can run the following "ideal" protocol. In round one, Bob sends the predicate f_r of the root r of the decision tree. Alice computes $f_r(x)$ (a bit) and sends it back. Then Bob follows the corresponding edge in the tree, and sends the predicate of the corresponding child, etc. We obtain communication $a \le d$ and $b \le w \cdot d$.

In general, however, the decision tree T_D is not balanced. In this case, Alice and Bob can simulate a standard binary search on a tree. Specifically, Bob finds a separator edge that splits the tree in two components, each of size at least $s/3$. Let this separating edge be (u, v). In round one, Alice and Bob want to detect whether, in the ideal protocol, Alice would eventually follow the edge (u, v). To determine this, Bob sends the predicates for all nodes on the path from the root r to u. Alice evaluates these predicates on x and sends back a 1 if she would follow the edge (u, v), and 0 otherwise. Then, the players recurse on the remaining part of the tree; they are done after $O(\log s)$ such rounds.

In the end, Alice sends only $a = O(\log s)$ bits, i.e. one bit per round. Bob sends $O(d \cdot w)$ bits per round, and thus $b = O(dw \log s)$. □

From this, we obtain the following very strong implications for the partial match and near neighbor problems:

Theorem 6.4. *A decision tree for the partial match problem with predicate size $O(n^\delta)$ must either have size $2^{\Omega(d)}$, or depth $\Omega(n^{1-2\delta}/d)$.*

Theorem 6.5. *A decision tree with predicate size $O(n^\delta)$, for $(1+\varepsilon)$-approximate near neighbor search in the Hamming cube $\{0,1\}^d$, must either have size $n^{\Omega(1/\varepsilon^2)}$, or depth $\Omega(n^{1-2\delta}/d)$.*

Note that with depth $O(n)$, one can just perform a linear scan of the database to find the answer. Then, the space (decision tree size) is just linear. These bounds show that for strongly sublinear query time, the space must be prohibitively large.

6.4 Near-Linear Space

We first describe the intuition for why a lower bound of $\Omega(d/\lg n)$ for space $S = n^{O(1)}$, should also imply a lower bound of $\Omega(d/\lg d)$, when the space is $S = n \cdot (d \lg n)^{O(1)}$. For very small databases, namely $n = d^{O(1)}$, the lower bound for polynomial space can be rewritten as $\Omega(d/\lg d)$. If n is larger, one can hope to partition the problem into $k = n/d^{O(1)}$ independent subproblems, each with database of size $N = d^{O(1)}$. Intuitively, each subproblem "gets" space $S/k = (d \cdot \lg n)^{O(1)} = N^{O(1)}$, and hence it requires $\Omega(d/\lg d)$ cell probes.

Transforming this intuition into an actual lower bound is surprisingly simple. Instead of simulating one query as part of a communication protocol, we will simulate k queries in parallel. In each step, the queriers need to send the *subset* of k cells which are probed, among the S cells in memory. Sending this information requires $O(\lg \binom{S}{k}) = O(k \lg \frac{S}{k})$ bits. This is $O(\lg \frac{S}{k})$ bits "on average" per query, whereas the normal reduction sends $O(\lg S)$ bits for one query. We will typically use $k = n/\lg^{O(1)} n$.

Our direct sum results from the previous chapter are crucial in this context. They can show that considering k independent copies increases the communication lower bound by a factor of $\Omega(k)$, which is exactly the intuition described above.

Technical details. Let PM_n^d be the partial match problem with a query in $\{0,1,\star\}^d$ and a database of n strings in $\{0,1\}^d$.

Theorem 6.6. *Consider a bounded error (Monte Carlo) data structure solving* PM_n^d *in the cell-probe model with cells of* $d^{O(1)}$ *bits, using space* S*. Assuming* $d \geq 2 \lg n$*, the cell-probe complexity of a query must be* $\Omega(d/\lg \frac{Sd}{n})$.

Proof. It is easy to convert a solution to PM_n^d into a solution to $\bigoplus^k \mathrm{PM}_N^D$, where $N = n/k$ and $D = d - \lg k \geq d/2$. One simply prefixes query and database strings with the subproblem number, taking $\lg k$ bits.

As we showed above, a lower bound for the communication complexity of partial match can be obtained by a very simple reduction from LSD. Our LSD lower bound from Chapter 5 was by richness. Interpreting this richness lower bound in the context of partial match, we see that on a certain domain $X \times Y$ for PM_N^D, we have:

- PM_N^D is $\frac{1}{2}$-dense.
- for any $\delta > 0$, in any rectangle of size $|X|/2^{O(\delta D)} \times |Y|/2^{O(N^{1-\delta}/D^2)}$, the density of zeros is $\Omega(1)$.

For concreteness, set $\delta = \frac{1}{2}$ in the above result. Applying our direct sum result for randomized richness, Theorem 5.18, to $\bigoplus^k \mathrm{PM}_N^D$, we obtain that either $T \lg \frac{S}{k} = \Omega(D)$, or $T'w = \Omega(\sqrt{N}/D^2)$. Setting $N = w^2 \cdot D^4 \cdot d = d^{O(1)}$, the second inequality becomes $T = \Omega(d)$, while the first becomes $T = \Omega(\lg \frac{Sd}{n})$. We thus conclude that $T \geq \min\{\Omega(d),\ \Omega(d/\lg \frac{Sd}{n})\} = \Omega(d/\lg \frac{Sd}{n})$. ☐

Chapter 7

Range Query Problems

In the previous chapter, we introduced a neat trick in using asymmetric communication complexity to obtain cell-probe lower bounds: consider multiple queries at the same time (on Alice's side) communicating to the same database. This allowed us to obtain lower bounds of $\Omega(\lg n / \lg\lg n)$ for a data structure using space $O(n \cdot \mathrm{polylog}\, n)$ in the cell-probe model with words of $O(\lg n)$ bits.

Unfortunately, our application was not so compelling. We showed $\Omega(\lg n / \lg\lg n)$ lower bounds for partial match, a very hard problem, where the optimal bound should probably be around $n^{1-o(1)}$. In this chapter, we demonstrate significantly more interesting applications. The simple idea of analyzing multiple queries at the same time turns out to capture the fundamental complexity of very important range query problems.

See Chapter 2 for ample background on the various range query problems that we consider here.

Our results stem from a lower bound on an unusual problem: reachability oracles in butterfly graphs.

Theorem 7.1. *Consider a data structure that represents a directed graph G that is a subgraph of a (directed) butterfly graph, and answers the following query:*

REACHABLE(u, v)**:** *is node v reachable from u through a directed path in G?*

If the data structure uses space $n\sigma$ in the cell probe model with w-bit cells, the query time must be $\Omega(\frac{\lg n}{\lg(\sigma+w)})$.

The proof of this result is given in §7.1, and it follows, somewhat surprisingly, by reduction from the complexity of lopsided set disjointness.

In §7.3, we show two corollaries of this result:

Theorem 7.2. *A data structure for orthogonal range stabbing in 2 dimensions using space $n \cdot \sigma$ in the cell probe model with w-bit cells, requires query time $\Omega(\frac{\lg n}{\lg(\sigma w)})$.*

Theorem 7.3. *A data structure for the dynamic marked ancestor problem, with amortized update time $t_u = O(\mathrm{polylog}\, n)$ requires query time $t_q = \Omega\big(\frac{\lg n}{(\lg\lg n)^2}\big)$.*

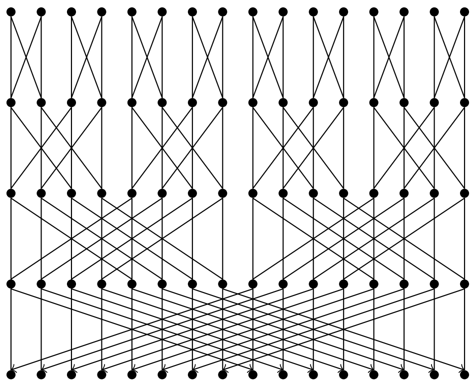


Figure 7-1: A butterfly with degree 2 and depth 4.

The lower bound for 2-dimensional stabbing immediately implies our lower bound for 4-dimensional range reporting, and 2-dimensional range counting by the reductions discussed in Chapter 2.

Our lower bound for the marked ancestor problem reproves the important result of Alstrup, Husfeldt, and Rauhe [8], with a $\lg \lg n$ loss in the query bound. Despite this slightly weaker bound, we feel our proof is interesting, as it shows a dynamic lower bound by arguing only about static problems, and significantly clears up the structure of the problems under consideration.

7.1 The Butterfly Effect

The butterfly is a well-known graph structure with high "shuffle abilities." The graph (Figure 7-1) is specified by two parameters: the degree b, and the depth d. The graph has $d+1$ layers, each having b^d vertices. The vertices on level 0 are sources, while the ones on level d are sinks. Each vertex except the sinks has out-degree d, and each vertex except the sources has in-degree d. If we view vertices on each level as vectors in $[b]^d$, the edges going out of a vertex on level i go to vectors that may differ only on the ith coordinate. This ensures that there is a unique path between any source and any sink: the path "morphs" the source vector into the sink node by changing one coordinate at each level.

For convenience, we will slightly abuse terminology and talk about "reachability oracles for G," where G is a butterfly graph. This problem is defined as follows: preprocess a subgraph of G, to answer queries of the form: is sink v reachable from source u? The query can be restated as: is any edge on the unique source–sink path missing from the subgraph?

7.1.1 Reachability Oracles to Stabbing

The reduction from reachability oracles to stabbing is very easy to explain formally, and we proceed to do that now. However, there is a deeper meaning to this reduction, which will be explored in §7.1.2.

Reduction 7.4. *Let G be a butterfly with M edges. The reachability oracle problem on G reduces to 2-dimensional stabbing over M rectangles.*

Proof. If some edge of G does not appear in the subgraph, what source-sink paths does this cut off? Say the edge is on level i, and is between vertices $(\cdots, v_{i-1}, v_i, v_{i+1}, \cdots)$ and $(\cdots, v_{i-1}, v_i', v_{i+1}, \cdots)$. The sources that can reach this edge are precisely $(\star, \cdots, \star, v_i, v_{i+1}, \cdots)$, where $\star$ indicates an arbitrary value. The sinks that can be reached from the edge are $(\cdots, v_{i-1}, v_i', \star, \cdots)$. The source–sink pairs that route through the missing edge are the Cartesian product of these two sets.

This Cartesian product has precisely the format of a 2D rectangle. If we read a source vector $(v_1, \ldots, v_d)$ as a number in base b with the most significant digit being v_d, the set of sources that can reach the edge is an interval of length b^{i-1}. Similarly, a sink is treated as a number with the most significant digit v_1, giving an interval of length b^{d-i}.

For every missing edge, we define a rectangle with the source and sink pairs that route through it. Then, a sink is reachable from a source iff no rectangle is stabbed by the (sink, source) point. ☐

Observe that the rectangles that we constructed overlap in complicated ways. This is in fact needed, because 2-dimensional range stabbing with non-overlapping rectangles can be solved with query time $O(\lg^2 \lg n)$ [35].

As explained in Chapter 2, 2D range stabbing reduces to 2D range counting and 4D range reporting.

7.1.2 The Structure of Dynamic Problems

The more interesting reduction is to the marked ancestor problem. The goal is to convert a solution to the dynamic problem into a solution to *some* static problem for which we can prove a lower bound.

A natural candidate would be to define the static problem to be the persistent version of the dynamic problem. Abstractly, this is defined as follows:

input: an (offline) sequence of updates to a dynamic problem, denoted by $u_1, \ldots, u_m$.

query: a query q to dynamic problem and a time stamp $\tau \leq m$. The answer should be the answer to q if it were executed by the dynamic data structure after updates $u_1, \ldots, u_\tau$.

An algorithm result for making data structures persistent can be used to imply a lower bound for the dynamic problem, based on a lower bound for the static problem. The following is a standard persistence result:

Lemma 7.5. *If a dynamic problem can be solved with update time t_u and query time t_q, its (static) persistent version will have a solution with space $O(m \cdot t_u)$ and query time $O(t_q \cdot \lg\lg(m \cdot t_u))$.*

Proof. We simulate the updates in order, and record their cell writes. Each cell in the simulation has a collection of values and timestamps (which indicate when the value was updated). For each cell, we build a predecessor structure *a la* van Emde Boas [101] over the time-stamps. The structures occupy $O(m \cdot t_u)$ space in total, supporting queries in $O(\lg\lg(mt_u))$ time. To simulate the query, we run a predecessor query for every cell read, finding the last update that changed the cell before time τ. □

Thus, if the static problem is hard, so is the dynamic problem (to within a doubly logarithmic factor). However, the reverse is not necessarily true, and the persistent version of marked ancestor turns out to be easy, at least for the incremental case. To see that, compute for each node the minimum time when it becomes marked. Then, we can propagate down to every leaf the minimum time seen on the root-to-leaf path. To query the persistent version, it suffices to compare the time stamp with this value stored at the leaf.

As it turns out, persistence is still the correct intuition for generating a hard static problem. However, we need the stronger notion of full persistence. In partial persistence, as seen above, the updates create a linear chain of versions (an update always affects the more recent version). In full persistence, the updates create a *tree* of versions, since updates are allowed to modify any historic version.

For an abstract dynamic problem, its fully-persistent version is defined as follows:

input: a rooted tree (called the *version tree*) in which every node is labeled with a sequence of update operations. The total number of updates is m.

query: a query q to the dynamic problem, and a node τ of the version tree. The answer should be the answer to q if it were executed after the sequence of updates found on the path through the version tree from the root to τ.

Like the partially persistent problem, the fully persistent one can be solved by efficient simulation of the dynamic problem:

Lemma 7.6. *If a dynamic problem can be solved with update time t_u and query time t_q, the fully-persistent static problem has a solution with space $O(m \cdot t_u)$ and query time $O(t_q \lg\lg(m \cdot t_u))$.*

Proof. For each cell of the simulated machine, consider the various nodes of the version tree in which the cell is written. Given a "time stamp" (node) τ, we must determine the most recent change that happened on the path from τ to the root. This is the longest matching prefix problem, which is equivalent to predecessor search. Thus, the simulation complexity is the same as in Lemma 7.5. □

We now have to prove a lower bound for the fully-persistent version of marked ancestor, which we accomplish by a reduction from reachability oracles in the butterfly:

Reduction 7.7. *Let G be a subgraph of a butterfly with M edges. The reachability oracle problem on G reduces to the fully-persistent version of the marked ancestor problem, with an input of $O(M)$ offline updates. The tree in the marked ancestor problem has the same degree and depth as the butterfly.*

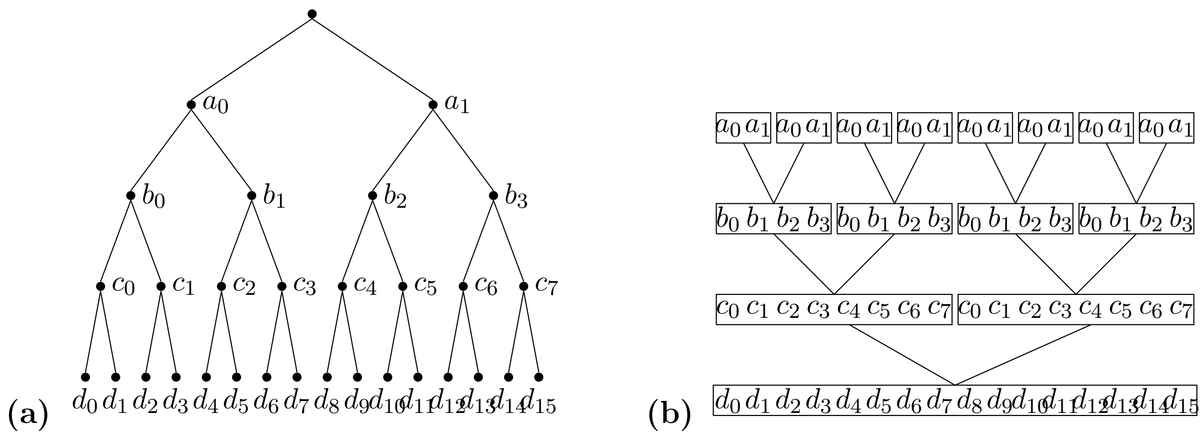


Figure 7-2: (a) The marked ancestor problem. (b) An instance of fully-persistent marked ancestor.

Proof. Our inputs to the fully-persistent problem have the pattern illustrated in Figure 7-2. At the root of the version tree, we have update operations for the leaves of the marked ancestor tree. If we desire a lower bound for the incremental marked ancestor problems, all nodes start unmarked, and we have an update for every leaf that needs to be marked. If we want a decremental lower bound, all nodes start marked, and all operations are *unmark*.

The root has b subversions; in each subversion, the level above the leaves in the marked ancestor tree is updates. The construction continues similarly, branching our more versions at the rate at which level size decreases. Thus, on each level of the version tree we have b^d updates, giving $b^d \cdot d$ updates in total.

With this construction of the updates, the structure of the fully persistent marked ancestor problem is isomorphic to a butterfly. Imagine what happens when we query a leaf v of the marked ancestor tree, at a leaf t of the version tree. We think of both v and t as vectors in $[b]^d$, spelling out the root to leaf paths. The path from the root to v goes through every level of the version tree:

- on the top level, there is a single version (t is irrelevant), in which v is updated.
- on the next level, the subversion we descend to is decided by the first coordinate of t. In this subversion, v's parent is updated. Note that v's parent is determined by the first $d-1$ coordinates of v.
- on the next level, the relevant subversion is dictated by the first two coordinates of t. In this subversion, v's grandparent is updated, which depends on the first $d-2$ coordinates of v.
- etc.

This is precisely the definition of a source-to-sink path in the butterfly graph, morphing the source into the sink one coordinate at a time. Each update will mark a node if the corresponding edge in the butterfly is missing in the subgraph. Thus, we encounter a marked ancestor iff some edge is missing. $\square$

Let us see how Reduction 7.7 combines with Lemma 7.6 to give an actual lower bound. Given a butterfly graph with m edges, we generate at most m updates. From Lemma 7.6, the space of the fully persistent structure is $S = O(m \cdot t_u)$, and the query time $O(t_q \lg\lg(mt_q))$, where t_u and t_q are the assumed running times for dynamic marked ancestor. If $t_u = O(\operatorname{polylog} m)$, the space is $S = O(m \operatorname{polylog} m)$.

The lower bound for reachability oracles from Theorem 7.1 implies that for space $O(m \operatorname{polylog} m)$, the query time must be $\Omega\left(\frac{\lg m}{\lg\lg m}\right)$. But we have an upper bound of $O(t_q \lg\lg(mt_q))$ for the query time, so $t_q = \Omega\left(\frac{\lg m}{\lg^2 \lg m}\right)$. This is weaker by a $\lg\lg m$ factor compared to the original bound of [8].

7.2 Adding Structure to Set Disjointness

Remember that in the lopsided set disjointness (LSD) problem, Alice and Bob receive sets S and T and must determine whether $S \cap T = \emptyset$. Denote the size of Alice's set by $|S| = N$, and let B be the fraction between the universe and N. In other words, $S, T \subseteq [N \cdot B]$. Note that $|T|$ may be as large as $N \cdot B$.

In Chapter 5, we showed the following deterministic lower bound for LSD:

Theorem 7.8. *Fix $\delta > 0$. In a deterministic protocol for* LSD*, either Alice sends $\Omega(N \lg B)$ bits, or Bob sends $NB^{1-\delta}$ bits.*

Just as it is more convenient to work with 3SAT that circuit-SAT for showing NP-completeness, our reductions uses a more restricted version of LSD, which we call 2-Blocked-LSD. In this problem, the universe is interpreted as $[\frac{N}{B}] \times [B] \times [B]$. It is guaranteed that for all $x \in [\frac{N}{B}]$ and $y \in [B]$, S contains a single element of the form $(x, y, \star)$ and a single element of the form $(x, \star, y)$.

It is possible (and in fact easy) to reanalyze the lower bounds of Chapter 5, and show that they also apply to 2-Blocked-LSD. However, in the spirit of this chapter, we choose to design a reduction from general LSD to this special case. We show that LSD is reducible to 2-Blocked-LSD with communication complexity essentially $O(N)$. Since the lower bound is $\omega(N)$, it must also hold for 2-Blocked-LSD.

Lemma 7.9. LSD *reduces to* 2-Blocked-LSD *by a deterministic protocol with communication complexity $O(N + \lg\lg B)$.*

Proof. Consider a random permutation π of the universe. If Alice and Bob can agree on a permutation, they can apply it on their own inputs and solve LSD on $\pi(S)$ and $\pi(T)$.

In 2-Blocked-LSD, there are $(B!)^{N/B}$ valid instances of S. Since $\pi(S)$ is a uniformly random N-subset of the universe, the probability that it generates an instance of 2-Blocked-LSD is:

$$(B!)^{N/B} / \binom{NB}{N} \;\ge\; \left(\frac{B}{e}\right)^{B\cdot(N/B)} / \left(\frac{eNB}{N}\right)^{N} \;=\; \left(\frac{B}{e}\right)^{N} / (B \cdot e)^{N} \;=\; e^{-2N}$$

We are looking for a small set $\mathcal{F}$ of permutations, such that for any input S, there exists $\pi \in \mathcal{F}$ for which $\pi(S)$ is an instance of 2-Blocked-LSD. Then, Alice and Bob can agree on $\mathcal{F}$ in advance. When seeing an instance S of LSD, Alice chooses a π that maps it to an instance of 2-Blocked-LSD, and sends the index of π in $\mathcal{F}$ to Bob, using $\lg |\mathcal{F}|$ bits.

By the probabilistic method, we show there exist $\mathcal{F}$ of size $e^{2N} \cdot 2N \lg B$, i.e. $\lg |\mathcal{F}| = O(N)$. Choose every element of $\mathcal{F}$ randomly. For a fixed S, the probability that no element is good is at most $(1 - e^{-2N})^{|\mathcal{F}|} \leq \exp(-|\mathcal{F}|/e^{-2N}) = e^{-2N \lg B}$. But there are only $\binom{NB}{N} \leq (eB)^N < e^{2N \lg B}$ choices of S, so by a union bound $\mathcal{F}$ works for all possible S with nonzero probability. □

Finally, we must clarify the notion of reduction from a communication problem to a data-structure problem. In such a reduction, Bob constructs a database based on his set T, and Alice constructs a set of k queries. It is then shown that LSD can be solved based on the answer to the k queries on Bob's database. If we are interested in lower bounds for space $n^{1+o(1)}$, we must reduce to a large number k of queries. In each cell probe, the queries want to read some k cells from the memory of size S. Then, Alice can send $\lg \binom{S}{k}$ bits, and Bob can reply with $k \cdot w$. Observe that $\lg \binom{S}{k} = \Theta(k \lg \frac{S}{k}) \ll k \lg S$, if k is large enough.

7.2.1 Randomized Bounds

In Chapter 5, we also showed the following randomized lower bound for LSD:

Theorem 7.10. *Assume Alice receives a set $S, |S| = n$ and Bob receives a set $T, |T| = m$, both sets coming from a universe of size $2nm$, for $m < n^\gamma$, where $\gamma < 1$ is a constant. In any randomized, two-sided error communication protocol deciding disjointness of S and T, either Alice sends $\Omega(m \lg n)$ bits or Bob sends $\Omega(n^{1-\delta})$ bits, for any $\delta > 0$.*

This bound will not be enough for our reduction to reachability oracles in butterfly graphs, because the universe is quadratic in the set sizes. We can replace this lower bound by an optimal randomized lower bound for LSD which only needs a linear universe.

However, we can also give a simple self-contained proof of a randomized lower bound without requiring this more complicated result. This alternative proof works by reducing from $\bigoplus^k$ LSD, a direct sum of randomized LSD problems. All this is needed is the direct sum result for randomized richness, which we proved in §5.4.3.

7.3 Set Disjointness to Reachability Oracles

Since we want a lower bound for near-linear space, we must reduce LSD to k parallel queries on the reachability oracle. The entire action is in what value of k we can achieve. Note, for instance, that $k = N$ is trivial, because Alice can pose a query for each item in her set. However, a reduction with $k = N$ is also useless. Remember that the communication complexity of Alice is $t \cdot \lg \binom{S}{k} \geq t \lg \binom{NB}{N}$. But LSD is trivially solvable with communication

$\lg \binom{NB}{N}$, since Alice can communicate her entire set. Thus, there is no contradiction with the lower bound.

To get a lower bound on t, k must be made as small as possible compared to N. Intuitively, a source–sink path in a butterfly of depth d traverses d edges, so it should be possible to test d elements by a single query. To do that, the edges must assemble in contiguous source–sink paths, which turns out to be possible if we carefully match the structure of the butterfly and the 2-Blocked-LSD problem:

Reduction 7.11. *Let G be a degree-B butterfly graph with N non-sink vertices and $N \cdot B$ edges, and let d be its depth.* 2-Blocked-LSD *reduces to $\frac{N}{d}$ parallel queries to a reachability oracle for a subgraph of G.*

Proof. Remember that in 2-Blocked-LSD, elements are triples (x, y, z) from the universe $[\frac{N}{B}] \times [B] \times [B]$. We define below a bijection between $[\frac{N}{B}] \times [B]$ and the non-sink vertices of G. Since (x, y) is mapped to a non-sink vertex, it is natural to associate (x, y, z) to an edge, specifically edge number z going out of vertex (x, y).

Bob constructs a reachability oracle for the graph G excluding the edges in his set T. Then, Alice must find out whether any edge from her set S has been deleted. By mapping the universe $[\frac{N}{B}] \times [B]$ to the nodes carefully, we will ensure that Alice's edges on each level form a perfect matching. Then, her set of N edges form $\frac{N}{d}$ disjoint paths from sources to sinks. Using this property, Alice can just issue $\frac{N}{d}$ queries for these paths. If any of the source–sink pairs is unreachable, some edge in S has been deleted.

To ensure Alice's edges form perfect matchings at each level, we first decompose the non-sink vertices of G into $\frac{N}{B}$ microsets of B elements each. Each microset is associated to some level i, and contains nodes of the form $(\cdots, v_{i-1}, \star, v_{i+1}, \cdot)$ on level i. A value (x, y) is mapped to node number y in a microset identified by x (through some arbitrary bijection between $[\frac{N}{B}]$ and microsets).

Let $(x, 1, z_1), \ldots, (x, B, z_B)$ be the values in S that give edges going out of microset x. If the nodes of the microset are the vectors $(\cdots, v_{i-1}, \star, v_{i+1}, \cdot)$, the nodes to which the edges of S go are the vectors $(\cdots, v_{i-1}, z_j, v_{i+1}, \cdot)$ on the next level, where $j \in [B]$. Observe that edges from different microsets cannot go to the same vertex. Also, edges from the same microset go to distinct vertices by the 2-Blocked property: for any fixed x, the z_j's are distinct. Since all edges on a level point to distinct vertices, they form a perfect matching. $\square$

Let us now compute the lower bounds implied by the reduction. We obtain a protocol for 2-Blocked-LSD in which Alice communicates $t \lg \binom{S}{k} = O(tk \lg \frac{S}{k}) = O(N \cdot \frac{t}{d} \lg \frac{Sd}{N})$ bits, and Bob communicates $k \cdot t \cdot w = O(N \cdot \frac{t}{d} \cdot w)$ bits. On the other hand, the lower bound for 2-Blocked-LSD says that Alice needs to communicate $\Omega(N \lg B)$ bits, or Bob needs to communicate $NB^{1-\delta}$, for any constant $\delta > 0$. It suffices to use, for instance, $\delta = \frac{1}{2}$.

Comparing the lower bounds with the reduction upper bound, we conclude that either $\frac{t}{d} \lg \frac{Sd}{N} = \Omega(\lg B)$, or $\frac{t}{d} w = \Omega(\sqrt{B})$. Set the degree of the butterfly to satisfy $B \geq w^2$ and $\lg B \geq \lg \frac{Sd}{N}$. Then, $\frac{t}{d} = \Omega(1)$, i.e. $t = \Omega(d)$. This is intuitive: it shows that the query needs to be as slow as the depth, essentially traversing a source to sink path.

Finally, note that the depth is $d = \Theta(\log_B N)$. Since $\lg B \geq \max\{2\lg w, \lg\frac{Sd}{N}\} = \Omega\big(\lg w + \lg\frac{Sd}{N}\big) = \Omega\big(\lg\frac{Sdw}{N}\big)$. Note that certainly $d < w$, so $\lg B = \Omega(\lg\frac{Sw}{N})$. We obtain $t = \Omega(d) = \Omega(\lg N / \lg\frac{Sw}{N})$.

Chapter 8

Near Neighbor Search in ℓ_∞

In this chapter, we deal with near neighbor search (NNS) under the distance $d(p, q) = \|p - q\|_\infty = \max_{i \in [d]} |p_i - q_i|$, called the ℓ_∞ norm. See §2.4.3 for background on near-neighbor search, and, in particular, for a discussion of this important metric.

The structure of the ℓ_∞ space is intriguingly different, and more mysterious than other natural spaces, such as the ℓ_1 and ℓ_2 norms. In fact, there is precisely one data structure for NNS under ℓ_∞ with provable guarantees. In FOCS'98, Indyk [61] described an NNS algorithm for d-dimensional ℓ_∞ with approximation $4\lceil \log_\rho \log_2 4d \rceil + 1$, which required space $dn^\rho \lg^{O(1)} n$ and query time $d \cdot \lg^{O(1)} n$, for any $\rho > 1$. For 3-approximation, Indyk also gives a solution with storage $O(n^{\log_2 d + 1})$. Note that in the regime of polynomial space, the algorithm achieves an uncommon approximation factor of $O(\lg \lg d)$.

In this chapter, we begin by describing Indyk's data structure in a manner that is conceptually different from the original description. Our view relies on an information-theoretic understanding of the algorithm, which we feel explains its behavior much more clearly.

Inspired by this understanding, we are able to prove a lower bound for the asymmetric communication complexity of c-approximate NNS in ℓ_∞:

Theorem 8.1. *Assume Alice holds a point $q \in \{0, \dots, m\}^d$, and Bob holds a database $D \subset \{-m, \dots m\}^d$ of n points. They communicate via a deterministic protocol to output:*
"1" *if there exists some $p \in D$ such that $\|q - p\|_\infty \le 1$;*
"0" *if, for all $p \in D$, we have $\|q - p\|_\infty \ge c$.*

Fix $\delta, \varepsilon > 0$; assume the dimension d satisfies $\Omega(\lg^{1+\varepsilon} n) \le d \le o(n)$, and the approximation ratio satisfies $3 < c \le O(\lg \lg d)$. Further define $\rho = \frac{1}{2}(\frac{\varepsilon}{2} \lg d)^{1/c} > 10$.

Then, either Alice sends $\Omega(\delta \rho \lg n)$ bits, or Bob sends $\Omega(n^{1-\delta})$ bits.

Note that this result is tight in the communication model, suggesting the Indyk's unusual approximation is in fact inherent to NNS in ℓ_∞. As explained in Chapter 6, this lower bound on asymmetric communication complexity immediately implies the following corollaries for data structures:

Corollary 8.2. *Let $\delta > 0$ be constant, and assume $\Omega(\lg^{1+\delta} n) \le d \le o(n)$. Consider any cell-probe data structure solving d-dimensional NNS under ℓ_∞ with approximation $c =$*

$O(\log_\rho \log_2 d)$. *If the word size is* $w = n^{1-\delta}$ *and the query complexity is* t*, the data structure requires space* $n^{\Omega(\rho/t)}$.

Corollary 8.3. *Let* $\delta > 0$ *be constant, and assume* $\Omega(\lg^{1+\delta} n) \le d \le o(n)$. *A decision tree of depth* $n^{1-2\delta}$ *with predicate size* n^δ *that solves* d*-dimensional near-neighbor search under* ℓ_∞ *with approximation* $c = O(\log_\rho \log_2 d)$*, must have size* $n^{\Omega(\rho)}$.

As with all known lower bounds for large space, Corollary 8.2 is primarily interesting for constant query time, and degrades exponentially with t. On the other hand, the lower bound for decision trees holds even for extremely high running time (depth) of $n^{1-\delta}$. A decision tree with depth n and predicate size $O(d \lg M)$ is trivial: simply test all database points.

Indyk's result is a deterministic decision tree with depth $O(d \cdot \operatorname{poly}\log n)$ and predicate size $O(\lg d + \lg M)$. Thus, we show an optimal trade-off between space and approximation, at least in the decision tree model. In particular, for polynomial space, the approximation factor of $\Theta(\lg \lg d)$ is intrinsic to NNS under ℓ_∞.

8.1 Review of Indyk's Upper Bound

Decision trees. Due to the decomposability of ℓ_∞ as a maximum over coordinates, a natural idea is to solve NNS by a decision tree in which every node is a coordinate comparison. A node v is reached for some set $Q_v \subseteq \mathbb{Z}^d$ of queries. If the node compares coordinate $i \in [d]$ with a "separator" x, its two children will be reached for queries in $Q_\ell = Q_v \cap \{q \mid q_i < x\}$, respectively in $Q_r = Q_v \cap \{q \mid q_i > x\}$ (assume x is non-integral to avoid ties).

Define $[x, y]_i = \big\{p \mid p_i \in [x, y]\big\}$. Then, $Q_\ell = Q_v \cap [-\infty, x]_i$ and $Q_r = Q_v \cap [x, \infty]_i$.

If the query is known to lie in some Q_v, the set of database points that could still be a near neighbor is $N_v = D \cap \big(Q_v + [-1, 1]^d\big)$, i.e. the points inside the Minkowski sum of the query set with the ℓ_∞ "ball" of radius one. For our example node comparing coordinate $i \in [d]$ with x, the children nodes have $N_\ell = N_v \cap [-\infty, x + 1]_i$, respectively $N_r = N_v \cap [x - 1, +\infty]_i$.

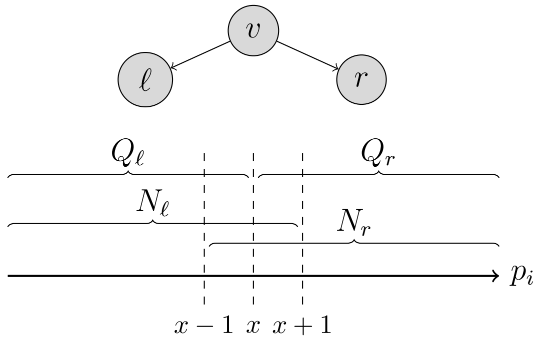


Figure 8-1: A separator x on coordinate i.

Observe that $N_\ell \cap N_r = N_v \cap [x - 1, x + 1]_i$. In some sense, the database points in this slab are being "replicated," since both the left and right subtrees must consider them as potential near neighbors. This recursive replication of database points is the cause of superlinear space. The contribution of Indyk [61] is an intriguing scheme for choosing a separator that guarantees a good bound on this recursive growth.

Information progress. Our first goal is to get a handle on the growth of the decision tree, as database points are replicated recursively. Imagine, for now, that queries come from some distribution μ. The reader who enjoys worst-case algorithms need not worry: μ is just an analysis gimmick, and the algorithm will be deterministic.

We can easily bound the tree size in terms of the measure of the smallest Q_v ever reached: there can be at most $1/\min_v \Pr_\mu[Q_v]$ distinct leaves in the decision tree, since different leaves are reached for disjoint Q_v's. Let $I_Q(v) = \log_2 \frac{1}{\Pr_\mu[Q_v]}$; this can be understood as the information learned about the query, when computation reaches node v. We can now rewrite the space bound as $O\big(2^{\max_v I_Q(v)}\big)$.

Another quantity that can track the behavior of the decision tree is $H_N(v) = \log_2 |N_v|$. Essentially, this is the "entropy" of the identity of the near neighbor, assuming that all database points are equally likely neighbors.

At the root λ, we have $I_Q(\lambda) = 0$ and $H_N(\lambda) = \lg n$. Decision nodes must reduce the entropy of the near neighbor until H_N reaches zero ($|N_v| = 1$). Then, the algorithm can simply read the single remaining candidate, and test whether it is a near neighbor of the query. Unfortunately, decision nodes also increase I_Q along the way, increasing the space bound. The key to the algorithm is to balance this tension between reducing the entropy of the answer, H_D, and not increasing the information about the query, I_Q, too much.

In this information-theoretic view, Indyk's algorithm shows that we can (essentially) always find a separator that decreases H_N by some δ but does not increase I_Q by more than $\rho \cdot \delta$. Thus, H_D can be pushed from $\lg n$ down to 0, without ever increasing I_Q by more than $\rho \lg n$. That is, space $O(n^\rho)$ is achieved.

Searching for separators. At the root λ, we let $i \in [d]$ be an arbitrary coordinate, and search for a good separator x on that coordinate. Let π be the frequency distribution (the empirical probability distribution) of the projection on coordinate i of all points in the database. To simplify expressions, let $\pi(x : y) = \sum_{j=x}^{y} \pi(j)$.

If x is chosen as a separator at the root, the entropy of the near neighbor in the two child nodes is reduced by:

$$\begin{aligned} H_N(\lambda) - H_N(\ell) &= \log_2 \frac{|N_\lambda|}{|N_\ell|} = \log_2 \frac{|D|}{|D \cap [-\infty, x+1]_i|} = \log_2 \frac{1}{\pi(-\infty : x+1)} \\ H_N(\lambda) - H_N(r) &= \log_2 \frac{1}{\pi(x-1 : \infty)} \end{aligned}$$

Remember that we have not yet defined μ, the assumed probability distribution on the query. From the point of view of the root, it only matters what probability μ assigns to Q_ℓ and Q_r. Let us reason, heuristically, about what assignments are needed for these probabilities in order to generate difficult problem instances. If we understand the most difficult instance, we can use that setting of probabilities to obtain an upper bound for all instances.

First, it seems that in a hard instance, the query needs to be close to some database point (at least with decent probability). Let us simply assume that the query is always *planted* in the neighborhood of a database point; the problem remains to find this near neighbor.

Assume by symmetry that $H_N(\ell) \geq H_N(r)$, i.e. the right side is smaller. Under our heuristic assumption that the query is planted next to a random database point, we can lower bound $\Pr_\mu[Q_r] \ge \pi(x+1,\infty)$. Indeed, whenever the query is planted next to a point in $[x+1,\infty]_i$, it cannot escape from $Q_r = [x,\infty]_i$. Remember that our space guarantee blows up when the information about Q_v increases quickly (i.e. the probability of Q_v decreases). Thus, the worst case seems to be when $\Pr_\mu[Q_r]$ is as low as possible, namely equal to the lower bound.

To summarize, we have convinced ourselves that it's reasonable to define μ such that:

$$\Pr_\mu[Q_\ell] = \pi(-\infty : x+1); \qquad \Pr_\mu[Q_r] = \pi(x+1,\infty) \tag{8.1}$$

We apply similar conditions at all nodes of the decision tree. Note that there exists a μ satisfying all these conditions: the space of queries is partitioned recursively between the left and right subtrees, so defining the probability of the left and right subspace at all nodes is an (incomplete) definition of μ.

From (8.1), we can compute the information revealed about the query:

$$\begin{aligned} I_Q(\ell) - I_Q(\lambda) &= \log_2 \frac{\Pr[Q_\lambda]}{\Pr[Q_\ell]} = \log_2 \frac{1}{\pi(-\infty : x+1)} \\ I_Q(r) - I_Q(\lambda) &= \log_2 \frac{1}{\pi(x+1:\infty)} \end{aligned}$$

Remember that our rule for a good separator was "$\Delta I_Q \le \rho \cdot \Delta H_N$." On the left side, $I_Q(\ell) - I_Q(\lambda) = H_N(\lambda) - H_N(\ell)$, so the rule is trivially satisfied. On the right, the rule asks that: $\log_2 \frac{1}{\pi(x+1:\infty)} \le \rho \cdot \log_2 \frac{1}{\pi(x-1:\infty)}$. Thus, x is a good separator iff $\pi(x+1:\infty) \ge \big[\pi(x-1:\infty)\big]^\rho$.

Finale. As defined above, any good separator satisfies the bound on the information progress, and guarantees the desired space bound of $O(n^\rho)$. We now ask what happens when no good separator exists.

We may assume by translation that the median of π is 0, so $\pi([1:\infty]) \le \frac{1}{2}$. If $x = 1\frac{1}{2}$ is not a good separator, it means that $\pi(3:\infty) < \big[\pi(1:\infty)\big]^\rho \le 2^{-\rho}$. If $x = 3\frac{1}{2}$ is not a good separator, then $\pi(5:\infty) < \big[\pi(3:\infty)\big]^\rho \le 2^{-\rho^2}$. By induction, the lack of a good separator implies that $\pi(2j+1:\infty) < 2^{-\rho^j}$. The reasoning works symmetrically to negative values, so $\pi(-\infty : -2j-1) < 2^{-\rho^j}$.

Thus, if no good separator exists on coordinate i, the distribution of the values on that coordinate is very concentrated around the median. In particular, only a fraction of $\frac{1}{2d}$ of the database points can have $|x_i| > R = 2\log_\rho \log_2 4d$. Since there is no good separator on any coordinate, it follows that less than $d \cdot \frac{n}{2d} = \frac{n}{2}$ points have *some* coordinate exceeding R. Let $D^\star$ be the set of such database points.

To handle the case when no good separator exists, we can introduce a different type of node in the decision tree. This node tests whether the query lies in an ℓ_∞ ball of radius $R+1$ (which is equivalent to d coordinate comparisons). If it does, the decision tree simply outputs any point in $D \setminus D^\star$. Such a point must be within distance $2R+1$ of the query, so it is an $O(\log_\rho \log d)$ approximation.

If the query is outside the ball of radius $R+1$, a near neighbor must be outside the ball of radius R, i.e. must be in $D^\star$. We continue with the recursive construction of a decision tree for point set $D^\star$. Since $|D^\star| \le |D|/2$, we get a one-bit reduction in the entropy of the answer for free. (Formally, our μ just assigns probability one to the query being outside the ball of radius $R+1$, because in the "inside" case the query algorithm terminates immediately.)

8.2 Lower Bound

Armed with this information-theoretic understanding of Indyk's algorithm, the path to a lower bound is more intuitive. We will define a distribution on coordinates decaying roughly like $2^{-\rho^x}$, since we know that more probability in the tail gives the algorithm an advantage. Database points will be independent and identically distributed, with each coordinate drawn independently from this distribution.

In the communication view, Alice's message sends a certain amount of information restricting the query space to some Q. The entropy of the answer is given by the measure of $N(Q) = Q + [-1,1]^d$, since the expected number of points in this space is just $n \cdot \Pr[N(Q)]$. The question that must be answered is: fixing $\Pr[Q]$, how small can $\Pr[N(Q)]$ be?

We will show an isoperimetric inequality proving that the least expanding sets are exactly the ones generated by Indyk's algorithm: intersections of coordinate cuts $[x,\infty]_i$. Note that $\Pr\big[[x,\infty]_i\big] \approx 2^{-\rho^x}$, and $N\big([x,\infty]_i\big) = [x-1,\infty]_i$. Thus, the set expands to measure $\Pr\big[x-1,\infty]_i\big] \approx 2^{-\rho^{x-1}} \approx \Pr\big[[x,\infty]_i\big]^{1/\rho}$. Our isoperimetric inequality will show that for any Q, its neighborhood has measure $\Pr[N(Q)] \ge \Pr[Q]^{1/\rho}$.

Then, if Alice's message has $o(\rho \lg n)$ bits of information, the entropy of the near neighbor decreases by only $o(\lg n)$ bits. In other words, $n^{1-o(1)}$ of the points are still candidate near neighbors, and we can use this to lower bound the message that Bob must send.

The crux of the lower bound is not the analysis of the communication protocol (which is standard), but proving the isoperimetric inequality. Of course, the key to the isopermitetric inequality is the initial conceptual step of defining an appropriate biased distribution, in which the right inequality is possible. The proof is rather non-standard for an isoperimetric inequality, because we are dealing with a very particular measure on a very particular space. Fortunately, a few mathematical tricks save it from being too technical.

Formal details. We denote the communication problem, c-approximate NNS, by the partial function F. Let the domain of $\bar{F}$ be $X \times Y$, where $X = \{0,1,\dots m\}^d$ and $Y = \big(\{0,1,\dots m\}^d\big)^n$. Complete the function F by setting $\bar{F}(q,D) = F(q,D)$ whenever $F(q,D)$ is defined (in the "0" or "1" instances), and $\bar{F}(q,D) = \star$ otherwise.

As explained already, our lower bound only applies to deterministic protocols, but it requires conceptual use of distributions on the input domains X and Y. First define a measure π over the set $\{0,1,\dots m\}$: for $i \ge 1$, let $\pi(\{i\}) = \pi_i = 2^{-(2\rho)^i}$; and let $\pi_0 = 1 - \sum_{i\ge1} \pi_i \ge \frac{1}{2}$. Here ρ is a parameter to be determined.

Now, define a measure μ over points by generating each coordinate independently according to π: $\mu(x_1, x_2, \dots, x_d)\}) = \pi_{x_1} \cdots \pi_{x_d}$. Finally, define a measure η over the database by generating each point independently according to μ.

First, we show that $\bar{F}$ is zero with probability $\Omega(1)$:

Claim 8.4. *If $d \geq \lg^{1+\varepsilon} n$ and $\rho \leq \frac{1}{2}(\frac{\varepsilon}{2}\lg d)^{1/c}$, then $\Pr_{q \leftarrow \mu, D \leftarrow \eta}[\bar{F}(q, D) \neq 0] \leq \frac{1}{2}$.*

Proof. Consider q and some $p \in D$: their jth coordinates differ by c or more with probability at least $2\pi_0\pi_c \geq \pi_c$. Thus,

$$\Pr[\|q-p\|_\infty < c] \leq (1-\pi_c)^d \leq e^{-\pi_c d} \leq e^{-2^{-(\varepsilon/2)\lg d} \cdot d} \leq e^{-d^{1-\varepsilon/2}} \leq e^{-(\lg n)^{(1+\varepsilon)(1-\varepsilon/2)}} \leq e^{-(\lg n)^{1+\varepsilon/4}}$$

By a union bound over all $p \in D$, we get that q has no neighbor at distance less than c with probability at least $1 - n \cdot \exp(-(\lg n)^{1+\varepsilon/4}) = 1 - o(1)$. ☐

Claim 8.5. *If Alice sends a bits and Bob sends b bits, there exists a combinatorial rectangle $\mathcal{Q} \times \mathcal{D} \subseteq X \times Y$ of measure $\mu(\mathcal{Q}) \geq 2^{-O(a)}$ and $\eta(\mathcal{D}) \geq 2^{-O(a+b)}$, on which $\bar{F}$ only takes values in $\{0, \star\}$.*

Proof. This is just the deterministic richness lemma (Lemma 5.4) in disguise. Let $F' : X \times Y \to \{0, 1\}$ be the output of the protocol. We have $F'(q, D) = \bar{F}(q, D)$ whenever $\bar{F}(q, D) \neq \star$. Since $\Pr[F'(q, D) = 0] \geq \frac{1}{2}$, F' is rich: half of the columns are at least half zero (in the weighted sense). Thus, we can find a rectangle of size $\mu(\mathcal{Q}) \geq 2^{-O(a)}$ and $\eta(\mathcal{D}) \geq 2^{-O(a+b)}$, on which F' is identically zero. Since the protocol is always correct, this means that $\bar{F}$ is 0 or $\star$ on the rectangle. ☐

To obtain a lower bound, we show that any big rectangle must contain at least a value of "1." This will follow from the following isoperimetric inequality in our measure space, shown in §8.3:

Theorem 8.6. *Consider any set $S \subseteq \{0, 1, \dots m\}^d$, and let $N(S)$ be the set of points at distance at most 1 from S under ℓ_∞: $N(S) = \{p \mid \exists s \in S : \|p - s\|_\infty \leq 1\}$. Then, $\mu(N(S)) \geq \big(\mu(S)\big)^{1/\rho}$.*

Claim 8.7. *Consider any rectangle $\mathcal{Q} \times \mathcal{D} \subseteq X \times Y$ of size $\mu(\mathcal{Q}) \geq 2^{-\delta\rho \lg n}$ and $\eta(\mathcal{D}) \geq 2^{-O(n^{1-\delta})}$. Then, there exists some $(q, D) \in \mathcal{Q} \times \mathcal{D}$ such that $\bar{F}(q, D) = 1$.*

Proof. By isoperimetry, $\mu(N(\mathcal{Q})) \geq \big(\mu_d(\mathcal{Q})\big)^{1/\rho} \geq 1/n^\delta$. All we need to show is that there exists a set $D \in \mathcal{D}$ that intersects with $N(\mathcal{Q})$.

For $D \in Y$, let $\sigma(D) = |D \cap N(\mathcal{Q})|$. The proof uses a standard concentration trick on σ. Suppose D is chosen randomly according to η, i.e. not restricted to $\mathcal{D}$. Then $\mathbf{E}[\sigma(D)] = n \cdot \Pr_\mu[N(\mathcal{Q})] \geq n^{1-\delta}$. Furthermore, $\sigma(D)$ is tightly concentrated around this mean, by the Chernoff bound. In particular, $\Pr[\sigma(D) = 0] \leq e^{-\Omega(n^{1-\delta})}$.

This probability is so small, that it remains small even is we restrict to $\mathcal{D}$. We have $\Pr[\sigma(D) = 0 \mid D \in \mathcal{D}] \leq \frac{\Pr[\sigma(D)=0]}{\Pr[D \in \mathcal{D}]} \leq e^{-\Omega(n^{1-\delta})}/\eta(\mathcal{D})$. Thus, if $\eta(\mathcal{D}) \geq 2^{-\gamma \cdot n^{1-\delta}}$ for some small enough constant γ, we have $\Pr[\sigma(D) = 0 \mid D \in \mathcal{D}] = o(1)$. In other words, there exists some $D \in \mathcal{D}$ such that $N(\mathcal{Q}) \cap D \neq \emptyset$, and thus, there exists an instance in the rectangle on which $\bar{F} = 1$. $\square$

Combining Claims 8.5 and 8.7, we immediately conclude that either Alice sends $a = \Omega(\delta\rho \lg n)$ bits or Bob sends $b = \Omega(n^{1-\delta})$ bits. This concludes the proof of Theorem 8.1.

8.3 An Isoperimetric Inequality

This section proves the inequality of Theorem 8.6: for any S, $\mu(N(S)) \geq \big(\mu(S)\big)^{1/\rho}$. As with most isoperimetric inequalities, the proof is by induction on the dimensions. In our case, the inductive step is provided by the following inequality, whose proof is deferred to §8.4:

Lemma 8.8. *Let $\rho \geq 10$ be an integer, and define $\pi_i = 2^{-(2\rho)^i}$ for all $i \in \{1, \ldots, m\}$, and $\pi_0 = 1 - \sum_{i=1}^m \pi_i$. For any $\beta_0, \ldots, \beta_m \in \mathbb{R}_+$ satisfying $\sum_{i=0}^m \pi_i \beta_i^\rho = 1$, the following inequality holds (where we interpret β_{-1} and β_{m+1} as zero):*

$$\sum_{i=0}^{m} \pi_i \cdot \max\{\beta_{i-1},\ \beta_i,\ \beta_{i+1}\} \ \geq\ 1 \tag{8.2}$$

To proceed to our inductive proof, let μ_d be the d-dimensional variant of our distribution. The base case is $d = 0$. This space has exactly one point, and $\mu_0(S)$ is either 0 or 1. We have $N(S) = S$, so $\mu_0(N(S)) = \mu_0(S) = \big(\mu_0(S)\big)^{1/\rho}$.

Now consider the induction step for $d-1$ to d dimensions. Given a set $S \subset \{0, 1, \ldots m\}^d$, let $S_{[i]}$ be the set of points in S whose first coordinate is i, i.e. $S_{[i]} = \{(s_2, \ldots, s_d) \mid (i, s_2, \ldots, s_d) \in S\}$. Define:

$$\beta_i = \left(\frac{\mu_{d-1}(S_{[i]})}{\mu_d(S)}\right)^{1/\rho} \quad \Rightarrow \quad \sum_{i=0}^{m} \pi_i \beta_i^\rho \ =\ \sum_{i=0}^{m} \pi_i \cdot \frac{\mu_{d-1}(S_{[i]})}{\mu_d(S)} \ =\ \frac{1}{\mu_d(S)} \sum_{i=0}^{m} \pi_i \mu_{d-1}(S_{[i]}) \ =\ 1$$

We have $N(S)_{[i]} = N(S_{[i-1]}) \ \cup\ N(S_{[i]}) \ \cup\ N(S_{[i+1]})$. Thus, we can lower bound:

$$\mu_d(N(S)) \ =\ \sum_{i=0}^{m} \pi_i \cdot \mu_{d-1}\big(N(S)_{[i]}\big) \ \geq\ \sum_{i=0}^{m} \pi_i \cdot \max\big\{\mu_{d-1}(N(S_{[i-1]})), \mu_{d-1}(N(S_{[i]})), \mu_{d-1}(N(S_{[i+1]}))\big\}$$

But the inductive hypothesis assures us that $\mu_{d-1}(N(S_{[i]})) \ \geq\ \big(\mu_{d-1}(S_{[i]})\big)^{1/\rho} \ =\ \beta_i \cdot \big(\mu_d(S)\big)^{1/\rho}$. Thus:

$$\mu_d(N(S)) \ \geq\ \big(\mu_d(S)\big)^{1/\rho} \cdot \sum_{i=0}^{m} \pi_i \cdot \max\big\{\beta_{i-1}, \beta_i, \beta_{i+1}\big\} \ \geq\ \big(\mu_d(S)\big)^{1/\rho},$$

where we have used inequality (8.2) in the last step. This finishes the proof of Theorem 8.6.

8.4 Expansion in One Dimension

Let $\Gamma = \left\{(\beta_0, \ldots, \beta_m) \in \mathbb{R}^{m+1} \mid \sum_{i=0}^{m} \pi_i \beta_i^{\rho} = 1\right\}$, and denote by $f\left(\beta_0, \ldots, \beta_m\right)$ the left hand side of (8.2). Since f is a continuous function on the compact set $\Gamma \subset \mathbb{R}^{m+1}$, it achieves its minimum. Call an $(m+1)$-tuple $(\beta_0, \ldots, \beta_m) \in \Gamma$ *optimal* if $f\left(\beta_0, \ldots, \beta_m\right)$ is minimal. Our proof strategy will be to show that if $(\beta_0, \ldots, \beta_m)$ is optimal, then $\beta_i = 1$.

We consider several possible configurations for sizes of β_i's in an optimal β, and rule them out in three separate lemmas. We then prove the inequality by showing that these configurations are all the configurations that we need to consider.

Lemma 8.9. *If there exists an index $i \in \{1, \ldots, m-1\}$ such that $\beta_{i-1} > \beta_i < \beta_{i+1}$, then $\bar{\beta} = (\beta_0, \ldots, \beta_m)$ is not optimal.*

Proof. Define a new vector $\bar{\beta}' = (\beta_0, \ldots, \beta_{i-2}, \beta_{i-1} - \epsilon, \beta_i + \delta, \beta_{i+1} - \epsilon, \beta_{i+2}, \ldots, \beta_m)$, where $\epsilon,\ \delta > 0$ are chosen suitably so that $\bar{\beta}' \in \Gamma$, and $\beta_{i-1} - \epsilon > \beta_i + \delta < \beta_{i+1} - \epsilon$. It's easy to see that $f\left(\bar{\beta}\right) > f\left(\bar{\beta}'\right)$, which contradicts the optimality of $\bar{\beta}$. $\square$

Lemma 8.10. *If there exists an index $i \in \{1, \ldots, m\}$ such that $\beta_{i-1} > \beta_i \geq \beta_{i+1}$, then $\bar{\beta} = (\beta_0, \ldots, \beta_m)$ is not optimal.*

Proof. Let $\beta = \left(\frac{\pi_{i-1}\beta_{i-1}^{\rho} + \pi_i \beta_i^{\rho}}{\pi_{i-1} + \pi_i}\right)^{1/\rho}$ and define $\bar{\beta}' = (\beta_0, \ldots, \beta_{i-2}, \beta, \beta, \beta_{i+1}, \ldots \beta_m)$. Then $\bar{\beta}' \in \Gamma$, and $\beta_{i-1} > \beta > \beta_i$.

We claim that $f(\bar{\beta}) > f(\bar{\beta}')$. Comparing the expressions for $f\left(\bar{\beta}\right)$ and $f\left(\bar{\beta}'\right)$ term by term, we see that it's enough to check that:

$$\pi_i \max\left\{\beta_{i-1}, \beta_i, \beta_{i+1}\right\} + \pi_{i+1} \max\left\{\beta_i, \beta_{i+1}, \beta_{i+2}\right\} \;>\; \pi_i \max\left\{\beta, \beta_{i+1}\right\} + \pi_{i+1} \max\left\{\beta, \beta_{i+1}, \beta_{i+2}\right\}$$

where the terms involving π_{i+1} are ignored when $i = m$. For $i = m$, the inequality becomes $\beta_{i-1} > \beta$ which holds by assumption. For $i = 1, \ldots, m-1$, this inequality is equivalent to:

$$\pi_i(\beta_{i-1} - \beta) > \pi_{i+1} \cdot \left(\max\left\{\beta, \beta_{i+2}\right\} - \max\left\{\beta_i, \beta_{i+2}\right\}\right)$$

which, in its strongest form (when $\beta_i \geq \beta_{i+2}$), is equivalent to $\pi_i(\beta_{i-1} - \beta) > \pi_{i+1}(\beta - \beta_i)$. But this is equivalent to:

$$\left(\frac{\pi_i \beta_{i-1} + \pi_{i+1}\beta_i}{\pi_i + \pi_{i+1}}\right)^{\rho} > \frac{\pi_{i-1}\beta_{i-1}^{\rho} + \pi_i \beta_i^{\rho}}{\pi_{i-1} + \pi_i}$$

which we can rewrite as:

$$\left(\frac{c_i + t}{c_i + 1}\right)^{\rho} - \frac{c_{i-1} + t^{\rho}}{c_{i-1} + 1} > 0, \tag{8.3}$$

letting $t = \frac{\beta_i}{\beta_{i-1}} \in [0, 1)$, and $c_i = \frac{\pi_i}{\pi_{i+1}} \geq 2^{(2\rho)^{i+1} - (2\rho)^i}$ (for $i > 0$ this is an equality; only for $i = 0$ is this a strict inequality, because p is large).

We are now left to prove (8.3). Let $F(t)$ denote the left hand side of this inequality, and note that $F(0) > 0$, because:

$$\left(\frac{c_i}{c_i+1}\right)^\rho = \left(1-\frac{1}{c_i+1}\right)^\rho \geq 1-\frac{\rho}{c_i+1} > 1-\frac{1}{c_{i-1}+1} = \frac{c_{i-1}}{c_{i-1}+1}$$

Here we used Bernoulli's inequality: $(1-x)^n \geq 1-nx$ for $0 < x < 1/n$. Then, we observed that $c_i + 1 > 2^{(2\rho)^{i+1}-(2\rho)^i} > \rho \cdot (2^{(2\rho)^i}+1) = \rho(\frac{1}{\pi_{i-1}}c_{i-1}+1) > \rho(c_{i-1}+1)$.

Now we let $t \in (0,1)$ and write $F(t) = F(0) + t^\rho G(t)$, where:

$$G(t) = \tfrac{1}{(c_i+1)^\rho}\left(\tbinom{\rho}{1}c_i^{\rho-1}\tfrac{1}{t} + \tbinom{\rho}{2}c_i^{\rho-2}\tfrac{1}{t^2} + \cdots + \tbinom{\rho}{\rho-1}c_i\tfrac{1}{t^{\rho-1}}\right) + \left(\tfrac{1}{(c_i+1)^\rho} - \tfrac{1}{c_{i-1}+1}\right).$$

If $G(t) \geq 0$, then clearly $F(t) \geq F(0) > 0$, so we are done. Otherwise, $G(t) < 0$, and in this case it easily follows that $G(1) < G(t) < 0$, hence $F(t) = F(0)+t^\rho G(t) > F(0)+G(1) = F(1) = 0$, as desired. This concludes the proof of the lemma. □

Lemma 8.11. *If there is an index $i \in \{0, 1 \ldots, m-1\}$ such that $\beta_{i-1} \leq \beta_i < \beta_{i+1}$, then $\beta = (\beta_0, \beta_1, \ldots, \beta_m)$ is not optimal.*

Proof. We proceed as in the previous lemma. Let $\beta = \left(\frac{\pi_i\beta_i^\rho+\pi_{i+1}\beta_{i+1}^\rho}{\pi_i+\pi_{i+1}}\right)^{1/\rho}$, and define $\bar{\beta}' = (\beta_0, \ldots, \beta_{i-1}, \beta, \beta, \beta_{i+2}, \ldots, \beta_m)$. As before, $\bar{\beta}' \in \Gamma$ and $\beta_i < \beta < \beta_{i+1}$. We claim that $f(\bar{\beta}) > f(\bar{\beta}')$. Comparing the expressions for $f\left(\bar{\beta}\right)$ and $f\left(\bar{\beta}'\right)$ term by term, we see that it's enough to check that

$$\pi_{i-1}\cdot\max\{\beta_{i-2}, \beta_{i-1}, \beta_i\} + \pi_i\cdot\max\{\beta_{i-1}, \beta_i, \beta_{i+1}\} > \pi_{i-1}\cdot\max\{\beta_{i-2}, \beta_{i-1}, \beta\} + \pi_i\cdot\max\{\beta_{i-1}, \beta, \beta\}$$

where the terms involving π_{i-1} appear unless $i = 0$. If $i = 0$, the above inequality becomes $\beta_{i+1} > \beta$ and we are done. For $i = 1, \ldots m-1$, the inequality is equivalent to

$$\pi_i(\beta_{i+1} - \beta) > \pi_{i-1} \cdot (\max\{\beta, \beta_{i-2}\} - \max\{\beta_i, \beta_{i-2}\})$$

which, in its strongest form (when $\beta_i \geq \beta_{i-2}$) is equivalent to $\pi_i(\beta_{i+1} - \beta) > \pi_{i-1}(\beta - \beta_i)$. The latter inequality is equivalent to

$$\left(\frac{\pi_i\beta_{i+1} + \pi_{i-1}\beta_i}{\pi_i + \pi_{i-1}}\right)^\rho > \frac{\pi_{i+1}\beta_{i+1}^\rho + \pi_i\beta_i^\rho}{\pi_{i+1} + \pi_i}$$

which we can rewrite as

$$\left(\frac{c_{i-1}t+1}{c_{i-1}+1}\right)^\rho - \frac{c_it^\rho+1}{c_i+1} > 0, \tag{8.4}$$

where $c_i = \pi_i/\pi_{i+1}$ as before, and $t = \beta_i/\beta_{i+1} \in [0, 1)$.

We are left to prove (8.4). Let $F(t)$ denote the left hand side of this inequality, and note that $F(0) > 0$, because:

$$\left(\frac{1}{c_{i-1}+1}\right)^{\rho} \;>\; \frac{1}{(2c_{i-1})^{\rho}} \;=\; \frac{1}{\pi_{i-1}^{\rho}} \cdot 2^{-\rho\cdot(2\rho)^i-\rho} \;>\; 2^{-\rho\cdot(2\rho)^i-\rho} \;\geq\; 2^{(2\rho)^i-(2\rho)^{i+1}} \;=\; \frac{1}{c_i} > \frac{1}{c_i+1}$$

Now we let $t \in (0,1)$ and write $F(t) = F(0) + t^{\rho}G(t)$, where

$$G(t) \;=\; \tfrac{1}{(c_{i-1}+1)^{\rho}}\left(\tbinom{\rho}{1}c_{i-1}\tfrac{1}{t} + \tbinom{\rho}{2}c_{i-1}^2\tfrac{1}{t^2} + \cdots + \tbinom{\rho}{\rho-1}c_{i-1}^{\rho-1}\tfrac{1}{t^{\rho-1}}\right) + \left(\left(\tfrac{c_{i-1}}{c_{i-1}+1}\right)^{\rho} - \tfrac{c_i}{c_{i-1}+1}\right).$$

If $G(t) \geq 0$, then clearly $F(t) \geq F(0) > 0$, so we are done. Otherwise, $G(t) < 0$, in which case it easily follows that $G(1) < G(t) < 0$, hence $F(t) = F(0)+t^{\rho}G(t) > F(0)+G(1) = F(1) = 0$, as desired. This concludes the proof of the lemma. □

To prove Lemma 8.8, assume $\bar{\beta} = (\beta_0, \ldots, \beta_m) \in \Gamma$ is optimal. By Lemmas 8.9 and 8.10, it follows that $\beta_0 \leq \beta_1 \leq \cdots \leq \beta_m$. Now Lemma 8.11 implies that $\beta_0 = \beta_1 = \cdots = \beta_m$. Since $\bar{\beta} \in \Gamma$, we have $\beta_i = 1$, and hence the minimal value of f over Γ is $f\,(1, 1, \ldots, 1) = 1$.

This concludes the proof of Lemma 8.8.

Chapter 9

Predecessor Search

In this chapter, we tell the fascinating story of the predecessor problem, reviewing upper bounds before delving into lower bounds, in an attempt to illustrate the powerful information-theoretic structures that abound in this problem.

See §2.1 for an introduction to the problem and a survey of known results. In this chapter, we concentrate on the static problem. When talking about upper bounds, we tend to focus on the information-theoretic aspects and gloss over implementation details on the word RAM. Formally, our descriptions can be seen as giving algorithms in the cell-probe model, in which it is free to compute anything on the data that has been read. We also ignore details regarding construction. In all cases, the preprocessing time can be made linear in the size of the data structure, starting from sorted input. For the data structures that use hash tables, the preprocessing time holds with high probability; the query is always deterministic.

Upper bounds. We begin in §9.1 with two fundamental techniques giving static, linear-space data structures: the data structure of van Emde Boas [101], and the fusion trees of Fredman and Willard [52].

The van Emde Boas data structure, dating back to FOCS'75, is undoubtedly a cornerstone of modern data structure theory. As the first interesting algorithm that exploited bounded precision for faster running times, it has prompted the modern study of predecessor search, sorting, sublogarithmic point location etc. The basic idea has continued to be an inspiration in other contexts, being applied for instance in cache oblivious layouts.

Fusion trees showed that $o(\lg n)$ search is always achievable, and demonstrated the theoretical power of word-level parallelism. This is perhaps the most widely used technique for exploiting bounded precision. The idea is to sketch multiple items down to a smaller size, and pack them in a single word. Then, regular word operations act "in parallel" over this vector of sketched data items.

Lower bounds. In §9.2, we prove our lower bounds, implying that the query time is, up to constant factors:

$$\min \begin{cases} \log_w n \\ \lg \frac{w - \lg n}{a} \\ \frac{\lg \frac{w}{a}}{\lg\left(\frac{a}{\lg n} \cdot \lg \frac{w}{a}\right)} \\ \frac{\lg \frac{w}{a}}{\lg\left(\lg \frac{w}{a} \ / \lg \frac{\lg n}{a}\right)} \end{cases}$$

Here, we defined $a = \lg \frac{S \cdot w}{n}$, where S was the space. Our main contribution is the tight lower bounds for $a = o(\lg n)$ (in particular, branches two and four of the trade-off). As mentioned already, previous techniques were helpless, since none could separate linear and polynomial space.

To avoid technical details and concentrate on the main developments, this thesis only gives the proof for the simplest case $w = 3 \lg n$, and $a = o(\lg n)$. In addition, we only talk about deterministic data structures. For the randomized lower bounds, see our publication [90].

A very strong consequence of our proofs is the idea that sharing between subproblems does not help for predecessor search. Formally, the best cell-probe complexity achievable by a data structure representing k independent subproblems (with the same parameters) in space $k \cdot \sigma$ is asymptotically equal to the best complexity achievable by a data structure for one subproblem, which uses space σ. The simplicity and strength of this statement make it interesting from both the data-structural and complexity-theoretic perspectives.

At a high level, it is precisely this sort of direct-sum property that enables us to beat communication complexity. Say we have k independent subproblems, and total space S. While in the communication game Alice sends $\lg S$ bits per round, our results intuitively state that $\lg \frac{S}{k}$ bits are sufficient. Then, by carefully controlling the increase in k and the decrease in key length (the query size), we can prevent Alice from communicating her entire input over a superconstant number of rounds.

A nice illustration of the strength of our result are the tight bounds for near linear universes, i.e. $w = \lg n + \delta$, with $\delta = o(\lg n)$. On the upper bound side, the algorithm can just start by a table lookup based on the first $\lg n$ bits of the key, which requires linear space. Then, it continues to apply van Emde Boas for δ-bit keys inside each subproblem, which gives a complexity of $O(\lg \frac{\delta}{a})$. Obtaining a lower bound is just as easy, given our techniques. We first consider $n/2^\delta$ independent subproblems, where each has 2^δ integers of 2δ bits each. Then, we prefix the integers in each subproblem by the number of the subproblem (taking $\lg n - \delta$ bits), and prefix the query with a random subproblem number. Because the universe of each subproblem ($2^{2\delta}$) is quadratically bigger than the number of keys, we can apply the usual proof showing the optimality of van Emde Boas' bound for polynomial universes. Thus, the complexity is $\Omega(\lg \frac{\delta}{a})$.

In the course of proving the deterministic lower bounds, we introduce a new concept which is crucial to the induction hypothesis: we allow the algorithm to *reject* queries, under certain conditions. In fact, the deterministic proof rejects almost all queries; nonetheless the few accepted queries remaining carry enough information to contradict a fast query time. We note that which queries are accepted depends non-trivially on the data structure and query algorithm.

Implications to range searching. Another problem closely related to predecessor search is static range searching in two dimensions. Given a set of n points at integer coordinates in the plane, the query asks whether an axis-parallel rectangle contains any point. Consider the colored predecessor problem, where elements of the set Y are red or blue, and the query should only return the color of the predecessor. Lower bounds for this problem (such as ours) also apply to range queries. The trick is to consider the interval stabbing problem, where intervals are define by a red point and the next blue point. This problem is itself easily reducible to 2D range searching (even to the special case of dominance queries, where one corner of the rectangle is always the origin).

An interesting special case is when coordinates are distinct integers in $[n]$, i.e. the problem is in rank space. This restriction occurs naturally in many important cases, such as recursion from higher-dimensional range structures, or geometric approaches to pattern matching. In FOCS'00, Alstrup et al. [7] gave a query time of $O(\lg\lg n)$, using space $O(n\lg^{\varepsilon} n)$. Clearly, predecessor lower bounds are irrelevant, since predecessors in $[n]$ are trivial to find with $O(n)$ space. In fact, no previous technique could prove a superconstant lower bound.

Our results imply a tight $\Omega(\lg\lg n)$ time bound for space $\widetilde{O}(n)$. Note that this requires a novel approach, since for dominance queries, as obtained by the old reduction, there is a constant-time upper bound (the RMQ data structures). In a nutshell, we consider a uniform $\sqrt[3]{n} \times \sqrt[3]{n}$ grid on top of our original space. In each cell, we construct a hard subproblem using the colored predecessor problem. This is possible since we get to place $\sqrt[3]{n}$ points in the space $\left[n^{2/3}\right]^2$. Finally, we can use the direct-sum properties of our lower bound, to argue that for this set of problems, the query time cannot be better than for one problem with $\sqrt[3]{n}$ points and $\widetilde{O}(\sqrt[3]{n})$ space.

9.1 Data Structures Using Linear Space

9.1.1 Equivalence to Longest Common Prefix

We write $\mathrm{lcp}(x, y)$ for the longest common prefix of x and y, i.e. the largest i, such that the most significant i bits of x and y are identical. The longest common prefix query on a set $S = \{y_1, \dots, y_n\}$ is defined as:
LCP(x)**:** returns some $i \in [n]$ that maximizes $\mathrm{lcp}(x, y_i)$.

It is immediate that predecessor search can solve LCP queries, because either the predecessor or the successor maximizes the common prefix.

It turns out that a reverse reduction also holds, and we will only reason about LCP queries from now on. (Note, though, that this reduction is not needed in practice. Most LCP data structures can solve predecessor search with some *ad hoc* tweaks, which enjoy better constant factors.)

Lemma 9.1. *If the* LCP *problem can be solved with space S and query time t_q, predecessor search can be solved with space $S + O(n)$ and query time $t_q + O(1)$.*

Proof. Consider a binary trie of depth w, and the root-to-leaf paths representing the values in $S = \{y_1, \ldots, y_n\}$. There are precisely $n - 1$ branching nodes among these n paths. Our data structure has two components:

- A hash table stores all the branching nodes, i.e. pairs (ℓ, v), where ℓ is the depth of the node and v is the prefix leading to that node. We also store as associated data the minimum and maximum values in S beginning with prefix v.
- For each $y_i \in S$, we store a bit vector of size w (exactly a word) indicating which ancestors of x are branching nodes.

The query for the predecessor of x proceeds as follows:

- Run the LCP query; let $i = \text{LCP}(x)$.
- Compute $\ell = \text{lcp}(x, y_i)$.
- Find the highest branching node, v, on the path y_i below height ℓ. This is done by examining the bit vector indicating the branching nodes above y_i, which takes constant time because the bit vector is a word.
- If v is to the left of x, the maximum value under v (stored as associated data in the hash table) is the predecessor.
- If v is to the right of x, the minimum value under v is the successor, and the predecessor is immediately before it in S.

□

9.1.2 The Data Structure of van Emde Boas

To introduce this idea, let us consider the following communication problem:

Definition 9.2. *In the* Greater-Than *problem, Alice has an n-bit number x, and Bob has an n-bit number y. They must communicate to determine whether $x > y$.*

Perhaps the most natural idea for this problem is to binary search for the length of the longest common prefix between x and y, giving a randomized protocol with $\lg n$ rounds. Formally, the algorithm works as follows:

- If $n = 1$, Alice sends x, and Bob outputs the answer.
- If $n \geq 2$, Alice sends a hash code of the most significant $\lceil \frac{n}{2} \rceil$ bits of x. Call these bits $\text{hi}(x)$, and the remaining bits $\text{lo}(x)$. Let h be the hash function.
- Bob sends a bit indicating whether $h(\text{hi}(x))$ is equal to $h(\text{hi}(y))$.
  - In the "different" case, the players know $\text{hi}(x) \neq \text{hi}(y)$, so $x < y$ iff $\text{hi}(x) < \text{hi}(y)$. They recurse on $\text{hi}(x)$ and $\text{hi}(y)$.

– In the "equal" case, the players assume $\mathrm{hi}(x) = \mathrm{hi}(y)$, which holds with good [1] probability, and recurse on $\mathrm{lo}(x)$ and $\mathrm{lo}(y)$.

The van Emde Boas data structure implements essentially the same idea for predecessor search: it binary searches for the length of the longest common prefix between the query and a value in the set. This idea was presented in somewhat obscure terms in the original paper of van Emde Boas [101]. The version we describe here was a later simplification of Willard [103], introduced under the name of "y-fast trees."

The algorithm. Let $\mathrm{hi}(y;\ell)$ be the most significant ℓ bits of a w-bit integer y. Given the input set $S = \{y_1, \ldots, y_n\}$, the data structure works as follows:

construction: For $\ell = 1 \mathinner{..} w$, we hold a hash table H_ℓ with $\{\mathrm{hi}(y_1;\ell), \ldots, \mathrm{hi}(y_n;\ell)\}$. This requires space $O(n \cdot w)$, but we show below how to improve this to $O(n)$.

query: Binary search for the largest ℓ such that $\mathrm{hi}(x;\ell) \in H_\ell$.

The query time is a very efficient $O(\lg w)$. The query time may also be expressed as $O(\lg\lg u)$, where $u = 2^w$ is the *universe* of the values.

While the double logarithm has historically been used for some artistic effect, we feel it makes readers overly optimistic about performance, and should be avoided. Consider the following back of the envelope calculation. If keys have 128 bits, as addresses in IPv6 do, then $\lg w$ is 7; note that the double logarithm comes from the huge universe 2^{128}. The leading constant is nontrivial, since each step needs a query to a hash table. If we approximate this constant as "three times slower than binary search" (computing a hash function, plus two memory accesses), then van Emde Boas becomes competitive with binary search only for sets of size $n = 2^{21} = 2M$.

Bucketing. The space can be reduced to $O(n)$ by a standard bucketing trick. We group the n elements of S in $O(n/w)$ buckets of $O(w)$ elements each. Each bucket is stored as a sorted array, and the minimum in each bucket is inserted in the predecessor structure from above. The space becomes $O(n)$.

To answer a query, first run the query algorithm from above to find the bucket i in which the predecessor should be searched. Then, do a binary search inside bucket i, taking additional $O(\lg w)$ time.

9.1.3 Fusion Trees

In external memory, if a page of B words can be read in constant time, predecessor search is solved in time $O(\log_B n)$ via B-trees. That basis of this solution is that fact that in external

[1] We omit a rigorous analysis of the failure probability, since this is just a toy example. To make the analysis precise, h is chosen from a universal family using public coins. The range of h is $O(\lg\lg n)$ bits, making a false positive occur with probability $1/\mathrm{polylog}(n)$. The protocol fails with probability $o(1)$ by a union bound over the $\lg n$ rounds.

memory, predecessor search among $O(B)$ elements can solved in constant time. B-trees are simply recursive B-ary search, based on this primitive.

Fredman and Willard asked the following intriguing question: on the word RAM, can we have constant-time search among B elements, for some $B = \omega(1)$? The memory unit is a word, which can only store one complete value. But can we somehow *compress* $\omega(1)$ words into just one word, and still be able to search for predecessors in compressed form?

The answer of Fredman and Willard [52] was affirmative. They designed a sketch that could compress $B = O(\sqrt{w})$ words down to a single word, such that LCP queries can be answered based on these sketches. By Lemma 9.1, this gives a predecessor structure among B elements with $O(1)$ query time and $O(B)$ space.

This primitive is used to construct a B-tree, giving query time $O(\log_w n)$. Since $w = \Omega(\lg n)$, this is always $O(\lg n / \lg\lg n)$, i.e. it is theoretically better than binary search for any word size. In fact, taking the best of fusion trees and van Emde Boas yields $\min\{\lg w, \log_w n\} = O(\sqrt{\lg n})$, a quadratic improvement over binary search. However, it should be noted that, unlike van Emde Boas, fusion trees do not lead to convincing practical improvements, due to the large constant factors involved.

Sketching. Let $S = \{y_1, \ldots, y_B\}$ be the values we want to sketch. As before, we view these values as root-to-leaf paths in a binary trie of depth w. There are $B - 1$ branching nodes on these paths; let T be the set of depths at which these nodes occur. We write $\text{PROJ}_T(x)$ for the projection of a value x on the bit positions of T, i.e. an integer of $|T|$ bits, with the bits of x appearing at the positions in T.

The sketch of S is simply $\text{PROJ}_T(S) = \{\text{PROJ}_T(y_1), \ldots, \text{PROJ}_T(y_B)\}$. This takes $B \cdot |T| = O(B^2)$ bits, which fits in a word for $B = O(\sqrt{w})$. We now claim that answering $\text{LCP}(x)$ on the original set S is equivalent to answering $\text{LCP}(\text{PROJ}_T(x))$ on the sketch $\text{PROJ}_T(S)$, a constant-time operation because the sketch is a word.

To prove our claim formally, consider a mental experiment in which we trace the path of x in the trie of S, and, in parallel, trace the path of $\text{PROJ}_T(x)$ in the trie of depth $|T|$ representing $\text{PROJ}_T(S)$.

We show inductively that $\text{LCP}(\text{PROJ}_T(x))$ in the sketch trie is also a valid answer for $\text{LCP}(x)$ in the original trie. We have the following cases for a node met by the path of x in the original trie:

- a branching node. This level is included in T, hence in the sketch. By induction, the LCP is valid in the two subtrees.

- a non-branching node, on some path(s) from S. The level is not necessarily included in the sketch, but may be, due to an unrelated branching node. If this level of the trie, the path in the sketch tree follows the corresponding node. Otherwise

- a node off any path in S. The first time this happens, we stop tracing the paths, since the LCP has been determined.

Implementation on the RAM. Implementing this algorithm on the word RAM is technically involved. The main ingredient of [52] is an implementation of $\textsc{proj}_T(x)$, which can compress $|T|$ scattered bits of x, via some carefully chosen multiplications and mask, into a space of $O(|T|^4)$ contiguous bits. Thus, one can only sketch $B = O(w^{1/5})$ values into a word, which is still enough for an asymptotic query time of $O(\log_w n)$.

An objection to [52] is that it uses multiplication, which is not an AC^0 operation (i.e. cannot be implemented in constant depth by a polynomial size circuit). Andersson, Miltersen and Thorup [11] show how to implement fusion trees using nonstandard AC^0 operations. The atomic heaps of Fredman and Willard [53] obtain $O(\lg n/\lg\lg n)$ query time without multiplication, via look-up tables of size $O(n^\varepsilon)$.

9.2 Lower Bounds

9.2.1 The Cell-Probe Elimination Lemma

An abstract decision data structure problem is defined by a function $f : D \times Q \to \{0, 1\}$. An input from D is given at preprocessing time, and the data structure must store a representation of it in some bounded space. An input from Q is given at query time, and the function of the two inputs must be computed through cell probes. We restrict the preprocessing and query algorithms to be deterministic. In general, we consider a problem in conjunction with a distribution $\mathcal{D}$ over $D \times Q$. Note that the distribution need not (and, in our case, will not) be a product distribution. We care about the probability the query algorithm is successful under the distribution $\mathcal{D}$, for a notion of success to be defined shortly.

As mentioned before, we work in the cell-probe model, and let w be the number of bits in a cell. We assume the query's input consists of at most w bits, and that the space bound is at most 2^w. For the sake of an inductive argument, we extend the cell-probe model by allowing the data structure to publish some bits at preprocessing time. These are bits depending on the data structure's input, which the query algorithm can inspect at no charge. Closely related to this concept is our model for a query being accepted. We allow the query algorithm not to return the correct answer, but only in the following very limited way. After inspecting the query and the published bits, the algorithm can declare that it cannot answer the query (we say it *rejects* the query). Otherwise, the query is *accepted*: the algorithm can make cell probes, and at the end it must answer the query correctly. Thus, it is not possible to reject later. In contrast to more common models of error, it actually makes sense to talk about tiny (close to zero) probabilities of accept, even for problems with boolean output.

For an arbitrary problem f and an integer $k \leq 2^w$, we define a direct-sum problem $\bigoplus^k f : D^k \times ([k] \times Q) \to \{0, 1\}$ as follows. The data structure receives a vector of inputs $(d^1, \dots, d^k)$. The representation depends arbitrarily on all of these inputs. The query is the index of a subproblem $i \in [k]$, and an element $q \in Q$. The output of $\bigoplus^k f$ is $f(q, d^i)$. We also define a distribution $\bigoplus^k \mathcal{D}$ for $\bigoplus^k f$, given a distribution $\mathcal{D}$ for f. Each d^i is chosen independently at random from the marginal distribution on D induced by $\mathcal{D}$. The subproblem i is chosen uniformly from $[k]$, and q is chosen from the distribution on Q

conditioned on d^i.

Given an arbitrary problem f and an integer $h \le w$, we can define another problem $f^{(h)}$ as follows. The query is a vector $(q_1, \dots, q_h)$. The data structure receives a regular input $d \in D$, and integer $r \in [h]$ and the prefix of the query $q_1, \dots, q_{r-1}$. The output of $f^{(h)}$ is $f(d, q_r)$. Note that we have shared information between the data structure and the querier (i.e. the prefix of the query), so $f^{(h)}$ is a partial function on the domain $D \times \bigcup_{i=0}^{t-1} Q^i \times Q$. Now we define an input distribution $\mathcal{D}^{(h)}$ for $f^{(h)}$, given an input distribution $\mathcal{D}$ for f. The value r is chosen uniformly at random. Each query coordinate q_i is chosen independently at random from the marginal distribution on Q induced by $\mathcal{D}$. Now d is chosen from the distribution on D, conditioned on q_r.

We give the $f^{(h)}$ operator precedence over the direct sum operator, i.e. $\bigoplus^k f^{(h)}$ means $\bigoplus^k \left[f^{(h)}\right]$. Using this notation, we are ready to state our central cell-probe elimination lemma:

Lemma 9.3. *There exists a universal constant C, such that for any problem f, distribution $\mathcal{D}$, and positive integers h and k, the following holds. Assume there exists a solution to $\bigoplus^k f^{(h)}$ with accept probability α over $\bigoplus^k \mathcal{D}^{(h)}$, which uses at most $k\sigma$ words of space, $\frac{1}{C}(\frac{\alpha}{h})^3 k$ published bits and T cell probes. Then, there exists a solution to $\bigoplus^k f$ with accept probability $\frac{\alpha}{4h}$ over $\bigoplus^k \mathcal{D}$, which uses the same space, $k\sqrt[h]{\sigma} \cdot Cw^2$ published bits and $T-1$ cell probes.*

9.2.2 Setup for the Predecessor Problem

Let $P(n, \ell)$ be the colored predecessor problem on n integers of ℓ bits each. Remember that this is the decision version of predecessor search, where elements are colored red or blue, and a query just returns the color of the predecessor. We first show how to identify the structure of $P(n, \ell)^{(h)}$ inside $P(n, h\ell)$, making it possible to apply our cell-probe elimination lemma.

Lemma 9.4. *For any integers $n, \ell, h \ge 1$ and distribution $\mathcal{D}$ for $P(n, \ell)$, there exists a distribution $\mathcal{D}^{*(h)}$ for $P(n, h\ell)$ such that the following holds. Given a solution to $\bigoplus^k P(n, h\ell)$ with accept probability α over $\bigoplus^k \mathcal{D}^{*(h)}$, one can obtain a solution to $\bigoplus^k P(n, \ell)^{(h)}$ with accept probability α over $\bigoplus^k \mathcal{D}^{(h)}$, which has the same complexity in terms of space, published bits, and cell probes.*

Proof. We give a reduction from $P(n, \ell)^{(h)}$ to $P(n, h\ell)$, which naturally defines the distribution $\mathcal{D}^{*(h)}$ in terms of $\mathcal{D}^{(h)}$. A query for $P(n, \ell)^{(h)}$ consists of $x_1, \dots, x_h \in \{0, 1\}^\ell$. Concatenating these, we obtain a query for $P(n, h\ell)$. In the case of $P(n, \ell)^{(h)}$, the data structure receives $i \in [h]$, the query prefix $x_1, \dots, x_{i-1}$ and a set Y of ℓ-bit integers. We prepend the query prefix to all integers in Y, and append zeros up to $h\ell$ bits. Then, finding the predecessor of x_i in Y is equivalent to finding the predecessor of the concatenation of $x_1, \dots, x_h$ in this new set. $\square$

Observe that to apply the cell-probe elimination lemma, the number of published bits must be just a fraction of k, but applying the lemma increases the published bits significantly. We want to repeatedly eliminate cell probes, so we need to amplify the number of subproblems each time, making the new number of published bits insignificant compared to the new k.

Lemma 9.5. *For any integers $t, \ell, n \geq 1$ and distribution $\mathcal{D}$ for $P(n, \ell)$, there exists a distribution $\mathcal{D}^{*t}$ for $P(n \cdot t, \ell + \lg t)$ such that the following holds. Starting from a solution to $\bigoplus^k P(n \cdot t, \ell + \lg t)$ with accept probability α over $\bigoplus^k \mathcal{D}^{*t}$, one can construct a solution to $\bigoplus^{kt} P(n, \ell)$ with accept probability α over $\bigoplus^{kt} \mathcal{D}$, which has the same complexity in terms of space, published bits, and cell probes.*

Proof. We first describe the distribution $\mathcal{D}^{*t}$. We draw $Y_1, \ldots, Y_t$ independently from $\mathcal{D}$, where Y_i is a set of integers, representing the data structures input. Prefix all numbers in Y_j by j using $\lg t$ bits, and take the union of all these sets to form the data structure's input for $P(nt, \ell + \lg t)$. To obtain the query, pick $j \in \{0, \ldots, t-1\}$ uniformly at random, pick the query from $\mathcal{D}$ conditioned on Y_j, and prefix this query by j. Now note that $\bigoplus^{kt} \mathcal{D}$ and $\bigoplus^k \mathcal{D}^{*t}$ are really the same distribution, except that the lower $\lg t$ bits of the problems index for $\bigoplus^{kt} \mathcal{D}$ are interpreted as a prefix in $\bigoplus^k \mathcal{D}^{*t}$. Thus, obtaining the new solution is simply a syntactic transformation. □

Our goal is to eliminate all cell probes, and then reach a contradiction. For this, we need the following:

Lemma 9.6. *For any $n \geq 1$ and $\ell \geq \log_2(n+1)$, there exists a distribution $\mathcal{D}$ for $P(n, \ell)$ such that the following holds. For all $(\forall) 0 < \alpha \leq 1$ and $k \geq 1$, there does not exist a solution to $\bigoplus^k P(n, \ell)$ with accept probability α over $\bigoplus^k \mathcal{D}$, which uses no cell probes and less than αk published bits.*

Proof. The distribution $\mathcal{D}$ is quite simple: the integers in the set are always 0 up to $n-1$, and the query is n. All that matters is the color of $n-1$, which is chosen uniformly at random among red and blue. Note that for $\bigoplus^k P(n, \ell)$ there are only k possible queries, i.e. only the index of the subproblem matters.

Let $\mathbf{p}$ be the random variable denoting the published bits. Since there are no cell probes, the answers to the queries are a function of $\mathbf{p}$ alone. Let $\alpha(p)$ be the fraction of subproblems that the query algorithm doesn't reject when seeing the published bits p. In our model, the answer must be correct for all these subproblems. Then, $\Pr[\mathbf{p} = p] \leq 2^{-\alpha(p)k}$, as only inputs which agree with the $\alpha(p)k$ answers of the algorithm can lead to these published bits. Now observe that $\alpha = \mathbf{E}_p[\alpha(p)] \leq \mathbf{E}_p\left[\frac{1}{k}\log_2 \frac{1}{\Pr[\mathbf{p}=p]}\right] = \frac{1}{k} H(\mathbf{p})$, where $H(\cdot)$ denotes binary entropy. Since the entropy of the published bits is bounded by their number (less than αk), we have a contradiction. □

9.2.3 Deriving the Trade-Offs

Because we will only be dealing with $\ell = w = O(\lg n)$, the bounds do not change if the space is S words instead of S bits. To simplify calculations, the exposition in this section assume the space is S words.

Our proof starts assuming that we for any possible distribution have a solution to $P(n, \ell)$ which uses $n \cdot 2^a$ space, no published bits, and successfully answers all queries in T probes, where T is small. We will then try to apply T *rounds* of the cell-probe elimination from Lemma 9.3 and 9.4 followed by the problem amplification from Lemma 9.5. After T rounds, we will be left with a non-trivial problem but no cell probes, and then we will reach a contradiction with Lemma 9.6. Below, we first run this strategy ignoring details about the distribution, but analyzing the parameters for each round. Later in Lemma 9.7, we will present a formal inductive proof using these parameters in reverse order, deriving difficult distributions for more and more cell probes.

We denote the problem parameters after i rounds by a subscript i. We have the key length ℓ_i and the number of subproblems k_i. The total number of keys remains n, so the have n/k_i keys in each subproblem. Thus, the problem we deal with in round $i+1$ is $\bigoplus^{k_i} P(\frac{n}{k_i}, \ell_i)$, and we will have some target accept probability α_i. The number of cells per subproblem is $\sigma_i = \frac{n}{k_i} 2^a$. We start the first round with $\ell_0 = \ell, \alpha_0 = 1, k_0 = 1$ and $\sigma_0 = n \cdot 2^a$.

For the cell probe elimination in Lemma 9.3 and 9.4, our proof will use the same value of $h \geq 2$ in all rounds. Then $\alpha_{i+1} \geq \frac{\alpha_i}{4h}$, so $\alpha_i \geq (4h)^{-i}$. To analyze the evolution of ℓ_i and k_i, we let t_i be the factor by which we increase the number of subproblems in round i when applying the problem amplification from Lemma 9.5. We now have $k_{i+1} = t_i \cdot k_i$ and $\ell_{i+1} = \frac{\ell_i}{h} - \lg t_i$.

When we start the first round, we have no published bits, but when we apply Lemma 9.3 in round $i+1$, it leaves us with up to $k_i \sqrt[h]{\sigma_i} \cdot Cw^2$ published bits for round $i+2$. We have to choose t_i large enough to guarantee that this number of published bits is small enough compared to the number of subproblems in round $i+2$. To apply Lemma 9.3 in round $i+2$, the number of published bits must be at most $\frac{1}{C}(\frac{\alpha_{i+1}}{h})^3 k_{i+1} = \frac{\alpha_i^3 t_i}{64Ch^6} k_i$. Hence we must set $t_i \geq \sqrt[h]{\sigma_i} \cdot 64C^2w^2h^6(\frac{1}{\alpha_i})^3$. Assume for now that $T = O(\lg \ell)$. Using $h \leq \ell$, and $\alpha_i \geq (4h)^{-T} \geq 2^{O(\lg^2 \ell)}$, we conclude it is enough to set:

$$(\forall) i : \qquad t_i \geq \sqrt[h]{\frac{n}{k_i}} \cdot 2^{a/h} \cdot w^2 \cdot 2^{\Theta(\lg^2 \ell)} \tag{9.1}$$

Now we discuss the conclusion reached at the end of the T rounds. We intend to apply Lemma 9.6 to deduce that the algorithm after T stages cannot make zero cell probes, implying that the original algorithm had to make more than T probes. Above we made sure that we after T rounds had $\frac{1}{C}(\frac{\alpha_T}{h})^3 k_T < \alpha_T k_T$ published bits, which are few enough compared to the number k_T of subproblems. The remaining conditions of Lemma 9.6 are:

$$\ell_T \geq 1 \qquad \text{and} \qquad \frac{n}{k_T} \geq 1 \tag{9.2}$$

Since $\ell_{i+1} \leq \frac{\ell_i}{2}$, this condition entails $T = O(\lg \ell)$, as assumed earlier.

Lemma 9.7. *With the above parameters satisfying* (9.1) *and* (9.2)*, for* $i = 0, \ldots, T$*, there is a distribution* $\mathcal{D}_i$ *for* $P(\frac{n}{k_i}, \ell_i)$ *so that no solution for* $\bigoplus^{k_i} P(\frac{n}{k_i}, \ell_i)$ *can have accept probability* α_i *over* $\bigoplus^{k_i} \mathcal{D}_i$ *using* $n \cdot 2^a$ *space,* $\frac{1}{C}(\frac{\alpha_i}{h})^3 k_i$ *published bits, and* $T - i$ *cell probes.*

Proof. The proof is by induction over $T - i$. A distribution that defies a good solution as in the lemma is called difficult. In the base case $i = T$, the space doesn't matter, and we get the difficult distribution directly from (9.2) and Lemma 9.6. Inductively, we use a difficult distribution $\mathcal{D}_i$ to construct a difficult distribution $\mathcal{D}_{i-1}$.

Recall that $k_i = k_{i-1}t_{i-1}$. Given our difficult distribution $\mathcal{D}_i$, we use the problem amplification in Lemma 9.5, to construct a distribution $\mathcal{D}_i^{*t_{i-1}}$ for $P(\frac{n}{k_i} \cdot t_{i-1}, \ell_i + \lg t_{i-1}) = P(\frac{n}{k_{i-1}}, \ell_i + \lg t_{i-1})$, which guarantees that no solution for $\bigoplus^{k_{i-1}} P(\frac{n}{k_{i-1}}, \ell_i + \lg t_{i-1})$ can have accept probability α_i over $\bigoplus^{k_{i-1}} \mathcal{D}_i^{*t_{i-1}}$ using $n \cdot 2^a$ space, $\frac{1}{C}(\frac{\alpha_i}{h})^3 k_i$ published bits, and $T - i$ cell probes.

Recall that (9.1) implies $k_{i-1} \sqrt[h]{\sigma_{i-1}} \cdot Cw^2 \leq \frac{1}{C}(\frac{\alpha_i}{h})^3 k_i$, hence that $k_{i-1} \sqrt[h]{\sigma_{i-1}}$ is less than the number of bits allowed published for our difficult distribution $\mathcal{D}_i^{*t_{i-1}}$. Also, recall that $\sigma_j k_j = n \cdot 2^a$ for all j. We can therefore use the cell probe elimination in Lemma 9.3, to construct a distribution $\left(\mathcal{D}_i^{*t_{i-1}}\right)^{(h)}$ for $P(\frac{n}{k_{i-1}}, \ell_i + \lg t_{i-1})^{(h)}$ so that no solution for $\bigoplus^{k_{i-1}} P(\frac{n}{k_{i-1}}, \ell_i + \lg t_{i-1})^{(h)}$ can have accept probability $\alpha_{i-1} \geq h\alpha_i$ over $\bigoplus^{k_{i-1}} \left(\mathcal{D}_i^{*t_{i-1}}\right)^{(h)}$ using $n \cdot 2^a$ space, $\frac{1}{C}(\frac{\alpha_{i-1}}{h})^3 k_{i-1}$ published bits, and $T - i + 1$ cell probes. Finally, using Lemma 9.4, we use $\left(\mathcal{D}_i^{*t_{i-1}}\right)^{(h)}$ to construct the desired difficult distribution $\mathcal{D}_{i-1}$ for $P(\frac{n}{k_{i-1}}, h(\ell_i + \lg t_{i-1})) = P(\frac{n}{k_{i-1}}, \ell_{i-1})$. □

We now show how to choose h and t_i in order to maximize the lower bound T, under the conditions of (9.1) and (9.2). We only consider the case $\ell = w = \gamma \lg n$, for constant $\gamma \geq 3$. In this case, it is enough to set $h = 2$ and $t_i = (\frac{n}{k_i})^{3/4}$. Then, $\frac{n}{k_{i+1}} = (\frac{n}{k_i})^{1/4}$, so $\lg \frac{n}{k_i} = 4^{-i} \lg n$ and $\lg t_i = \frac{3}{4} 4^{-i} \lg n$. By our recursion for ℓ_i, we have $\ell_{i+1} = \frac{\ell_i}{2} - \frac{3}{4} 4^{-i} \lg n$. Given $\ell_0 \geq 3 \lg n$, it can be seen by induction that $\ell_i \geq 3 \cdot 4^{-i} \lg n$. Indeed, $\ell_{i+1} \geq 3 \cdot 4^{-i} \cdot \frac{1}{2} \lg n - \frac{3}{4} 4^{-i} \lg n \geq 3 \cdot 4^{-(i+1)} \lg n$. By the above, (9.2) is satisfied for $T \leq \Theta(\lg \lg n)$. Finally, note that condition (9.1) is equivalent to:

$$
\begin{aligned}
\lg t_i \;&\geq\; \frac{1}{h} \lg \frac{n}{k_i} + \frac{a}{h} + \Theta(\lg w + \lg^2 \ell) \\
\Leftrightarrow\;\; \frac{3}{4} 4^{-i} \lg n \;&\geq\; \frac{1}{2} 4^{-i} \lg n + \frac{a}{2} + \Theta(\lg^2 \lg n) \\
\Leftrightarrow\;\; T \;&\leq\; \Theta\left(\lg \; \min\left\{ \frac{\lg n}{a}, \frac{\lg n}{\lg^2 \lg n} \right\}\right) \;=\; \Theta\left(\lg \frac{\lg n}{a}\right)
\end{aligned}
$$

Since (9.1) and (9.2) are satisfied, we can apply Lemma 9.7 with $i = 0$ and the initial parameters $\ell_0 = w, \alpha_0 = 1, k_0 = 1$. We conclude that there is a difficult distribution $\mathcal{D}_0$ for $P(n, \ell)$ with no solution getting accept probability 1 using $n \cdot 2^a$ space, 0 published bits, and T cell probes. Thus we have proved:

Theorem 9.8. *In any solution to the static colored predecessor problem on n ℓ-bit keys, if $\ell = \gamma \lg n$ for constant $\gamma \geq 3$, and we are allowed $n \cdot 2^a$ space, then there are data instances for which some queries take $\Omega\left(\lg \frac{\lg n}{a}\right)$ cell probes.*

9.3 Proof of Cell-Probe Elimination

We assume a solution to $\bigoplus^k f^{(h)}$, and use it to construct a solution to $\bigoplus^k f$. The new solution uses the query algorithm of the old solution, but skips the first cell probe made by this algorithm. A central component of our construction is a structural property about any query algorithm for $\bigoplus^k f^{(h)}$ with the input distribution $\bigoplus^k \mathcal{D}^{(h)}$. We now define and claim this property. In §9.3.1 uses it to construct a solution for $\bigoplus^k f$, while §9.3.2 gives the proof.

We first introduce some convenient notation. Remember that the data structure's input for $\bigoplus^k f^{(h)}$ consists of a vector $(d^1, \ldots, d^k) \in D^k$, a vector selecting the interesting segments $(r^1, \ldots, r^k) \in [h]^k$ and the query prefixes Q^i_j for all $j \in [r^i - 1]$. Denote by $\mathbf{d}, \mathbf{r}$ and $\mathbf{Q}$ the random variables giving these three components of the input. Also let $\mathbf{p}$ be the random variable representing the bits published by the data structure. Note that $\mathbf{p}$ can also be understood as a function $\mathbf{p}(\mathbf{d}, \mathbf{r}, \mathbf{Q})$. The query consists of an index i selecting the interesting subproblem, and a vector $(q_1, \ldots, q_h)$ with a query to that subproblem. Denote by $\mathbf{i}$ and $\mathbf{q}$ these random variables. Note that in our probability space $\bigoplus^k f^{(h)}$, we have $\mathbf{q}_j = \mathbf{Q}^{\mathbf{i}}_j, (\forall) j < \mathbf{r}^{\mathbf{i}}$.

Fix some instance p of the published bits and a subproblem index $i \in [k]$. Consider a prefix $(q_1, \ldots, q_j)$ for a query to this subproblem. Depending on $q_{j+1}, \ldots, q_h$, the query algorithm might begin by probing different cells, or might reject the query. Let $\Gamma^i(p; q_1, \ldots, q_j)$ be the set of cells that could be inspected by the first cell probe. Note that this set could be $\varnothing$, if all queries are rejected.

Now define:

$$\delta^i(p) = \begin{cases} 0, & \text{iff } \Gamma^i(p; \mathbf{Q}^i) = \varnothing \\ \Pr\left[|\Gamma^i(p; \mathbf{q}_1, \ldots, \mathbf{q}_{\mathbf{r}^i})| \geq \frac{\min\{\sigma, |\Gamma^i(p; \mathbf{Q}^i)|\}}{\sqrt[h]{\sigma}} \mid \mathbf{i} = i\right] & \text{otherwise} \end{cases} \tag{9.3}$$

The probability space is that defined by $\bigoplus^k \mathcal{D}^{(h)}$ when the query is to subproblem i. In particular, such a query will satisfy $\mathbf{q}_j = \mathbf{Q}^i_j, (\forall) j < \mathbf{r}^i$, because the prefix is known to the data structure. Note that this definition completely ignores the suffix $\mathbf{q}_{\mathbf{r}^i+1}, \ldots, \mathbf{q}_h$ of the query. The intuition behind this is that for any choice of the suffix, the correct answer to the query is the same, so this suffix can be "manufactured" at will. Indeed, an arbitrary choice of the suffix is buried in the definition of Γ^i.

With these observations, it is easier to understand (9.3). If the data structure knows that no query to subproblem i will be accepted, $\delta_i = 0$. Otherwise, we compare two sets of cells. The first contains the cells that the querier might probe given what the data structure knows: $\Gamma^i(p, \mathbf{Q}^i)$ contains all cells that could be probed for various $\mathbf{q}^i_{\mathbf{r}^i}$ and various suffixes. The second contains the cells that the querier could choose to probe considering its given input $\mathbf{q}^i_{\mathbf{r}^i}$ (the querier is only free to choose the suffix). Obviously, the second set is a subset of the first. The good case, whose probability is measured by δ_i, is when it is a rather large subset, or at least large compared to σ.

For convenience, we define $\delta^*(p) = \mathbf{E}_{\mathbf{i}}[\delta^{\mathbf{i}}(p)] = \frac{1}{k}\sum_i \delta^i(p)$. Using standard notation from probability theory, we write $\delta^i(p \mid E)$, when we condition on some event E in the probability of (9.3). We also write $\delta^i(p \mid X)$ when we condition on some random variable X, i.e. $\delta^i(p \mid X)$ is a function $x \mapsto \delta^i(p \mid X = x)$. We are now ready to state our claim, to be proven in §9.3.2.

Lemma 9.9. *There exist* $\mathfrak{r}$ *and* $\mathfrak{Q}$*, such that:*

$$\mathbf{E}_{\mathbf{d}}[\delta^*(\mathbf{p}(\mathfrak{r}, \mathfrak{Q}, \mathbf{d}) \mid \mathbf{r} = \mathfrak{r}, \mathbf{Q} = \mathfrak{Q}, \mathbf{d})] \geq \frac{\alpha}{2h}$$

9.3.1 The Solution for $\bigoplus^k f$

As mentioned before, we use the solution for $\bigoplus^k f^{(h)}$, and try to skip the first cell probe. To use this strategy, we need to extend an instance of $\bigoplus^k f$ to an instance of $\bigoplus^k f^{(h)}$. This is done using the $\mathfrak{r}$ and $\mathfrak{Q}$ values whose existence is guaranteed by Lemma 9.9. The extended data structure's input consists of the vector $(d^1, \ldots, d^k)$ given to $\bigoplus^k f$, and the vectors $\mathfrak{r}$ and $\mathfrak{Q}$. A query's input for $\bigoplus^k f$ is a problem index $i \in [k]$ and a $q \in Q$. We extend this to $(q_1, \ldots, q_h)$ by letting $q_j = \mathfrak{Q}^i_j, (\forall) j < \mathfrak{r}^i$, and $q_{\mathfrak{r}^i} = q$, and manufacturing a suffix $q_{\mathfrak{r}^i+1}, \ldots, q_h$ as described below.

First note that extending an input of $\bigoplus^k f$ to an input of $\bigoplus^k f^{(h)}$ by this strategy preserves the desired answer to a query (in particular, the suffix is irrelevant to the answer). Also, this transformation is well defined because $\mathfrak{r}$ and $\mathfrak{Q}$ are "constants", defined by the input distribution $\bigoplus^k \mathcal{D}^{(h)}$. Since our model is nonuniform, we only care about the existence of $\mathfrak{r}$ and $\mathfrak{Q}$, and not about computational aspects.

To fully describe a solution to $\bigoplus^k f$, we must specify how to obtain the data structure's representation and the published bits, and how the query algorithm works. The data structure's representation is identical to the representation for $\bigoplus^k f^{(h)}$, given the extended input. The published bits for $\bigoplus^k f$ consist of the published bits for $\bigoplus^k f^{(h)}$, plus a number of published cells from the data structure's representation. Which cells are published will be detailed below. We publish the cell address together with its contents, so that the query algorithm can tell whether a particular cell is available.

The query algorithm is now simple to describe. Remember that $q_1, \ldots, q_{\mathfrak{r}^i-1}$ are prescribed by $\mathfrak{Q}$, and $q_{\mathfrak{r}^i} = q$ is the original input of $\bigoplus^k f$. We now iterate through all possible query suffixes. For each possibility, we simulate the extended query using the algorithm for $\bigoplus^k f^{(h)}$. If this algorithm rejects the query, or the first probed cell is not among the published cells, we continue trying suffixes. Otherwise, we stop, obtain the value for the first cell probe from the published cells and continue to simulate this query using actual cell probes. If we don't find any good suffix, we reject the query. It is essential that we can recognize accepts in the old algorithm by looking just at published bits. Then, searching for a suffix that would not be rejected is free, as it does not involve any cell probes.

Publishing Cells

It remains to describe which cells the data structure chooses to publish, in order to make the query algorithm accept with the desired probability. Let p be the bits published by the $\bigoplus^k f^{(h)}$ solution. Note that in order for query (i, q) to be accepted, we must publish one cell from $\Gamma^i(p; \mathfrak{Q}^i, q)$. Here, we slightly abuse notation by letting $\mathfrak{Q}^i, q$ denote the $\mathfrak{r}^i$ entries of the prefix $\mathfrak{Q}^i$, followed by q. We will be able to achieve this for all (i, q) satisfying:

$$\Gamma^i(p; \mathfrak{Q}^i) \neq \varnothing, \quad |\Gamma^i(p; \mathfrak{Q}^i, q)| \geq \frac{\min\{\sigma, |\Gamma^i(p; \mathbf{Q}^i)|\}}{\sqrt[h]{\sigma}} \tag{9.4}$$

Comparing to (9.3), this means the accept probability is at least $\delta^*(p \mid \mathbf{r} = \mathfrak{r}, \mathbf{Q} = \mathfrak{Q}, \mathbf{d} = (d_1, \dots, d_k))$. Then on average over possible inputs $(d_1, \dots, d_k)$ to $\bigoplus^k f$, the accept probability will be at least $\frac{\alpha}{2h}$, as guaranteed by Lemma 9.9.

We will need the following standard result:

Lemma 9.10. *Consider a universe $U \neq \varnothing$ and a family of sets $\mathcal{F}$ such that $(\forall) S \in \mathcal{F}$ we have $S \subset U$ and $|S| \geq \frac{|U|}{B}$. Then there exists a set $T \subset U, |T| \leq B \ln |\mathcal{F}|$ such that $(\forall) S \in \mathcal{F}, S \cap T \neq \varnothing$.*

Proof. Choose $B \ln |\mathcal{F}|$ elements of U with replacement. For a fixed $S \in \mathcal{F}$, an element is outside S with probability at most $1 - \frac{1}{B}$. The probability all elements are outside S is at most $(1 - \frac{1}{B})^{B \ln |\mathcal{F}|} < e^{-\ln |\mathcal{F}|} < \frac{1}{|\mathcal{F}|}$. By the union bound, all sets in $\mathcal{F}$ are hit at least once with positive probability, so a good T exists. $\square$

We distinguish three types of subproblems, parallel to (9.4). If $\Gamma^i(p; \mathfrak{Q}^i) = \varnothing$, we make no claim (the accept probability can be zero). Otherwise, if $|\Gamma^i(p; \mathfrak{Q}^i)| < \sigma$, we handle subproblem i using a local strategy. Consider all q such that $|\Gamma^i(p; \mathfrak{Q}^i, q)| \geq \frac{|\Gamma^i(p; \mathfrak{Q}^i)|}{\sqrt[h]{\sigma}}$. We now apply Lemma 9.10 with the universe $\Gamma^i(p; \mathfrak{Q}^i)$ and the family $\Gamma^i(p; \mathfrak{Q}^i, q)$, for all interesting q's. There are at most 2^w choices of q, bounding the size of the family. Then, the lemma guarantees that the data structure can publish a set of $O(\sqrt[h]{\sigma} \cdot w)$ cells which contains at least one cell from each interesting set. This means that each interesting q can be handled by the algorithm.

We handle the third type of subproblems, namely those with $|\Gamma^i(p; \mathfrak{Q}^i)| \geq \sigma$, in a global fashion. Consider all "interesting" pairs (i, q) with $|\Gamma^i(p; \mathfrak{Q}^i, q)| \geq \sigma^{1-1/h}$. We now apply Lemma 9.10 with the universe consisting of all $k\sigma$ cells, and the family being $\Gamma^i(p; \mathfrak{Q}^i, q)$, for interesting (i, q). The cardinality of the family is at most 2^w, since i and q form a query, which takes at most one word. Then by Lemma 9.10, the data structure can publish a set of $O(k \sqrt[h]{\sigma} \cdot w)$ cells, which contains at least one cell from each interesting set. With these cells, the algorithm can handle all interesting (i, q) queries.

The total number of cells that we publish is $O(k \sqrt[h]{\sigma} \cdot w)$. Thus, we publish $O(k \sqrt[h]{\sigma} \cdot w^2)$ new bits, plus $O(k)$ bits from the assumed solution to $\bigoplus^k f^{(h)}$. For big enough C, this is at most $k \sqrt[h]{\sigma} \cdot C w^2$.

9.3.2 An Analysis of $\bigoplus^k f^{(h)}$: Proof of Lemma 9.9

Our analysis has two parts. First, we ignore the help given by the published bits, by assuming they are constantly set to some value p. As $\mathbf{r}^i$ and $\mathbf{Q}^i$ are chosen randomly, we show that the conditions of (9.3) are met with probability at least $\frac{1}{h}$ times the accept probability for subproblem i. This is essentially a lower bound on δ^i, and hence on δ^*.

Secondly, we show that the published bits do not really affect this lower bound on δ^*. The intuition is that there are two few published bits (much fewer than k) so for most subproblems they are providing no information at all. That is, the behavior for that subproblem is statistically close to when the published bits would not be used. Formally, this takes no more than a (subtle) application of Chernoff bounds. The gist of the idea is to consider some setting p for the published bits, and all possible inputs (not just those leading to p being published). In this probability space, δ^i are independent for different i, so the average is close to δ^* with overwhelmingly high probability. Now pessimistically assume all inputs where the average of δ^i is not close to δ^* are possible inputs, i.e. input for which p would be the real published bits. However, the probability of this event is so small, that even after a union bound for all p, it is still negligible.

We now proceed to the first part of the analysis. Let $\alpha^i(p)$ be the probability that the query algorithm accepts when receiving a random query for subproblem i. Formally, $\alpha^i(p) = \Pr[\Gamma^i(p; \mathbf{q}) \neq \varnothing \mid \mathbf{i} = i]$. We define $\alpha^i(p \mid E), \alpha^i(p \mid X)$ and $\alpha^*(\cdot)$ similar to the functions associated to δ^i. Observe that the probability of correctness guaranteed by assumption is $\alpha = \mathbf{E}_{\mathbf{r},\mathbf{Q},\mathbf{d}}[\alpha^*(\mathbf{p}(\mathbf{r}, \mathbf{Q}, \mathbf{d}) \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})]$.

Lemma 9.11. *For any i and p, we have $\delta^i(p) \geq \frac{\alpha^i(p)}{h}$.*

Proof. Let us first recall the random experiment defining $\delta^i(p)$. We select a uniformly random $r \in [h]$ and random $q_1, \ldots, q_{r-1}$. First we ask whether $\Gamma^i(p; q_1, \ldots, q_{r-1}) = \varnothing$. If not, we ask about the probability that a random q_r is good, in the sense of (9.3). Now let us rephrase the probability space as follows: first select $q_1, \ldots, q_h$ at random; then select $r \in [h]$ and use just $q_1, \ldots, q_r$ as above. The probability that the query $(q_1, \ldots, q_h)$ is accepted is precisely $\alpha^i(p)$. Let's assume it doesn't. Then, for any r, $\Gamma^i(p; q_1, \ldots, q_{r-1}) \neq \varnothing$ because there is at least one suffix which is accepted. We will now show that there is at least one choice of r such that q_r is good when the prefix is $q_1, \ldots, q_{r-1}$. When averaged over $q_1, \ldots, q_{r-1}$, this gives a probability of at least $\frac{\alpha^i(p)}{h}$

To show one good r, let $\phi_r = \min\{|\Gamma^i(p; q_1, \ldots, q_{r-1})|, \sigma\}$. Now observe that $\frac{\phi_1}{\phi_2} \cdot \frac{\phi_2}{\phi_3} \cdot \cdots \cdot \frac{\phi_{h-1}}{\phi_h} = \frac{\phi_1}{\phi_h} \leq \phi_1 \leq \sigma$. By the pigeonhole principle, $(\exists) r : \frac{\phi_r}{\phi_{r+1}} \leq \sigma^{1/h}$. This implies $|\Gamma^i(p; q_1, \ldots, q_r)| \geq \frac{\min\{\sigma, |\Gamma^i(p; q_1, \ldots, q_{r-1})|}{\sqrt[h]{\sigma}}$, as desired. $\square$

Note that if the algorithm uses zero published bits, we are done. Thus, for the rest of the analysis we may assume $\frac{1}{C}(\frac{\alpha}{h})^3 k \geq 1$. We now proceed to the second part of the analysis, showing that δ^* is close to the lower bound of the previous lemma, even after a union bound over all possible published bits.

Lemma 9.12. *With probability at least $1 - \frac{\alpha}{8h}$ over random $\mathbf{r}, \mathbf{Q}$ and $\mathbf{d}$, we have $(\forall) p : \delta^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) \geq \frac{\alpha^*(p)}{h} - \frac{\alpha}{4h}$*

Proof. Fix p arbitrarily. By definition, $\delta^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) = \frac{1}{k}\sum_i \delta^i(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})$. By Lemma 9.11, $\mathbf{E}[\delta^i(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})] = \delta^i(p) \geq \frac{\alpha^i(p)}{h}$, which implies $\delta^*(p) \geq \frac{\alpha^*(p)}{h}$. Thus, our condition can be rephrased as:

$$\frac{1}{k}\sum_i \delta^i(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) \geq \mathbf{E}\left[\frac{1}{k}\sum_i \delta^i(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})\right] - \frac{\alpha}{4h}$$

Now note that $\delta^i(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})$ only depends on $\mathbf{r}^i, \mathbf{Q}^i$ and $\mathbf{d}^i$, since we are looking at the behavior of a query to subproblem i for a fixed value of the published bits; see the definition of δ^i in (9.3). Since $(\mathbf{r}^i, \mathbf{Q}^i, \mathbf{d}^i)$ are independent for different i, it follows that $\delta^i(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})$ are also independent. Then we can apply a Chernoff bound to analyze the mean $\delta^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})$ of these independent random variables. We use an additive Chernoff bound [4]:

$$\Pr_{\mathbf{r},\mathbf{Q},\mathbf{d}}\left[\delta^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) < \delta^*(p) - \frac{\alpha}{4h}\right] < e^{-\Omega(k(\frac{\alpha}{h})^2)}$$

Now we take a union bound over all possible choices p for the published bits. The probability of the bad event becomes $2^{\frac{1}{C}(\frac{\alpha}{h})^3 k} e^{-\Omega((\frac{\alpha}{h})^2 k)}$. For large enough C, this is $\exp(-\Omega((\frac{\alpha}{h})^2 k))$, for any α and h. Now we use that $\frac{1}{C}(\frac{\alpha}{h})^3 k \geq 1$, from the condition that there is at lest one published bit, so this probability is at most $e^{-\Omega(Ch/\alpha)}$. Given that $\frac{h}{\alpha} \geq 1$, this is at most $\frac{\alpha}{8h}$ for large enough C. ∎

Unfortunately, this lemma is not exactly what we would want, since it provides a lower bound in terms of $\alpha^*(p)$. This accept probability is measured in the original probability space. As we condition on $\mathbf{r}, \mathbf{Q}$ and $\mathbf{d}$, the probability space can be quite different. However, we show next that in fact α^* cannot change too much. As before, the intuition is that there are too few published bits, so for most subproblems they are not changing the query distribution significantly.

Lemma 9.13. *With probability at least* $1-\frac{\alpha}{8}$ *over random* $\mathbf{r}, \mathbf{Q}$ *and* $\mathbf{d}$*, we have:* $(\forall)p:\ \alpha^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) \leq \alpha^*(p) + \frac{\alpha}{4}$

Proof. The proof is very similar to that of Lemma 9.12. Fix p arbitrarily. By definition, $\alpha^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})$ is the average of $\alpha^i(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})$. Note that for fixed p, α^i depends only on $\mathbf{r}^i, \mathbf{Q}^i$ and $\mathbf{d}^i$. Hence, the α^i values are independent for different i, and we can apply a Chernoff bound to say the mean is close to its expectation. The rest of the calculation is parallel to that of Lemma 9.12. ∎

We combine Lemmas 9.12 and 9.13 by a union bound. We conclude that with probability at least $1 - \frac{\alpha}{4}$ over random $\mathbf{r}, \mathbf{Q}$ and $\mathbf{d}$, we have that $(\forall)p$:

$$\left.\begin{array}{l} \delta^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) \geq \frac{\alpha^*(p)}{h} - \frac{\alpha}{4h} \\ \alpha^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) \leq \alpha^*(p) + \frac{\alpha}{4} \end{array}\right\} \quad \Rightarrow \quad \delta^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) - \frac{\alpha^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})}{h} \geq -\frac{\alpha}{2h} \tag{9.5}$$

Since this holds for all p, it also holds for $p = \mathbf{p}$, i.e. the actual bits $\mathbf{p}(\mathbf{r}, \mathbf{Q}, \mathbf{d})$ published by the data structure given its input. Now we want to take the expectation over $\mathbf{r}, \mathbf{Q}$ and

d. Because $\delta^*(\cdot), \alpha^*(\cdot) \in [0,1]$, we have $\delta^*(\cdot) - \frac{1}{h}\alpha^*(\cdot) \geq -\frac{1}{h}$. We use this as a pessimistic estimate for the cases of $\mathbf{r}, \mathbf{Q}$ and $\mathbf{d}$ where (9.5) does not hold. We obtain:

$$\begin{aligned} & \mathbf{E}\left[\delta^*(\mathbf{p} \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) \quad - \quad \frac{\alpha^*(\mathbf{p} \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})}{h}\right] \geq -\frac{\alpha}{2h} - \frac{\alpha}{4} \cdot \frac{1}{h} \\ \Rightarrow \quad & \mathbf{E}\big[\delta^*(\mathbf{p} \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})\big] \;\geq\; \frac{1}{h}\mathbf{E}\big[\alpha^*(\mathbf{p} \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})\big] - \frac{3\alpha}{4h} \;=\; \frac{1}{h}\alpha - \frac{3\alpha}{4h} \;=\; \frac{\alpha}{4h} \end{aligned}$$

Bibliography

[1] Yehuda Afek, Anat Bremler-Barr, and Sariel Har-Peled. Routing with a clue. *IEEE/ACM Transactions on Networking*, 9(6):693–705, 2001. See also SIGCOMM'99.

[2] Pankaj K. Agarwal. Range searching. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry (2nd edition)*. Chapman & Hall/CRC, 2004.

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

[4] Noga Alon and Joel Spencer. *The Probabilistic Method*. John Wiley, 2nd edition, 2000.

[5] Stephen Alstrup, Amir M. Ben-Amram, and Theis Rauhe. Worst-case and amortised optimality in union-find. In *Proc. 31st ACM Symposium on Theory of Computing (STOC)*, pages 499–506, 1999.

[6] Stephen Alstrup, Gerth S. Brodal, Inge Li Gørtz, and Theis Rauhe. Time and space efficient multi-method dispatching. In *Proc. 8th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 20–29, 2002.

[7] Stephen Alstrup, Gerth S. Brodal, and Theis Rauhe. New data structures for orthogonal range searching. In *Proc. 41st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 198–207, 2000.

[8] Stephen Alstrup, Thore Husfeldt, and Theis Rauhe. Marked ancestor problems. In *Proc. 39th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 534–543, 1998.

[9] Stephen Alstrup, Thore Husfeldt, and Theis Rauhe. A cell probe lower bound for dynamic nearest-neighbor searching. In *Proc. 12th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 779–780, 2001.

[10] Arne Andersson, Peter Bro Miltersen, Søren Riis, and Mikkel Thorup. Static dictionaries on AC^0 RAMs: Query time $\Theta(\sqrt{\log n/\log\log n})$ is necessary and sufficient. In *Proc. 37th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 441–450, 1996.

[11] Arne Andersson, Peter Bro Miltersen, and Mikkel Thorup. Fusion trees can be implemented with AC^0 instructions only. *Theoretical Computer Science*, 215(1-2):337–344, 1999.

[12] Arne Andersson and Mikkel Thorup. Dynamic ordered sets with exponential search trees. *Journal of the ACM*, 54(3), 2007. See also FOCS'96, STOC'00.

[13] Alexandr Andoni, Dorian Croitoru, and Mihai Pătraşcu. Hardness of nearest-neighbor search under l-infinity. In *Proc. 49th IEEE Symposium on Foundations of Computer Science (FOCS)*, 2008.

[14] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1):117–122, 2008. See also FOCS'06.

[15] Alexandr Andoni, Piotr Indyk, and Robert Krauthgamer. Overcoming the ℓ_1 non-embeddability barrier: Algorithms for product metrics. Manuscript, 2008.

[16] Alexandr Andoni, Piotr Indyk, and Mihai Pătraşcu. On the optimality of the dimensionality reduction method. In *Proc. 47th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 449–458, 2006.

[17] David Applegate, Gruia Calinescu, David S. Johnson, Howard J. Karloff, Katrina Ligett, and Jia Wang. Compressing rectilinear pictures and minimizing access control lists. In *Proc. 18th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 1066–1075, 2007.

[18] Lars Arge. The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, 2003. See also WADS'95.

[19] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *Journal of Computer and System Sciences*, 68(4):702–732, 2004. See also FOCS'02.

[20] Omer Barkol and Yuval Rabani. Tighter lower bounds for nearest neighbor search and related problems in the cell probe model. *Journal of Computer and System Sciences*, 64(4):873–896, 2002. See also STOC'00.

[21] Paul Beame and Faith E. Fich. Optimal bounds for the predecessor problem and related problems. *Journal of Computer and System Sciences*, 65(1):38–72, 2002. See also STOC'99.

[22] Amir M. Ben-Amram and Zvi Galil. A generalization of a lower bound technique due to Fredman and Saks. *Algorithmica*, 30(1):34–66, 2001. See also FOCS'91.

[23] Amir M. Ben-Amram and Zvi Galil. Lower bounds for dynamic data structures on algebraic RAMs. *Algorithmica*, 32(3):364–395, 2002. See also FOCS'91.

[24] Michael A. Bender and Martin Farach-Colton. The lca problem revisited. In *Proc. 4th Latin American Theoretical Informatics (LATIN)*, pages 88–94, 2000.

[25] Allan Borodin, Rafail Ostrovsky, and Yuval Rabani. Lower bounds for high dimensional nearest neighbor search and related problems. In *Proc. 31st ACM Symposium on Theory of Computing (STOC)*, pages 312–321, 1999.

[26] Amit Chakrabarti, Bernard Chazelle, Benjamin Gum, and Alexey Lvov. A lower bound on the complexity of approximate nearest-neighbor searching on the hamming cube. In *Proc. 31st ACM Symposium on Theory of Computing (STOC)*, pages 305–311, 1999.

[27] Amit Chakrabarti and Oded Regev. An optimal randomised cell probe lower bound for approximate nearest neighbour searching. In *Proc. 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 473–482, 2004.

[28] Timothy M. Chan. Closest-point problems simplified on the RAM. In *Proc. 13th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 472–473, 2002.

[29] Moses Charikar, Piotr Indyk, and Rina Panigrahy. New algorithms for subset query, partial match, orthogonal range searching, and related problems. In *Proc. 29th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 451–462, 2002.

[30] Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17:427–462, 1988. See also FOCS'85.

[31] Bernard Chazelle. Lower bounds for orthogonal range searching II. The arithmetic model. *Journal of the ACM*, 37(3):439–463, 1990. See also FOCS'86.

[32] Bernard Chazelle. Lower bounds for off-line range searching. *Discrete & Computational Geometry*, 17(1):53–65, 1997. See also STOC'95.

[33] Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In *Proc. 36th ACM Symposium on Theory of Computing (STOC)*, pages 91–100, 2004.

[34] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. 20th ACM Symposium on Computational Geometry (SoCG)*, pages 253–262, 2004.

[35] Mark de Berg, Marc J. van Kreveld, and Jack Snoeyink. Two- and three-dimensional point location in rectangular subdivisions. *Journal of Algorithms*, 18(2):256–277, 1995. See also SWAT'92.

[36] Mikael Degermark, Andrej Brodnik, Svante Carlsson, and Stephen Pink. Small forwarding tables for fast routing lookups. In *Proc. ACM SIGCOMM*, pages 3–14, 1997.

[37] Erik D. Demaine, Friedhelm Meyer auf der Heide, Rasmus Pagh, and Mihai Pătraşcu. De dictionariis dynamicis pauco spatio utentibus (lat. on dynamic dictionaries using little space). In *Proc. Latin American Theoretical Informatics (LATIN)*, pages 349–361, 2006.

[38] Paul F. Dietz. Optimal algorithms for list indexing and subset rank. In *Proc. 1st Workshop on Algorithms and Data Structures (WADS)*, pages 39–46, 1989.

[39] Martin Dietzfelbinger and Friedhelm Meyer auf der Heide. A new universal class of hash functions and dynamic hashing in real time. In *Proc. 17th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 6–19, 1990.

[40] David Eppstein, Giuseppe F. Italiano, Roberto Tamassia, Robert E. Tarjan, Jeffery R. Westbrook, and Moti Yung. Maintenance of a minimum spanning forest in a dynamic planar graph. *Journal of Algorithms*, 13(1):33–54, 1992. See also SODA'90.

[41] David Eppstein and S. Muthukrishnan. Internet packet filter management and rectangle geometry. In *Proc. 12th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 827–835, 2001.

[42] Ronald Fagin. Combining fuzzy information from multiple systems. *Journal of Computer and System Sciences*, 58(1):83–99, 1999. See also PODS'96, PODS'98.

[43] Martin Farach-Colton and Piotr Indyk. Approximate nearest neighbor algorithms for hausdorff metrics via embeddings. In *Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 171–180, 1999.

[44] Anja Feldmann and S. Muthukrishnan. Tradeoffs for packet classification. In *Proc. IEEE INFOCOM*, pages 1193–1202, 2000.

[45] Paolo Ferragina and S. Muthukrishnan. Efficient dynamic method-lookup for object oriented languages. In *Proc. 4th European Symposium on Algorithms (ESA)*, pages 107–120, 1996.

[46] Paolo Ferragina, S. Muthukrishnan, and Mark de Berg. Multi-method dispatching: A geometric approach with applications to string matching problems. In *Proc. 31st ACM Symposium on Theory of Computing (STOC)*, pages 483–491, 1999.

[47] Michael L. Fredman. A lower bound on the complexity of orthogonal range queries. *Journal of the ACM*, 28:696–705, 1981.

[48] Michael L. Fredman. The complexity of maintaining an array and computing its partial sums. *Journal of the ACM*, 29(1):250–260, 1982.

[49] Michael L. Fredman and Monika Rauch Henzinger. Lower bounds for fully dynamic connectivity problems in graphs. *Algorithmica*, 22(3):351–362, 1998.

[50] Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0(1) worst case access time. *Journal of the ACM*, 31(3):538–544, 1984. See also FOCS'82.

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

[52] Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47(3):424–436, 1993. See also STOC'90.

[53] Michael L. Fredman and Dan E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *Journal of Computer and System Sciences*, 48(3):533–551, 1994. See also FOCS'90.

[54] Haripriyan Hampapuram and Michael L. Fredman. Optimal biweighted binary trees and the complexity of maintaining partial sums. *SIAM Journal on Computing*, 28(1):1–9, 1998. See also FOCS'93.

[55] Monika Rauch Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *Journal of the ACM*, 46(4):502–516, 1999. See also STOC'95.

[56] Monika Rauch Henzinger and Mikkel Thorup. Sampling to provide or to bound: With applications to fully dynamic graph algorithms. *Random Structures and Algorithms*, 11(4):369–379, 1997. See also ICALP'96.

[57] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM*, 48(4):723–760, 2001. See also STOC'98.

[58] Thore Husfeldt and Theis Rauhe. New lower bound techniques for dynamic partial sums and related problems. *SIAM Journal on Computing*, 32(3):736–753, 2003. See also ICALP'98.

[59] Thore Husfeldt, Theis Rauhe, and Søren Skyum. Lower bounds for dynamic transitive closure, planar point location, and parentheses matching. In *Proc. 5th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 198–211, 1996.

[60] Piotr Indyk. Approximate algorithms for high-dimensional geometric problems. Invited talk at DIMACS Workshop on Computational Geometry'02. `http://people.csail.mit.edu/indyk/high.ps`, 2001.

[61] Piotr Indyk. On approximate nearest neighbors under ℓ_∞ norm. *Journal of Computer and System Sciences*, 63(4):627–638, 2001. See also FOCS'98.

[62] Piotr Indyk. Approximate nearest neighbor algorithms for Frechet metric via product metrics. In *Proc. 18th ACM Symposium on Computational Geometry (SoCG)*, pages 102–106, 2002.

[63] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. 30th ACM Symposium on Theory of Computing (STOC)*, pages 604–613, 1998.

[64] T. S. Jayram, Subhash Khot, Ravi Kumar, and Yuval Rabani. Cell-probe lower bounds for the partial match problem. *Journal of Computer and System Sciences*, 69(3):435–447, 2004. See also STOC'03.

[65] Bala Kalyanasundaram and Georg Schnitger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5(4):545–557, 1992. See also Structures'87.

[66] Robert Krauthgamer, James R. Lee, Manor Mendel, and Assaf Naor. Measured descent: A new embedding method for finite metrics. *Geometric And Functional Analysis*, 15(4):839–858, 2005. See also FOCS'04.

[67] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM Journal on Computing*, 30(2):457–474, 2000. See also STOC'98.

[68] Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995. See also FOCS'94.

[69] Ding Liu. A strong lower bound for approximate nearest neighbor searching. *Information Processing Letters*, 92(1):23–29, 2004.

[70] Peter Bro Miltersen. The bit probe complexity measure revisited. In *Proc. 10th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 662–671, 1993.

[71] Peter Bro Miltersen. Lower bounds for Union-Split-Find related problems on random access machines. In *Proc. 26th ACM Symposium on Theory of Computing (STOC)*, pages 625–634, 1994.

[72] Peter Bro Miltersen. Cell probe complexity - a survey. In *Proc. 19th Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 1999. Advances in Data Structures Workshop.

[73] Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences*, 57(1):37–49, 1998. See also STOC'95.

[74] Peter Bro Miltersen, Sairam Subramanian, Jeffrey S. Vitter, and Roberto Tamassia. Complexity models for incremental computation. *Theoretical Computer Science*, 130(1):203–236, 1994. See also STACS'93.

[75] Christian Worm Mortensen. Fully dynamic orthogonal range reporting on ram. *SIAM Journal on Computing*, 35(6):1494–1525, 2006. See also SODA'03.

[76] Christian Worm Mortensen, Rasmus Pagh, and Mihai Pătraşcu. On dynamic range reporting in one dimension. In *Proc. 37th ACM Symposium on Theory of Computing (STOC)*, pages 104–111, 2005.

[77] Rajeev Motwani, Assaf Naor, and Rina Panigrahy. Lower bounds on locality sensitive hashing. *SIAM Journal on Discrete Mathematics*, 21(4):930–935, 2007. See also SoCG'06.

[78] S. Muthukrishnan and Martin Müller. Time and space efficient method-lookup for object-oriented programs. In *Proc. 7th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 42–51, 1996.

[79] Yakov Nekrich. A data structure for multi-dimensional range reporting. In *Proc. 23rd ACM Symposium on Computational Geometry (SoCG)*, pages 344–353, 2007.

[80] Rina Panigrahy. Entropy based nearest neighbor search in high dimensions. In *Proc. 17th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 1186–1195, 2006.

[81] Mihai Pătraşcu. Lower bounds for 2-dimensional range counting. In *Proc. 39th ACM Symposium on Theory of Computing (STOC)*, pages 40–46, 2007.

[82] Mihai Pătraşcu. (data) structures. In *Proc. 49th IEEE Symposium on Foundations of Computer Science (FOCS)*, 2008.

[83] Mihai Pătraşcu and Erik D. Demaine. Lower bounds for dynamic connectivity. In *Proc. 36th ACM Symposium on Theory of Computing (STOC)*, pages 546–553, 2004.

[84] Mihai Pătraşcu and Erik D. Demaine. Tight bounds for the partial-sums problem. In *Proc. 15th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 20–29, 2004.

[85] Mihai Pătraşcu and Mikkel Thorup. Planning for fast connectivity updates. In *Proc. 48th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 263–271, 2007.

[86] Mihai Pătraşcu and Erik D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4):932–963, 2006. See also SODA'04 and STOC'04.

[87] Mihai Pătraşcu and Corina Tarniţă. On dynamic bit-probe complexity. *Theoretical Computer Science*, 380:127–142, 2007. See also ICALP'05.

[88] Mihai Pătraşcu and Mikkel Thorup. Higher lower bounds for near-neighbor and further rich problems. In *Proc. 47th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 646–654, 2006.

[89] Mihai Pătraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *Proc. 38th ACM Symposium on Theory of Computing (STOC)*, pages 232–240, 2006.

[90] Mihai Pătraşcu and Mikkel Thorup. Randomization does not help searching predecessors. In *Proc. 18th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 555–564, 2007.

[91] Satish Rao. Small distortion and volume preserving embeddings for planar and Euclidean metrics. In *Proc. 15th ACM Symposium on Computational Geometry (SoCG)*, pages 300–306, 1999.

[92] Alexander A. Razborov. On the distributional complexity of disjointness. *Theoretical Computer Science*, 106(2):385–390, 1992.

[93] Ronald L. Rivest. Partial-match retrieval algorithms. *SIAM Journal on Computing*, 5(1):19–50, 1976. See also FOCS'74 and Stanford PhD thesis.

[94] Hanan Samet. *Foundations of Multidimensional and Metric Data Structures.* Elsevier, 2006.

[95] Pranab Sen and Srinivasan Venkatesh. Lower bounds for predecessor searching in the cell probe model. *Journal of Computer and System Sciences*, 74(3):364–385, 2008. See also ICALP'01, CCC'03.

[96] Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk, editors. *Nearest Neighbor Methods in Learning and Vision.* Neural Processing Information Series, MIT Press, 2006.

[97] Daniel D. Sleator and Robert E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983. See also STOC'81.

[98] Kunal Talwar, Rina Panigrahy, and Udi Wieder. A geometric approach to lower bounds for approximate near-neighbor search and partial match. In *Proc. 49th IEEE Symposium on Foundations of Computer Science (FOCS)*, 2008.

[99] Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In *Proc. 32nd ACM Symposium on Theory of Computing (STOC)*, pages 343–350, 2000.

[100] Mikkel Thorup. Space efficient dynamic stabbing with fast queries. In *Proc. 35th ACM Symposium on Theory of Computing (STOC)*, pages 649–658, 2003.

[101] Peter van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10:99–127, 1977. Conference version by van Emde Boas alone in FOCS'75.

[102] Marcel Waldvogel, George Varghese, Jonathan S. Turner, and Bernhard Plattner. Scalable high speed ip routing lookups. In *Proc. ACM SIGCOMM*, pages 25–36, 1997.

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

[104] Dan E. Willard. Examining computational geometry, van Emde Boas trees, and hashing from the perspective of the fusion tree. *SIAM Journal on Computing*, 29(3):1030–1049, 2000. See also SODA'92.

[105] Bing Xiao. New bounds in cell probe model. PhD thesis, University of California at San Diego, 1992.

[106] Andrew Chi-Chih Yao. Should tables be sorted? *Journal of the ACM*, 28(3):615–628, 1981. See also FOCS'78.

[107] Andrew Chi-Chih Yao. On the complexity of maintaining partial sums. *SIAM Journal on Computing*, 14:277–288, 1985.