THEORY OF COMPUTING, Volume 11 (19), 2015, pp. 471–489
www.theoryofcomputing.org

Adapt or Die: Polynomial Lower Bounds for Non-Adaptive Dynamic Data Structures

Joshua Brody Kasper Green Larsen

Received August 12, 2013; Revised October 3, 2014; Published December 30, 2015

Abstract: In this paper, we study the role *non-adaptivity* plays in maintaining dynamic data structures. Roughly speaking, a data structure is non-adaptive if the memory locations it reads and/or writes when processing a query or update depend only on the query or update and not on the contents of previously read cells. We study such non-adaptive data structures in the cell probe model. The cell probe model is one of the least restrictive lower bound models and in particular, cell probe lower bounds apply to data structures developed in the popular word-RAM model. Unfortunately, this generality comes at a high cost: the highest lower bound proved for any data structure problem is only polylogarithmic (if allowed adaptivity). Our main result is to demonstrate that one can in fact obtain polynomial cell probe lower bounds for non-adaptive data structures.

To shed more light on the seemingly inherent polylogarithmic lower bound barrier, we study several different notions of non-adaptivity and identify key properties that must be dealt with if we are to prove polynomial lower bounds without restrictions on the data structures.

Finally, our results also unveil an interesting connection between data structures and depth-2 circuits. This allows us to translate conjectured hard data structure problems into good candidates for high circuit lower bounds; in particular, in the area of linear circuits for linear operators. Building on lower bound proofs for data structures in slightly more restrictive models, we also present a number of properties of linear operators which we believe are worth investigating in the realm of circuit lower bounds.

ACM Classification: F.1, F.2.3, E.4

AMS Classification: 68Q17, 68Q25, 68P05, 68P30

Key words and phrases: data structures, dynamic data structures, circuit complexity, lower bounds, adaptive, communication complexity, encoding

 DOI: 10.4086/toc.2015.v011a019

1 Introduction

Proving lower bounds on the performance of data structures has been an important line of research for decades. Over time, numerous computational models have been proposed, of which the *cell probe model* of Yao [28] is the least restrictive. Lower bounds proved in this model apply to essentially any imaginable data structure, including those developed in the most popular upper bound model, the word-RAM. Much effort has therefore been spent on deriving cell probe lower bounds for natural data structure problems. Nevertheless, the highest lower bound that has been proved for *any* data structure problem remains just polylogarithmic.

In this paper, we consider a natural restriction of data structures, namely *non-adaptivity*. Roughly speaking, a non-adaptive data structure is a data structure for which the memory locations read when answering a query or processing an update depend only on the query or update itself, and not on the contents of the previously read memory locations. Interestingly, we are able to derive polynomially high cell probe lower bounds for such data structures.

1.1 The cell probe model

In the cell probe model, a data structure consists of a collection of memory cells, each storing w bits. Each cell has an integer address amongst $[2^w] = \{1,\dots,2^w\}$, i. e., we assume any cell has enough bits to address any other cell. When a data structure is presented with a query, the query algorithm starts reading, or *probing*, cells of the memory. The cell probed at each step may depend arbitrarily on the query and the contents of all cells probed so far. After probing a number of cells, the query algorithm terminates with the answer to the query.

A dynamic data structure in the cell probe model must also support updates. When presented with an update, the update algorithm similarly starts reading and/or writing cells of the data structures. We refer jointly to reading or writing a cell as probing the cell. The cell probed at each step, and the contents written to a cell at each step, may again depend arbitrarily on the update operation and the cells probed so far.

The query and update times of a cell probe data structure are defined as the number of cells probed when answering a query or update respectively. We say that a data structure uses S cells of space if it only uses cells with addresses in $[S]$.

1.2 Previous cell probe lower bound techniques

As mentioned, the state-of-the-art techniques for proving cell probe lower bounds unfortunately yield just polylogarithmic bounds. In the following, we give a brief overview of the highest lower bounds that has been achieved since the introduction of the model, and also the most promising line of attack towards polynomial lower bounds.

Static data structures. One of the most important early papers on cell probe lower bounds for static data structures is the paper of Miltersen et al. [19]. They demonstrated an elegant reduction to data structures from an asymmetric communication game. This connection allowed them to obtain lower bounds of the form $t_q = \Omega(\lg m / \lg S)$, where m denotes the number of queries to the data structure problem, S the space usage in number of cells and t_q the query time. Note, however, that this bound is insensitive to polynomial changes in S and cannot give super-constant lower bounds for problems where the number of possible queries is just polynomial in the input size (which is true for most natural problems). This barrier was overcome in the seminal work of Pătraşcu and Thorup [23], who extended the communication game of Miltersen et al. [19] and obtained lower bounds of $t_q = \Omega(\lg m / \lg(St_q/n))$, which peaks at $t_q = \Omega(\lg m / \lg\lg m)$ for data structures using $n\,\mathrm{poly}(\lg m)$ space.

An alternative approach to static lower bounds was given by Panigrahy et al. [20]. Their method is based on sampling the cells of a data structure and showing that many queries can be answered from a small set of cells if the query time is too small (we note that similar ideas have been used for succinct data structure lower bounds, see, e. g., Gál and Miltersen [12]). The maximum lower bounds that can be obtained from this technique are of the form $t_q = \Omega(\lg m / \lg(S/n))$, see Larsen [16]. For linear space, this reaches $t_q = \Omega(\lg m)$, which remains the highest static lower bound to date.

Dynamic data structures. The first technique for proving lower bounds on dynamic data structures was the *chronogram technique* of Fredman and Saks [10]. For data structures with update time t_u, this technique gives lower bounds of the form $t_q = \Omega(\lg n / \lg(wt_u))$ and plays a fundamental role in all later techniques for proving dynamic data structure lower bounds. Pătraşcu and Demaine [22] extended the technique of Fredman and Saks with their *information transfer* technique. This extension allowed for lower bounds of $\max\{t_q, t_u\} = \Omega(\lg n)$. Very recently, Larsen [15] combined the chronogram technique of Fredman and Saks with the cell sampling method of Panigrahy et al. to obtain a lower bound of $t_q = \Omega((\lg n / \lg(wt_u))^2)$, which remains the highest lower bound achieved so far.

Conditional lower bounds. Examining all of the above results, we observe that no lower bound has yet exceeded $\max\{t_u, t_q\} = \Omega((\lg n / \lg\lg n)^2)$ in the most natural case of polynomially many queries, i. e., $m = \mathrm{poly}(n)$. In an attempt to overcome this barrier, Pătraşcu [21] defined a dynamic version of a set disjointness problem, named the *multiphase problem*. We study problems that are closely related to the multiphase problem, so we summarize it here:

The Multiphase Problem. This problem consists of three phases:

- **Phase I:** In this phase, we receive k sets $S_1, \dots, S_k$, each a subset of a universe $[n]$. We are allowed to preprocess these sets into a data structure using time $O(\tau kn)$.
- **Phase II:** We receive another set $T \subseteq [n]$ and have time $O(\tau n)$ to read and update cells of the data structure constructed in Phase I.
- **Phase III:** We receive an index $i \in [k]$ and have time $O(\tau)$ to read cells of the data structure constructed during Phase I and II in order to determine whether $S_i \cap T = \emptyset$.

Pătraşcu conjectured that there exist constants $\mu > 1$ and $\varepsilon > 0$ such that any solution for the multiphase problem must have $\tau = \Omega(n^{\varepsilon})$ when $k = n^{\mu}$, i. e., for the right relationship between n and k, any data structure must have either polynomial preprocessing time, update time or query time. Furthermore, he reduced the multiphase problem to a number of natural data structure problems, including, e. g., the following problems.

- **Reachability in Directed Graphs.** In a preprocessing phase, we are given a directed graph with n nodes and m edges. We are then to support inserting directed edges into the graph. A query is finally specified by two nodes of the graph, u and v, and the goal is to determine whether there exists a directed path from u to v.

- **Subgraph Connectivity.** In a preprocessing phase, we are given an undirected graph with n nodes and m edges. We are then to process updates consisting of turning nodes *on* and *off*. A query is finally specified by two nodes of the graph, u and v, and the goal is to determine whether there exists a path from u to v using only *on* nodes.

We also mention the following problem, which was shown by Chan et al. [4] to solve the multiphase problem.

- **Range Mode.** In a preprocessing phase, we are given an array $A[1:n] = (A[1],\dots,A[n])$ of integers and are to support value updates $A[i] \leftarrow A[i]+x$. Queries are specified by two indices i and j, and the goal is to find the most frequently occurring integer in the subarray $A[i:j]$.

These reductions imply polynomial lower bounds for the above problems, if the multiphase problem has a polynomial lower bound. Thus it seems fair to say that studying the multiphase problem is the most promising direction for obtaining polynomial data structure lower bounds.

1.3 Non-adaptivity

Given that we are generally clueless about how to prove polynomial lower bounds in the cell probe model, it is natural to investigate under what circumstances such bounds can be achieved. In this paper, we study the performance of data structures that are non-adaptive. To make the notion of non-adaptivity precise, we define it in the following:

- **Non-Adaptive Query Algorithm.** A cell probe data structure has a non-adaptive query algorithm, if the cells it probes when answering a query depend only on the query, and not on the contents of previously probed cells.

- **Non-Adaptive Update Algorithm.** Similarly, a cell probe data structure has a non-adaptive update algorithm, if the cells it probes when processing an update depend only on the update, and not on the contents of previously probed cells.

- **Memoryless Update Algorithm.** In this paper, we also study a slightly more restrictive type of update algorithm. A cell probe data structure has a memoryless update algorithm, if the update algorithm is non-adaptive, and furthermore, the contents written to a cell during an update depend only on the update and the current contents of the cell, i. e., they may not depend on the contents of other cells probed during the update operation.[1]

[1]A caveat on the semantics of updates: in this work, we assume updates specify how data *changes* (e. g., updates are of the form $A[k] \leftarrow A[k]+\Delta$) as opposed to specifying new values for data (e. g., updates of the form $A[k] \leftarrow v$). The latter notion goes against the notion of non-adaptive updates, since to rewrite a cell, one must know how an update changes data. One solution is to assume that the data structure stores raw data directly, and to allow memoryless updates to depend on the current contents of a cell, the update, and the previous value of the update. We view this issue as largely semantic, and do not discuss it further.

- **Linear Data Structures.** Finally, we study a subclass of the data structures with a memoryless update algorithm, which we refer to as linear data structures. These data structures are defined for problems where the input can be interpreted as an array A of n bits and an update operation can be interpreted as flipping a bit of A (from 0 to 1 or 1 to 0). A linear data structure has non-adaptive query and update algorithms. Furthermore, when processing an update, the contents of all probed cells are simply flipped, and on a query, the data structure returns the XOR of the bits stored in all the probed cells. Note that these data structures use only a word size of $w = 1$ bit, every cell stores a linear combination of the bits of $A \pmod 2$ and a query again computes a linear combination over the stored linear combinations $\bmod 2$.

While linear data structures might appear to be severely restrictive, for many data structure problems (particularly in the area of range searching), natural solutions are in fact linear. An example is the well-studied *prefix sum problem*, where the goal is to dynamically maintain an array A of bits under flip operations, and a query asks for the XOR of elements in a prefix range $A[1 \dots k]$. One-dimensional range trees are linear data structures that solve prefix sum with update and query time $O(\lg n)$. This is optimal when memory cells store only single bits (as shown by Pătraşcu and Demaine [22]), even for adaptive data structures. More elaborate problems in range searching would be: Given a fixed set P of n points in d-dimensional space, support deleting and re-inserting points of P while answering queries of the form "what is the parity of the number of points inside a given query range?". Here query ranges could be axis-aligned rectangles, halfspaces, simplices, etc. We note that all the known data structures for range counting can easily be modified to yield linear data structures when given a fixed set of points P, and still, this setting seems to capture the hardness of range counting.

The main difference between non-adaptive and memoryless update algorithms is that non-adaptive update algorithms may *move* the information about an update operation around the data structure, even on later updates. As an example, consider a data structure with a non-adaptive update algorithm and two possible updates, say updates u_1 and u_2. Even if the data structure only probes the first memory cell on update u_1, information about u_1 can be stored at many other places in the data structure. Imagine the data structure initially stores the value 0 in the first memory cell. Whenever update u_1 is performed, the data structure increments the contents of the first memory cell by one. On update u_2, the data structure copies the contents of the first memory cell to the second memory cell. Clearly both operations are non-adaptive, and we observe that whenever we have performed update u_2, the second memory cell stores the number of times update u_1 has been performed, even though u_1 never probes the cell. For memoryless updates, information about an update is only stored in cells that are actually probed when processing the update operation.

Linear data structures are inherently memoryless. However, some features possible with memoryless updates are not available to linear data structures. For example, memoryless update algorithms can support cells that maintain a count of the total number of updates executed. This is not possible with linear data structures, since the contents of each cell is a *fixed* linear combination of the data being stored.

1.4 Our results

The main result of this paper, is to demonstrate that polynomial cell probe lower bounds can be achieved when we restrict data structures to be non-adaptive. First, we prove lower bounds for data structures where only the query algorithm is non-adaptive. The concrete data structure problem that we study in this setting is the following indexing problem.

Indexing Problem. In a preprocessing phase, we receive a set of k binary strings $S_1,\dots,S_k$, each of length n. We are then to support updates, consisting of an index $j \in [n]$, which we think of as an index into the strings $S_1,\dots,S_k$. A query is finally specified by an index $i \in [k]$ and the goal is to return the j-th bit of S_i.

Theorem 1.1. *Any cell probe data structure solving the indexing problem with a non-adaptive query algorithm must either have $t_q = \Omega(n/w)$ or $t_u = \Omega(k/w)$, regardless of the preprocessing time and space usage.*

Examining this problem, one quickly observes that it is a special case of the multiphase problem presented in Section 1.2 (set $T = \{j\}$), thus by setting the parameters in the reductions of Pătraşcu [21] and Chan et al. [4] correctly we obtain, amongst others, the following lower bounds as an immediate corollary of our lower bound for the indexing problem:

Corollary 1.2. *Any cell probe data structure that uses a non-adaptive query algorithm to solve (i) reachability in directed graphs or (ii) subgraph connectivity must either have $t_q = \Omega(n/w)$ or $t_u = \Omega(n/w)$. Any cell probe data structure that solves range mode with a non-adaptive query algorithm must have $t_q t_u = \Omega(n/w^2)$.*

We also prove lower bounds for data structures where the query algorithm is allowed to be adaptive, but the update algorithm is memoryless. Again, we prove our lower bound for a special case of the multiphase problem:

Set Disjointness Problem. In a preprocessing phase, we receive a subset S of a universe $[n]$. We are then to support inserting elements $x \in [n]$ into an initially empty set T. Finally a query simply asks to return whether $S \cap T = \emptyset$, i. e., the problem has just one query.

Theorem 1.3. *Any cell probe data structure solving the set disjointness problem with a memoryless update algorithm must have $t_q = \Omega(n/w)$, regardless of the preprocessing time, space usage and update time.*

Again, using the reductions of Pătraşcu [21] and Chan et al. [4], we obtain the following lower bounds as a corollary of our lower bound for the set disjointness problem:

Corollary 1.4. *Any cell probe data structure that uses a memoryless update algorithm to solve (i) reachability in directed graphs, (ii) subgraph connectivity, or (iii) range mode must have $t_q = \Omega(n/w)$.*

The lower bounds described above appear in Section 2. In Section 3, we show a strong connection between nonadaptive data structures and the wire complexity of depth-2 circuits. In these circuits, gates have *unbounded* fan-in and fan-out and compute *arbitrary* functions. Thus, trivial bounds on the number of gates exist. Instead, the size of a circuit $s(C)$ is defined to be the number of wires.

Proving lower bounds on the wire complexity of circuits computing explicit functions $F : \{0,1\}^n \to \{0,1\}^m$ has been studied in several works. In particular, Valiant [26] showed that an $\omega(n^2/(\lg\lg n))$

bound on the number of wires in circuits computing F implies that F cannot be computed by log-depth, linear size, bounded fan-in circuits. Currently, the best bounds known for an explicit function are $\Omega(n^{3/2})$. Cherukhin [9] gave such a bound for circuits computing cyclic convolutions. Jukna [13] gave a similar lower bound for circuits computing matrix multiplication, and developed a general technique for proving such lower bounds, formalizing the intuition of Cherukhin [9].

First, we show how to use simple encoding arguments common to data structure lower bounds to achieve circuit lower bounds, using matrix multiplication as an example. Our bound matches Jukna's result, but yields a simpler argument. We discuss Jukna's technique in more detail in Section 3.

Theorem 1.5 (Jukna [13])**.** *Any circuit computing matrix multiplication of two $\sqrt{n} \times \sqrt{n}$ matrices has wire complexity at least $n^{3/2}$.*

Depth-2 circuits computing explicit *linear* functions (henceforth referred to as *linear operators*) are of particular interest. Currently, the best lower bound for an explicit linear operator is the recent $\Theta(n(\lg n/\lg\lg n)^2)$ bound of Gál et al. [11] for circuits that compute error correcting codes. Another interesting question is whether general circuits are more powerful than *linear* circuits for computing linear operators. Linear circuits use only XOR gates; i. e., each gate outputs a linear combination in $\mathbf{GF}(2)$ over its inputs.

We show a generic connection between linear data structures and linear circuits. Define a problem $\mathcal{P}$ as a mapping $F_{\mathcal{P}} = (f_1,\ldots,f_m) : \{0,1\}^n \to \{0,1\}^m$, where each $f_j : \{0,1\}^n \to \{0,1\}$. For linear data structures, think of the domain $\{0,1\}^n$ as the input array A with n bits, and view each f_j as a query, where $f_j(A)$ is the answer to the query f_j on the input A. A linear data structure hence solves $\mathcal{P}$, if after any sequence of updates to A, it holds for all $1 \le j \le m$ that answering the query f_j returns the value $f_j(A)$.

Lemma 1.6. *If there is a linear data structure for a problem $\mathcal{P}$ with average query time of t_q and average update time t_u, then there exists a depth-2 linear circuit C computing $F_{\mathcal{P}}$ with $s(C) \le nt_u + mt_q$ wires.*

If there is a depth-2 linear circuit C that computes $F_{\mathcal{P}}$, then there is a linear data structure for $\mathcal{P}$ with average query time at most $s(C)/m$ and average update time at most $s(C)/n$.

In the above, *average* query time (update time) is defined as the average of the query time (update time) over all queries (updates).

Lemma 1.6 thus gives a new way to attack circuit lower bounds. We believe the connection between non-adaptive data structures and depth-2 circuits has the potential to yield strong insight to this problem, and that several linear operators conjectured to have strong data structure lower bounds are good candidates for hard circuit problems (for linear or general circuits).

Apart from being interesting lower bounds in their own right, we believe our results shed much light on the inherent difficulties of proving polynomial lower bounds in the cell probe model. In particular the *movement* of data when performing updates (see the discussion in Section 1.3) appears to be a major obstacle. We conclude in Section 4 with a discussion of our results and potential directions for future research.

1.5 Techniques

We employ two standard techniques for proving data structure lower bounds: *encoding* and *communication complexity*. In a typical encoding proof, one shows how to use an efficient data structure to encode an

n-bit string x. A decoder must be able to take the encoding and unambiguously recover the original string; any such encoding naturally requires n bits. If the data structure-based encoding of x uses, say, $t_q w$ bits, one concludes $t_q \geq n/w$.

Communication complexity, introduced by Yao [27], has become one of the primary tools for proving lower bounds in several areas of computer science. In a typical communication problem, two players named Alice and Bob receive inputs $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ respectively and wish to jointly compute some function $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ using the minimum amount of communication possible. A *protocol* π is a distributed algorithm which specifies how the players communicate—π specifies for all possible (x, y) who is to speak first, what messages get sent, and when the protocol ends. At the end of a protocol, Alice and Bob output $\pi(x, y) \in \mathcal{Z}$. We say that π computes f if $\pi(x, y) = f(x, y)$ for all x, y. The communication cost of π on inputs (x, y), denoted $\text{cost}(\pi; x, y)$, is the total number of bits exchanged. The communication cost of π, denoted $\text{cost}(\pi)$, is the maximal $\text{cost}(\pi; x, y)$ taken over all (x, y). The communication complexity of a function f is the *minimal* $\text{cost}(\pi)$, taken over all protocols which compute f. Often, it is useful to consider *randomized* protocols. In this setting, Alice and Bob additionally have access to a shared random string $R \in \{0, 1\}^*$ they can use to help select what messages to send. When communication ends, Alice and Bob again output some $\pi(x, y, R) \in \mathcal{Z}$. A randomized protocol computes f if for all (x, y), $\pi(x, y, R) = f(x, y)$ with probability at least $2/3$, where the probability is taken over the shared random string R. The cost of a randomized protocol is the maximum number of bits sent, taken over all (x, y) and all possible values of R. The randomized communication complexity of f is the minimum $\text{cost}(\pi)$ taken over all randomized protocols π computing f.

In this work, we use the Set Disjointness communication problem DISJ. In this problem, each player's input is a subset of $\{1, \ldots, n\}$, and the players need to compute whether their sets intersect. DISJ has a long history and is one of the cornerstones of communication complexity. While we will not need randomized communication complexity in our proofs, we note that the following theorem holds also for communication protocols that are allowed to err with probability $1/3$:

Theorem 1.7 (Babai et al. [2], Kalyanasundaram and Schnitger [14], Razborov [24], Bar-Yossef et al. [3])**.**

DISJ *requires* $\Omega(n)$ *communication.*

We use encoding to prove Theorem 1.1 and Theorem 1.5 and communication complexity to prove Theorem 1.3. We view our lower bound proofs not as great technical advancements, but as clean, simple applications of current lower bound techniques. Furthermore, given the new connection we establish between nonadaptive data structures and depth-2 circuit lower bounds, we hope that our lower bounds illustrate the power of these techniques and that our work is a good demonstration of how encoding can be used to prove circuit lower bounds.

Pătraşcu, in his paper establishing the multiphase problem [21], also proposed a certain three-player communication complexity variant of the multiphase problem, whose conjectured direct-sum style lower bounds imply lower bounds for the data structure version of multiphase (and consequently strong dynamic data structure lower bounds). Furthermore, in a follow-up paper, Chattopadhyay et al. [5] refute the strongest conjectures for the communication complexity of multiphase, but not in a way that prevents strong data structure lower bounds. Additionally, Chattopadhyay et al. [5] provide stronger lower bounds for the multiphase problem in restricted settings. Their restrictions limit the amount of interaction between players. As pointed out by an anonymous referee in a previous draft of this work, suitable extensions of their proof suffice to prove Theorem 1.1; we view our proof as simpler and more direct.

2 Lower bounds

In this section, we first prove lower bounds for data structures where only the query algorithm is assumed non-adaptive. The problem we study is the indexing problem defined in Section 1.4.

Theorem 2.1 (Restatement of Theorem 1.1)**.** *Any cell probe data structure that solves the indexing problem with a non-adaptive query algorithm must either have $t_q = \Omega(n/w)$ or $t_u = \Omega(k/w)$, regardless of the preprocessing time and space usage. Here t_q denotes the query time, t_u the update time and w the cell size in bits.*

We prove this using an encoding argument. Specifically, consider a game between an encoder and a decoder. The encoder receives as input k binary strings $S_1,\dots,S_k$, each of length n and must from this send a message to the decoder. From the message alone, the decoder must uniquely recover all the strings $S_1,\dots,S_k$. If the strings $S_1,\dots,S_k$ are drawn from a distribution, then by Shannon's source coding theorem [25], the expected length of the message must be at least $H(S_1\cdots S_k)$ bits, where $H(\cdot)$ denotes binary Shannon entropy.

The idea in our proof is to assume for contradiction that a data structure for the indexing problem exists with a non-adaptive query algorithm that simultaneously has $t_q = o(n/w)$ and $t_u = o(k/w)$. Using this data structure as a black box, we construct a message that is shorter than $H(S_1\cdots S_k)$ bits, but at the same time, the decoder can recover $S_1,\dots,S_k$ from the message, i. e., we have reached the contradiction. We let the k strings $S_1,\dots,S_k$ given as input to the encoder be uniform random bit strings of length n. Clearly $H(S_1\cdots S_k) = kn$.

Encoding procedure. When given the strings $S_1,\dots,S_k$ as input, the encoder first runs the preprocessing algorithm of the claimed data structure on $S_1,\dots,S_k$. He then examines every possible query index $i\in[k]$, and for each i, collects the set of addresses of the cells probed on query i. Since the query algorithm is non-adaptive, these sets of addresses are independent of $S_1,\dots,S_k$ and any updates we might perform on the data structure. Letting C denote the set containing all these addresses for all i, the encoder starts by writing down the concatenation of the contents of all cells with an address in C. This constitutes the first part of the message.

The encoder now runs through every possible update $j\in[n]$. For each j, he runs the update algorithm as if update j had been performed on the data structure. While running update j, the decoder appends the contents of the probed cells (as they are when the update reads the cells, not after potential changes) to the constructed message. After processing all updates j, the encoder finally sends the constructed message to the decoder. This completes the encoding procedure.

Decoding procedure. The decoder receives as input the message consisting first of the contents of all cells with an address in C after preprocessing $S_1,\dots,S_k$. Since the query algorithm is non-adaptive, the decoder knows the addresses of all these cells simply by examining the query algorithm of the claimed data structure. The decoder will now run the update algorithm of every $j \in [n]$. While doing this, he maintains the contents of all cells in C and all cells probed during the updates. Specifically, the decoder does the following:

For each $j = 1, \dots, n$ in turn, he starts to run the update algorithm for j. Observe that the contents of each probed cell (before potential changes) can be recovered from the message (the contents appear one after another in the message). This allows the decoder to completely simulate the update algorithm for each $j = 1, \dots, n$. Note furthermore that for each cell that is probed during these updates, the address can also be recovered simply by examining the update algorithm. In this way, the decoder always knows the contents of all cells in C and all cells probed by the update algorithm as they would have been after preprocessing $S_1, \dots, S_k$ and performing the updates after this preprocessing. While processing the updates $j = 1, \dots, n$, the decoder also executes a number of queries: After having completely processed an update j, the decoder runs the query algorithm for every $i \in [k]$. Note that the decoder knows the contents of all the probed cells as if the preprocessing on $S_1, \dots, S_k$ had been performed, followed by updates $j' = 1, \dots, j$. This implies that the simulation of the query algorithm for each $i \in [k]$ terminates precisely with the answer being the j-th bit of S_i. It follows immediately that the decoder can recover every bit of every S_i from the message.

Analysis. What remains is to analyze the size of the message. Since by assumption, the query time is $t_q = o(n/w)$, the first part of the message has $t_q kw = o(kn)$ bits. Similarly, we assumed $t_u = o(k/w)$, thus the second part of the message has $t_u nw = o(kn)$ bits. Thus the entire message has $o(kn)$ bits. Since $H(S_1 \cdots S_k) = kn$, we have reached our contradiction. This completes the proof of Theorem 1.1.

Next, we prove lower bounds for data structures where only the update algorithm is assumed to be memoryless; that is, we allow the query algorithm to be adaptive. In this setting, we study the set disjointness problem defined in Section 1.4:

Theorem 2.2 (Restatement of Theorem 1.3)**.** *Any cell probe data structure that solves the set disjointness problem with a memoryless update algorithm must have $t_q = \Omega(n/w)$, regardless of the preprocessing time, space usage and update time. Here t_q denotes the query time and w the cell size in bits.*

Proof. We reduce from the set disjointness communication problem DISJ. Recall that in this problem, Alice and Bob are given sets $A, B \subseteq \{1, \dots, n\}$ respectively and must communicate to decide whether A and B are disjoint. As mentioned previously, DISJ requires $\Omega(n)$ communication [14, 24, 3], even with randomness.

To create a protocol for DISJ, fix an arbitrary data structure for the Set Disjointness problem that uses memoryless updates. Given inputs $A, B \subseteq \{1, \dots, n\}$, Alice and Bob emulate the data structure in the following way. First, they preprocess B, then insert elements of A before finally querying to determine whether A and B are disjoint.

To perform the emulation in a communication-efficient manner, Bob first preprocesses B. Then Alice starts to run the query algorithm. For each cell probed, she asks Bob for the contents of the cell as it was immediately after preprocessing B. Since the update algorithm is memoryless, Alice can now run the update algorithm on A and check whether the cell contents returned by Bob change upon inserting elements from A. If they *do* change, Alice continues the simulation with the updated contents; otherwise, she uses the contents simply as they were. Alice and Bob continue in this manner for each probe of the query algorithm until all probes have been performed. Since each cell used in Alice's simulation has the same contents as if A had been inserted after preprocessing B, the answer will be precisely whether A intersects B. Each cell probed by the query algorithm results in $2w$ bits of communication. Therefore, the total communication cost is $2wt_q$. Since DISJ requires $\Omega(n)$ bits of communication, it follows that $t_q = \Omega(n/w)$. □

Again, we prove this using an encoding argument. In this encoding proof, we let the input of the encoder be a uniform random set $S \subseteq [n]$. Clearly $H(S) = n$ bits. We now assume for contradiction that there exists a data structure for the set disjointness problem with a memoryless update algorithm and at the same it has query time $t_q = o(n/w)$. The encoder uses this data structure to send a message encoding S in less than n bits, i. e., a contradiction.

Encoding procedure. When the encoder receives S, he runs the preprocessing algorithm of the claimed data structure. Then, he computes $\bar{S} = [n] \setminus S$ and inserts $\bar{S}$ into the data structure as the set T. Finally, the encoder runs the query algorithm and notes the set of cells C probed. Note that by the choice of $\bar{S}$, the query algorithm will output *disjoint*, and furthermore, $\bar{S}$ is the largest possible set that will result in a *disjoint* answer.

The encoding consists of three parts:[2] (i) the addresses of the cells in C, (ii) the contents of the cells in C after preprocessing but before inserting $\bar{S}$, and (iii) the contents of the cells in C after inserting $\bar{S}$.

Decoding procedure. The decoder iterates over all sets $S' \subseteq [n]$. Each time, the decoder initializes the contents of cells in C to match the second part of the encoder's message. Then, he inserts each element of S' into the data structure, changing the contents of any cell in C where appropriate. When a cell outside of C is to be changed, the decoder does nothing. Since the update algorithm is memoryless, this procedure ends with all cells in C having the same contents as they would have had after preprocessing S and inserting elements of S'. Moreover, if the contents match the contents written down in the third part of the encoding, then it must be that S and S' are disjoint (we know that the query answers *disjoint* when the contents of C are like that). When $S' = \bar{S}$, the contents of C will match the last part of the encoding, and it is trivially the largest set to do so. Thus, the decoder selects the largest set S^* whose updates to C match the contents written down in the third part of the encoding. In this way, the decoder recovers $S = [n] \setminus S^*$.

Analysis. Finally, we analyze the size of the encoding. Since we assumed $t_q = o(n/w)$, the encoding has size $3t_q w = o(n)$ bits. But $H(S) = n$, thus we have reached a contradiction.

3 Circuits and non-adaptive data structures

In this section, we demonstrate a strong connection between non-adaptive data structures and the wire complexity of depth-2 circuits. A depth-2 circuit computing $F = (f_1, \dots, f_m) : \{0,1\}^n \to \{0,1\}^m$

[2] In fact, it is possible for the decoder to recover C from the second two parts of the encoding, so the first part is unnecessary. However, this does not materially affect our lower bound, so we omit the details.

is a directed graph with three layers of vertices. The first layer consists of n input nodes, labeled $x_1,\ldots,x_n \in \{0,1\}$. Vertices in the second layer are interior gates and output Boolean values. The last layer consists of m output gates, labeled $z_1,\ldots,z_m \in \{0,1\}$. There are edges between input nodes and interior gates, and between interior gates and output gates. Each gate computes an *arbitrary* function of its inputs. Since non-input nodes compute arbitrary functions, f can be trivially computed using m gates. Instead, we define the *size* $s(C)$ of a depth-2 circuit C as the total number of wires in it; i. e., the number of edges in the graph.

First, we show how to use the encoding technique common to data structure lower bounds to achieve size bounds for depth-2 circuits. As a proof of concept, we prove such a lower bound for matrix multiplication. We say that a circuit computes matrix multiplication if there are $2n$ inputs, each corresponding to an entry in one of two $\sqrt{n}\times\sqrt{n}$ binary matrices A and B, and each output gate computes an entry in the product $A\cdot B$. Arithmetic is in $\mathbf{GF}(2)$.

Jukna [13] considered depth-2 circuits and gave an $n^{3/2}$ lower bound for circuits computing Boolean matrix multiplication. At a high level, his proof proceeds in the following fashion.

1. Partition input nodes into sets $I_1,\ldots,I_t$ and output gates into sets $J_1,\ldots,J_t$.
2. Prove that for each $1\le \ell \le t$, the number of wires leaving inputs from I_ℓ plus the number of wires entering outputs in J_ℓ must be large.
3. Conclude a large lower bound by summing the terms from Step 2.

Note that since $\{I_\ell\}$ and $\{J_\ell\}$ are partitions, they induce a partition on the wires in the circuit. Jukna proves Step 2 by proving lower bounds on what he calls the *entropy* of an operator. He proves a lower bound on the entropy of an operator by carefully analyzing subfunctions of the operator. In the case of matrix multiplication, subfunctions are created by fixing entries in B to be all zero, except for a single cell $B[k,\ell]$. Each I_ℓ, J_ℓ represents a column in B and in $A\cdot B$ respectively. By ranging over different k,ℓ, Jukna is able to argue that the entropy of matrix multiplication is high. The details of this argument are technical.

We give a new proof for Step 2 using an encoding argument. The encoder exploits the circuit operations to encode a $\sqrt{n}\times\sqrt{n}$ matrix A. The encoded message has length precisely equal to the number of outgoing wires in I_ℓ and incoming wires to J_ℓ. The argument is very similar to the arguments in Section 2.

Theorem 3.1 (Restatement of Theorem 1.5)**.** *Any circuit C computing Boolean matrix multiplication has size $s(C)\ge n^{3/2}$.*

Proof. Fix a circuit C. Let $P = A\cdot B$. For $1\le \ell\le \sqrt{n}$, let I_ℓ denote the ℓ-th column of B; that is, I_ℓ consists of all inputs corresponding to $B[k,\ell]$ for some k. Similarly, J_ℓ is the set of all outputs corresponding to the ℓ-th column of P; that is, all outputs given by $P[k,\ell]$ for some k. Let $t_{u,\ell}$ denote the number of wires leaving inputs in I_ℓ. Similarly, let $t_{q,\ell}$ denote the number of wires entering outputs in J_ℓ.

Claim 3.2. *For any ℓ, we have $t_{u,\ell}+t_{q,\ell}\ge n$.*

Before proving this claim, note that Theorem 3.1 follows directly, since there are $\sqrt{n}$ pairs (I_ℓ,J_ℓ) and the wires corresponding to each pair are disjoint. □

Proof. This proof will involve an encoding argument. The encoder will receive a $\sqrt{n} \times \sqrt{n}$ Boolean matrix A, where A is drawn uniformly amongst all such Boolean matrices. He will then use the matrix multiplication circuit to encode A in such a way that the size of the encoding depends on the wires leaving I_ℓ and entering J_ℓ.

Encoding procedure. The encoder receives A and treats A as the first half of the input to the circuit C. As a first step, he sets all entries in B to zero. He then writes down the output of all interior gates adjacent to an output in J_ℓ. In the second step, for each $1 \le k \le \sqrt{n}$, the encoder performs the following: he sets $B[k,\ell] \leftarrow 1$ and sets all other entries in B to zero. He then writes down the output of all interior gates adjacent to $B[k,\ell]$. This completes the encoding procedure.

Decoding procedure. Note that $P[i,\ell] = \sum_j A[i,j]B[j,\ell]$. In particular, when B consists of a 1 in entry $[k,\ell]$ and zero in all other entries, then the ℓ-th column of P corresponds to the k-th column of A. The decoder thus recovers the k-th column of A by using C to compute the ℓ-th column of P, i. e., by querying all outputs in J_ℓ. For each output gate in J_ℓ, she looks at all interior gates adjacent to it. For each of *these* gates g, the decoder checks to see if g is adjacent to the input gate $B[k,\ell]$. If so, then she recovers the correct output value of this gate from the second part of the encoding. Otherwise, she recovers the correct output from the first part (noting that in this case, changing the value of $B[k,\ell]$ does not affect g). In this way, the decoder recovers the ℓ-th column of P, which is also the k-th column of A. Doing this for all k completes the decoding.

Analysis. The first part of the encoding consists of the output of each interior gate adjacent to at least one output in J_ℓ. Thus, the first part of the encoding can be described in at most $t_{q,\ell}$ bits. The second part of the encoding consists of the output of each interior gate adjacent to each input node in I_ℓ. This requires at most $t_{u,\ell}$ bits. Thus, the total length of the encoding is at most $t_{u,\ell} + t_{q,\ell}$. The decoder recovers all of A from this message. Since each entry of A is independent and uniform, $H(A) = n$. Thus, $t_{u,\ell} + t_{q,\ell} \ge n$. □

Remark. As mentioned previously, Jukna proves his lower bounds by defining the *entropy* of an operator. He lower bounds the wire complexity of a circuit by the entropy of the operator it computes. He proves a lower bound on the entropy of an operator by carefully analyzing subfunctions of the operator, created by fixing subsets of the variables to specific values and considering the induced function on the remaining variables.

Parts of our proof are similar in spirit to Jukna's proof. In particular, the way we encode M by fixing the matrix B to be one in entry $[k,\ell]$ and zero elsewhere corresponds to the subfunctions Jukna considers in his proof. In fact, we believe that *any* lower bound provable using Jukna's technique can also be proved using our method. Our advantage is in replacing Jukna's technical and somewhat complicated machinery with a simple encoding argument.

3.1 Linear circuits and linear data structures

In this section, we provide a strong connection between depth-2 linear circuits and linear data structures. The connection is almost immediately established:

Lemma 3.3 (Restatement of Lemma 1.6)**.** *If there is a linear data structure for a problem $\mathcal{P}$ with average query time of t_q and average update time t_u, then there exists a depth-2 linear circuit C computing $F_{\mathcal{P}}$ with size $s(C) \le nt_u + mt_q$.*

If there is a depth-2 linear circuit C computing $F_{\mathcal{P}}$, then there is a linear data structure for $\mathcal{P}$ with average query time at most $s(C)/m$ and average update time at most $s(C)/n$.

Proof. First, suppose there exists a linear data structure solving $\mathcal{P}$. We construct the corresponding depth-2 circuit directly. Input nodes correspond to the n bits of the input (the array A in the definition of linear data structures). Output nodes correspond to the m possible queries, and there is an interior node for each cell in the database. For each update $1 \le i \le n$ (flip an entry of A), add edges from x_i to each of the cells updated by the data structure. Similarly, add wires (c_i, z_j) whenever the j-th query probes the i-th cell in the data structure. Correctness follows immediately. Finally, note that since the average update time and query time is t_u and t_q respectively, the total number of wires in the circuit is bounded by $s(C) \le nt_u + mt_q$.

Constructing a linear data structure from a linear depth-2 circuit C is similar. Letting $t_{u,i}$ and $t_{q,j}$ denote the number of cells probed during the i-th update and j-th query respectively, it is easy to see that

$$s(C) = \sum_{i=1}^{n} t_{u,i} + \sum_{j=1}^{m} t_{q,j} .$$

It follows that the average update time is at most

$$\frac{1}{n} \sum t_{u,i} \le s(C)/n ,$$

and similarly that the average query time is at most

$$\frac{1}{m} \sum t_{q,j} \le s(C)/m . \qquad \square$$

The main contribution of Lemma 3.3 is a new range of candidate hard problems for linear circuits, all inspired by data structure problems. As mentioned in Section 1.3, linear data structures most naturally occur in the field of range searching. Furthermore, these data structure problems turn out to correspond precisely to linear operators: Let $P = \{p_1, \ldots, p_n\}$ be a fixed set of n points in $\mathbb{R}^d$, and let $\mathcal{R}$ be a set of query ranges, where each $R_i \in \mathcal{R}$ is a subset of $\mathbb{R}^d$. P and $\mathcal{R}$ naturally define a linear operator

$$A(P, \mathcal{R}) \in \{0,1\}^{|\mathcal{R}| \times |P|} ,$$

where the i-th row of $A(P, \mathcal{R})$ has a 1 in the j-th column if $p_j \in R_i$ and 0 otherwise. The matrix $A(P, \mathcal{R})$ is referred to as the incidence matrix of P and $\mathcal{R}$. In the light of Lemma 3.3, assume a linear data structure solves the following range counting problem: Given the fixed set of points P, each assigned a weight in $\{0,1\}$, support flipping the weights of the points (intuitively inserting/deleting the points) while also supporting to efficiently compute the parity of the weights assigned to the points inside a query range $R_i \in \mathcal{R}$. Then that linear data structure immediately translates into a linear circuit for the linear operator $A(P, \mathcal{R})$ and vice versa. Thus we expect that hard range searching problems of the above form also provide hard linear operators for linear circuits. The seemingly hardest range searching problem is *simplex range searching* (a *simplex* is simply the generalization of a triangle to higher dimensions and is formally defined as the convex hull of $d+1$ points in $\mathbb{R}^d$), where we believe that the following holds:

Conjecture 3.4. *There exists a constant $\varepsilon > 0$, a set $\mathcal{R}$ of $\Theta(n)$ simplices in $\mathbb{R}^d$ and a set of n points in $\mathbb{R}^d$, such that any data structure solving the above range counting problem (flip weights, parity queries), must have average query and update times satisfying $t_u t_q = \Omega(n^{\varepsilon})$.*

We have toned down Conjecture 3.4 somewhat, since the community generally believes ε can be replaced by $1-1/d$, but to be on the safe side we only conjecture the above. In the circuit setting, this conjecture translates to

Corollary 3.5. *If Conjecture 3.4 is true for linear data structures, then there exists a constant $\delta > 0$, a set $\mathcal{R}$ of $\Theta(n)$ simplices in $\mathbb{R}^d$ and a set P of n points, such that any linear circuit computing the linear operator $A(P,\mathcal{R})$ must have $\Omega(n^{1+\delta})$ wires.*

Note that a lower bound of $\Omega(n^{1+\delta})$ would be a major result for any constant $\delta > 0$ since the highest current lower bound for linear operators is the $\Omega(n(\lg n/\lg\lg n)^2)$ bound of Gál et al. [11].

Furthermore, the research on data structure lower bounds also provides a lot of insight into which concrete sets P and $\mathcal{R}$ might be difficult. More specifically, polynomial lower bounds for simplex range searching have been proved for: range reporting in the pointer machine (Chazelle and Rosenberg [8], Afshani [1]) and I/O-model (Afshani [1]), range searching in the semigroup model (Chazelle [6]) and range searching in the group model (Larsen [17], Larsen and Nguyen [18]). The group model comes closest in spirit to linear data structures. In the group model, each input point is assigned a weight from a group and a data structure is allowed to store linear combinations (with integer coefficients) over these weights. An update consists of changing the weight of a point. When given a query range, the data structure must add and subtract the stored group sums to yield precisely the sum of the weights assigned to points in the query range. Thus group model data structures are just like linear data structures, except that instead of storing linear combinations over $\mathbf{GF}(2)$, they store linear combinations with integer coefficients (and no mod operations). Similarly, queries are answered by computing linear combinations over the stored elements, but with integer coefficients and not over $\mathbf{GF}(2)$. In the following, we list properties that lead to high lower bounds in the group model:

- If $A(P,\mathcal{R})$ has $\Omega(n^{\varepsilon})$ *red-blue discrepancy* for some constant $\varepsilon > 0$, then any group model data structure must have $t_u t_q = n^{\Omega(\varepsilon)}$. Here red-blue discrepancy is defined as the minimum over all 2-colorings of the points, of the largest deviation in number of points of the two colors in any range, i. e., the discrepancy of $A(P,\mathcal{R})$ is
$$\min_{\chi:P\to\{-1,1\}} \max_{R\in\mathcal{R}} \left| \sum_{p\in P\cap R} \chi(p) \right| .$$
- If $|R_i \cap P| = \Omega(n^{\varepsilon})$ for all $R_i \in \mathcal{R}$ and $|R_i \cap R_j \cap P| = O(1)$ for all $i \neq j$, then any group model data structure must have $t_u t_q = n^{\Omega(\varepsilon)}$.

The last property directly translates to $A(P,\mathcal{R})$ having rows and columns with polynomially many 1s and any two rows/columns having at most a constant number of 1s in common. Given the tight correspondence between group model data structures and linear data structures, we believe these properties are worth investigating in the circuit setting. Furthermore, a concrete set of n points P and a set of $\Theta(n)$ simplices $\mathcal{R}$, with both properties, is known even in $\mathbb{R}^2$. This example can be found in the work of Chazelle and Liu [7], where it is stated for $\mathcal{R}$ being lines (i. e., degenerate simplices). Note that the lower bound of Chazelle and Lui [7] is for range reporting in the pointer machine, but using the observations in Larsen [17] and Larsen and Nguyen [18] it is easily seen that the above properties hold.

Even if these properties are not enough to obtain lower bounds for linear operators, we believe the geometric approach might be useful in its own right.

4 Conclusion

In this paper, we have studied the role non-adaptivity plays in dynamic data structures. Surprisingly, we were able to prove polynomially high lower bounds for such data structures. Perhaps more importantly, we believe our results shed much new light on the current polylogarithmic barriers if we do not make any restrictions on data structures. We also presented an interesting connection between data structures and depth-2 circuits. The connection between linear operators and range searching is particularly intriguing, revealing a number of new properties to investigate further in the realm of circuit lower bounds.

Acknowledgments

We are grateful to Elad Verbin for several helpful discussions, and to anonymous reviewers for useful comments.

References

[1] PEYMAN AFSHANI: Improved pointer machine and I/O lower bounds for simplex range reporting and related problems. *Int. J. Comput. Geometry Appl.*, 23(4-5):233–252, 2013. Preliminary version in SCG'12. [doi:10.1142/S0218195913600054] 485

[2] LÁSZLÓ BABAI, PETER FRANKL, AND JANOS SIMON: Complexity classes in communication complexity theory (preliminary version). In *Proc. 27th FOCS*, pp. 337–347. IEEE Comp. Soc. Press, 1986. [doi:10.1109/SFCS.1986.15] 478

[3] ZIV BAR-YOSSEF, THATHACHAR S. JAYRAM, RAVI KUMAR, AND D. SIVAKUMAR: An information statistics approach to data stream and communication complexity. *J. Comput. System Sci.*, 68(4):702–732, 2004. Preliminary version in FOCS'02. [doi:10.1016/j.jcss.2003.11.006] 478, 480

[4] TIMOTHY M. CHAN, STEPHANE DUROCHER, KASPER GREEN LARSEN, JASON MORRISON, AND BRYAN T. WILKINSON: Linear-space data structures for range mode query in arrays. *Theory Comput. Syst.*, 55(4):719–741, 2014. Preliminary version in STACS'12. [doi:10.1007/s00224-013-9455-2] 474, 476

[5] ARKADEV CHATTOPADHYAY, JEFF EDMONDS, FAITH ELLEN, AND TONIANN PITASSI: A little advice can be very helpful. In *Proc. 23rd Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'12)*, pp. 615–625. SIAM, 2012. ACM DL. 478

[6] BERNARD CHAZELLE: Lower bounds on the complexity of polytope range searching. *J. Amer. Math. Soc.*, 2(4):637–666, 1989. Preliminary version in ICALP'03. [doi:10.1090/S0894-0347-1989-1001852-0] 485

[7] BERNARD CHAZELLE AND DING LIU: Lower bounds for intersection searching and fractional cascading in higher dimension. *J. Comput. System Sci.*, 68(2):269–284, 2004. Preliminary version in STOC'01. [doi:10.1016/j.jcss.2003.07.003] 486

[8] BERNARD CHAZELLE AND BURTON ROSENBERG: Simplex range reporting on a pointer machine. *Comput. Geom.*, 5(5):237–247, 1996. [doi:10.1016/0925-7721(95)00002-X] 485

[9] DMITRIY YU. CHERUKHIN: The lower estimate of complexity in the class of schemes of depth 2 without restrictions on a basis. *Vestnik Moscow University Series 1, Mathematika*, 60(4):54–56, 2005. 477

[10] MICHAEL L. FREDMAN AND MICHAEL E. SAKS: The cell probe complexity of dynamic data structures. In *Proc. 21st STOC*, pp. 345–354. ACM Press, 1989. [doi:10.1145/73007.73040] 473

[11] ANNA GÁL, KRISTOFFER ARNSFELT HANSEN, MICHAL KOUCKÝ, PAVEL PUDLÁK, AND EMANUELE VIOLA: Tight bounds on computing error-correcting codes by bounded-depth circuits with arbitrary gates. *IEEE Trans. Inform. Theory*, 59(10):6611–6627, 2013. Preliminary version in STOC'12. [doi:10.1109/TIT.2013.2270275] 477, 485

[12] ANNA GÁL AND PETER BRO MILTERSEN: The cell probe complexity of succinct data structures. *Theoret. Comput. Sci.*, 379(3):405–417, 2007. Preliminary version in ICALP'03. [doi:10.1016/j.tcs.2007.02.047] 473

[13] STASYS JUKNA: Entropy of operators or why matrix multiplication is hard for depth-two circuits. *Theory Comput. Syst.*, 46(2):301–310, 2010. [doi:10.1007/s00224-008-9133-y] 477, 482

[14] BALA KALYANASUNDARAM AND GEORG SCHNITGER: The probabilistic communication complexity of set intersection. *SIAM J. Discrete Math.*, 5(4):545–557, 1992. Preliminary version in SCTC'87. [doi:10.1137/0405044] 478, 480

[15] KASPER GREEN LARSEN: The cell probe complexity of dynamic range counting. In *Proc. 44th STOC*, pp. 85–94. ACM Press, 2012. [doi:10.1145/2213977.2213987, arXiv:1105.5933] 473

[16] KASPER GREEN LARSEN: Higher cell probe lower bounds for evaluating polynomials. In *Proc. 53rd FOCS*, pp. 293–301. IEEE Comp. Soc. Press, 2012. [doi:10.1109/FOCS.2012.21] 473

[17] KASPER GREEN LARSEN: On range searching in the group model and combinatorial discrepancy. *SIAM J. Comput.*, 43(2):673–686, 2014. Preliminary version in FOCS'11. [doi:10.1137/120865240] 485, 486

[18] KASPER GREEN LARSEN AND HUY LE NGUYEN: Improved range searching lower bounds. In *Proc. 28th Ann. Symp. on Computational Geometry (SCG'12)*, pp. 171–178. ACM Press, 2012. [doi:10.1145/2261250.2261275] 485, 486

[19] PETER BRO MILTERSEN, NOAM NISAN, SHMUEL SAFRA, AND AVI WIGDERSON: On data structures and asymmetric communication complexity. *J. Comput. System Sci.*, 57(1):37–49, 1998. Preliminary version in STOC'95. [doi:10.1006/jcss.1998.1577] 472, 473

[20] RINA PANIGRAHY, KUNAL TALWAR, AND UDI WIEDER: Lower bounds on near neighbor search via metric expansion. In *Proc. 51st FOCS*, pp. 805–814. IEEE Comp. Soc. Press, 2010. [doi:10.1109/FOCS.2010.82] 473

[21] MIHAI PĂTRAŞCU: Towards polynomial lower bounds for dynamic problems. In *Proc. 42nd STOC*, pp. 603–610. ACM Press, 2010. [doi:10.1145/1806689.1806772] 473, 476, 478

[22] MIHAI PĂTRAŞCU AND ERIK D. DEMAINE: Logarithmic lower bounds in the cell-probe model. *SIAM J. Comput.*, 35(4):932–963, 2006. Preliminary versions in STOC'04 and SODA'04. [doi:10.1137/S0097539705447256] 473, 475

[23] MIHAI PĂTRAŞCU AND MIKKEL THORUP: Higher lower bounds for near-neighbor and further rich problems. *SIAM J. Comput.*, 39(2):730–741, 2009. Preliminary version in FOCS'06. [doi:10.1137/070684859] 473

[24] ALEXANDER A. RAZBOROV: On the distributional complexity of disjointness. *Theoret. Comput. Sci.*, 106(2):385–390, 1992. Preliminary version in ICALP'90. [doi:10.1016/0304-3975(92)90260-M] 478, 480

[25] CLAUDE SHANNON: A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, July, October 1948. Reprinted in Mobile Computing and Communications Review, 2001. 479

[26] LESLIE VALIANT: Graph-theoretic methods in low-level complexity. In *Proc. 6th Internat. Symp. on Mathemat. Found. of Comp. Sci. (MFCS'77)*, pp. 162–176, 1977. [doi:10.1007/3-540-08353-7_135] 476

[27] ANDREW CHI-CHIH YAO: Some complexity questions related to distributive computing (preliminary report). In *Proc. 11st STOC*, pp. 209–213. ACM Press, 1979. [doi:10.1145/800135.804414] 478

[28] ANDREW CHI-CHIH YAO: Should tables be sorted? *J. ACM*, 28(3):615–628, 1981. Preliminary version in FOCS'78. [doi:10.1145/322261.322274] 472

AUTHORS

Joshua Brody
Assistant professor
Swarthmore College
Swarthmore, PA
joshua.e.brody@cs.swarthmore.edu
http://www.cs.swarthmore.edu/~brody

Kasper Green Larsen
Assistant professor
Aarhus University
Aarhus, Denmark
larsen@cs.au.dk
http://cs.au.dk/~larsen

ABOUT THE AUTHORS

JOSHUA BRODY received his Ph. D. from Dartmouth College in 2010 under the supervision of Amit Chakrabarti. His research interests include communication complexity, property testing, streaming algorithms, and data structures. He joined the Computer Science Department of Swarthmore College in 2013.

KASPER GREEN LARSEN defended his Ph. D. at Aarhus University in 2013 under the supervision of Lars Arge. His thesis won the 2014 Aarhus University Ph. D. Prize. His main research area is data structures, with an emphasis on range searching and lower bounds. After a year in industry, Larsen has recently returned to academia as an Assistant Professor at Aarhus University.