Lower Bounds for Dynamic Connectivity

Mihai Pătraşcu
MIT CSAIL
mip@mit.edu

Erik D. Demaine
MIT CSAIL
edemaine@mit.edu

ABSTRACT

We prove an $\Omega(\lg n)$ cell-probe lower bound on maintaining connectivity in dynamic graphs, as well as a more general trade-off between updates and queries. Our bound holds even if the graph is formed by disjoint paths, and thus also applies to trees and plane graphs. The bound is known to be tight for these restricted cases, proving optimality of these data structures (e.g., Sleator and Tarjan's dynamic trees). Our trade-off is known to be tight for trees, and the best two data structures for dynamic connectivity in general graphs are points on our trade-off curve. In this sense these two data structures are optimal, and this tightness serves as strong evidence that our lower bounds are the best possible. From a more theoretical perspective, our result is the first logarithmic cell-probe lower bound for any problem in the natural class of dynamic language membership problems, breaking the long standing record of $\Omega(\lg n/\lg\lg n)$. In this sense, our result is the first data-structure lower bound that is "truly" logarithmic, i.e., logarithmic in the problem size counted in bits. Obtaining such a bound is listed as one of three major challenges for future research by Miltersen [13] (the other two challenges remain unsolved). Our techniques form a general framework for proving cell-probe lower bounds on dynamic data structures. We show how our framework also applies to the partial-sums problem to obtain a nearly complete understanding of the problem in cell-probe and algebraic models, solving several previously posed open problems.

Categories and Subject Descriptors

E.1 [**Data**]: Data Structures; F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity

General Terms

Theory, Performance, Algorithms

Keywords

Dynamic connectivity, dynamic graph problems, partial sums problem, cell-probe complexity, lower bounds for data structures

1. INTRODUCTION

This paper builds a new framework for proving amortized lower bounds on online dynamic data structures in the powerful cell-probe model of computation. The *cell-probe model* [19] measures running time as the number of memory words accessed (read or written) by the algorithm. Thus any lower bound in the cell-probe model also holds in the word-RAM model most commonly used by data structures, as well as weaker models such as pointer machines. Our framework extends previous work by the authors [15], enabling us to obtain lower bounds for a wider class of problems in which the answer to each query carries little information (e.g., a single bit). In addition, we show how to obtain trade-off and adaptive lower bounds.

We apply our framework to obtain new, often optimal lower bounds for several problems including dynamic connectivity, dynamic minimum spanning forest, and partial sums. These applications and results are summarized in the next three subsections.

1.1 Dynamic Graph Problems

The *dynamic-connectivity problem* is one of the most important and well-motivated dynamic graph problems, and has been the focus of a large body of research. Formally, the problem 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.

We prove an $\Omega(\lg n)$ amortized lower bound per operation on any data structure for this problem. In addition, we show a more general trade-off between updates and queries. Let t_u be the amortized running time of an update operation, and let t_q be the amortized running time of a query. We prove that $t_q \cdot \lg(t_u/t_q) = \Omega(\lg n)$ and $t_u \cdot \lg(t_q/t_u) = \Omega(\lg n)$. The first bound is relevant when $t_q < t_u$, and the second when $t_u < t_q$. Our bounds apply in the powerful cell-probe model with cells of size $O(\lg n)$, and hold for the average-case of a certain probability distribution. The best previous lower bound was $\Omega(\lg n/\lg\lg n)$, discovered independently by Miltersen et al. [14] and Fredman and Henzinger [6].

Our bound holds even if the graph is always formed by disjoint paths, and thus also applies to forests and plane graphs. For forests, a logarithmic upper bound is given by Sleator and Tarjan's classic data structure for dynamic trees [17]. A trade-off result matching ours for the case $t_q < t_u$ can be obtain using Euler Tour Trees [9]. For plane graphs, a logarithmic upper bound is given by Eppstein et al. [3]. For general graphs, the best known upper bound is only slightly worse: Thorup [18] gives a data structure supporting updates in $O(\lg n(\lg\lg n)^3)$, and queries in $O(\lg n/\lg\lg\lg n)$. Previously, Holm, de Lichtenberg, and Thorup [10] gave a data structure supporting updates in $O(\lg^2 n)$ and queries in $O(\lg n/\lg\lg n)$. These two solutions are not directly comparable, because each sacrifices one running time in order to improve the other. Indeed, both of these data structures are optimal in the sense that they are both on the trade-off curve described above, so it is impossible to obtain a better query time given the update time of each algorithm. This optimality gives strong evidence that our lower bound is the best possible. In addition, we feel that our trade-off result might provide key insight for constructing an optimal data structure.

We can show that our lower bound holds even if Monte Carlo randomization is allowed, and the data structure need only give correct answers with high probability. In addition, we can prove a lower bound of $\Omega(\log_B n)$ in the external memory model with pages of size B. In a sense, this is the first true lower bound of this magnitude for the external memory model, because most previous lower bounds of this type are for the comparison model. These two results will be described in the full version of the paper.

Our result implies lower bounds of the same magnitude for several important dynamic graph problems: dynamic minimum spanning forest (even in plane graphs, and even if all edges have unit costs); dynamic planarity testing; testing whether the entire graph is connected; dynamic bipartiteness testing; dynamic 1-center and dynamic diameter. These reductions are deferred to the final version of this paper. All of these problems have polylogarithmic solutions. In particular, dynamic minimum spanning forests in plane graphs can be maintained in $O(\lg n)$ time [3], matching our lower bound. In [3], a logarithmic lower bound for maintaining the minimum spanning forest was argued by a reduction to sorting. However, such a reduction is not relevant in the powerful RAM and cell-probe models, where we can sort must faster. Our lower bound essentially shows that it is not the costs, but rather the structure of the graph, that is hard to maintain.

1.2 Dynamic Language Membership Problems

Given a language L that is polynomial-time decidable, the *dynamic language membership problem* is 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 $w \in L$. It is not hard to see that the dynamic-connectivity problem can be expressed as a dynamic language membership problem. The idea is to let strings w encode adjacency matrices and define the language L as the set of strings representing undirected graphs for which two special (fixed) nodes are in the same connected component. The size of the problem is now quadratic in the number of nodes, but this affects our logarithmic bound by only a constant factor.

Dynamic language membership problems are advocated by Miltersen [13] as an effective way to gauge the power of lower-bound techniques. For many problems, it is possible to obtain good bounds in terms of a central parameter n, by letting the word size be a large function of n (superexponential universe sizes are not uncommon). This assumption makes lower bounds easier to obtain because updates and queries usually receive a word as a parameter, so the bigger the word size, the more information the data structure has to digest. While lower bounds obtained in this way are interesting in themselves, they usually reflect a better understanding of the particular problem as opposed to a development in lower-bound techniques in general. Dynamic language membership problems are immune to such effects because of the minimal set of operations—bit flips and queries taking no input—and because the size of the problem is counted in bits, so a higher word size makes obtaining a lower bound only harder.

In previous work [15], we obtained an $\Omega(\lg n)$ lower bound for reporting partial sums in a dynamic vector with elements from $\{1,\ldots,n\}$. That result, however, depends on the high entropy of the output of each query ($\Theta(\lg n)$ bits), and thus is not quite satisfactory. The main contribution of this paper is to extend that technique to prove logarithmic lower bounds even if the output to each query is a single bit. In the partial-sums context, this is equivalent to verifying that a given partial sum in correct, rather than computing a partial sum from scratch.

1.3 Partial-Sums Problem

The partial-sums problem is a classic data-structure problem with applications to list indexing and dynamic ranking [7, 2], dynamic arrays [16], histogram maintenance [5], and frequency tables for arithmetic coding [4]. The partial-sums problem has also been very relevant to the lower-bound community because it led the development of several important techniques; see [15] for a comprehensive list of previous results. Formally, the problem asks to maintain an array $A[1..n]$ subject to the following operations:

`update`(k,Δ)**:** modify $A[k] \leftarrow A[k] + \Delta$.
`sum`(k)**:** returns the partial sum $\sum_{i=1}^{k} A[i]$.
`select`(σ)**:** returns an index i satisfying $\mathrm{sum}(i-1) < \sigma \le \mathrm{sum}(i)$. For this operation to be meaningful, the elements of A should come from an ordered set (usually the integers), and we must have all $A[i] > 0$.

Our framework enables us to prove several new lower bounds on the partial-sums problem. These bounds give a nearly complete understanding of the problem in the cell-probe and algebraic models, and solve several previously posed open problems. In particular, we obtain matching trade-off upper and lower bounds for `update` and `sum` in the group model, adaptive entropy bounds in the group model, tight lower bounds for `update` and `select` in the cell-probe model, matching trade-off upper and lower bounds for `update` and `sum` in the cell-probe model, and trade-off lower bounds for `update` and `select` in the cell-probe model. We describe these results in the appendix.

2. GENERAL FRAMEWORK

We now present the framework for our lower bounds in general terms. Consider a sequence of data-structure operations $A_1, A_2, \ldots, A_k$, where each A_i incorporates all information characterizing operation i, i.e., the operation type and any parameters for that type of operation. Upon receiving request A_i, the data structure must produce an appropriate response; for operations other than queries, the response is empty. In this paper, our hard sequences of operations will have a fixed response, and the data structure need only confirm that the answer is correct. Such predictable answers for the hard sequence do not trivialize the problem: the data structure has no guarantee about the sequence of operations, and the information it gathers during a query (by probing certain cells) must provide a certificate that the predicted answer is correct. In other words, the probed cells must uniquely identify the answer to the query, and thus must encode sufficient information to do so. As a consequence, our lower bounds hold even if the algorithm makes nondeterministic cell probes, or if an all-powerful prover reveals a minimal set of cells sufficient to show that a certain answer to a query is correct. One possible framework for nondeterministic computation in the context of online data structures is defined in [12]. Our bounds also hold in this framework.

To establish the lower bounds of this paper, we establish lower bounds for a simpler type of problem. Consider two adjacent intervals of operations: $A_i, \ldots, A_{j-1}$ and $A_j, \ldots, A_k$. At all times, conceptually associate with each memory cell a *chronogram* [7], i.e., the index t of the operation A_t during which the memory cell was last modified. Now consider all read instructions executed by the data structure during operations $A_j, \ldots, A_k$ that access cells with a chronogram in the interval $[i, j-1]$. In other words, we consider the set of cells written during the time interval $[i, j-1]$ and read during the interval $[j, k]$ before they are overwritten. Our intent is to show that the set of locations probed, together with the values read from these locations, must encode a lot of information. Indeed, all the information necessary to answer the queries in the interval $[j, k]$ must come from these cell probes, because an update happening during $[i, j-1]$ cannot be reflected in a cell written before time i. Such bounds will stem from an encoding argument, in conjunction with a simple information-theoretic analysis.

It remains to explain how we can use such lower bounds to show a lower bound for the data-structure problems we are considering. Consider a binary tree whose leaves represent the entire sequence of operations in time order. For every node in the tree, we consider the "information transfer" through that node as follows. Let i and $j-1$ be the indices of the leftmost and rightmost leaves in the subtree of the node's left child, and let j and k be the indices of the leftmost and rightmost leaves in the subtree of the node's right child. Thus, $[i, j-1]$ and $[j, k]$ are two adjacent intervals of operations as described above. We can use the kind of lower bound described above to obtain a lower bound on the number of read instructions executed in the subtree of the right child that read data from the subtree of the left child—we call such read instructions *information transfer* through our node. To get a lower bound for the number of cell probes performed during the entire execution, we simply sum up these lower bounds of information transfer through each node.

To show that this sum of individual lower bounds is indeed an overall lower bound, we must make two important points. First, we claim that we are not double counting any read instructions. Any read instruction is characterized by the time when it occurs and the time when the location was last written. Such a read instruction is counted by only one node, namely, the lowest common ancestor of the read and write times, because the write must happen in the left subtree of the node, and the read must happen in the right subtree. The second point concerns the correctness of summing up individual lower bounds. This approach works for the arguments in this paper, because all lower bounds hold in the average case under the same probability distribution for the operations. Therefore, we can use linearity of expectation to break up the total number of read instructions performed on average into these distinct components. Needless to say, worst-case lower bounds could not be summed in this way.

This line of argument has two important generalizations that we will use. Instead of considering binary trees, we can consider trees of arbitrary degree. Then, we may consider the information transfer either between any node and all its left siblings, or between any node and all its right siblings. It is easy to see that neither of these strategies double counts any read instruction, because a read instruction is counted only for a node immediately below the lowest common ancestor of the read and write times.

3. LOGARITHMIC LOWER BOUND FOR DYNAMIC CONNECTIVITY

We first describe the shape of the graphs we use; refer to Figure 1. The vertices of the graph form an integer grid of size $\sqrt{n}$ by $\sqrt{n}$. Edges only connect vertices from adjacent columns. Each vertex is incident to at most two edges, one edge connecting to a vertex in the previous column and one edge connecting to a vertex in the next column. These edges do not exist only when they cannot because the vertex is in the first or last column. The edges between two adjacent columns of vertices thus form a perfect matching in the complete bipartite graph $K_{\sqrt{n},\sqrt{n}}$, describing a permutation of order $\sqrt{n}$. More precisely, point (x, y_1) is connected to point $(x+1, y_2)$ precisely when $\pi_x(y_1) = y_2$ for a permutation π_x. Another way to look at the graph is in terms of permutation networks. We can imagine that the graph is formed by $\sqrt{n}$ horizontal wires, going between permutation boxes. Inside each permutation box, the order of all wires is changed arbitrarily.

Our graph is always the disjoint union of $\sqrt{n}$ paths. This property immediately implies that the graph is plane, because any embedding maintains planarity (though the edges may have to be routed along paths with several bends).

We define the sequences of operations we consider in terms of *macro-operations*. Macro-operations are of two types, queries and updates, and all receive as parameters a permutation and the index x of a permutation box. To perform an update, all the edges inside the named permutation box are first deleted, and then reconstructed according to the new permutation. Queries on box x test that point $(1, y)$ is connected to point $(x+1, \pi(y))$, for all $y \in \{1, 2, \ldots, \sqrt{n}\}$. The conjunction of these tests is equivalent to testing that the composition of $\pi_1, \pi_2, \ldots, \pi_x$ (the permutations describing the boxes to the left) is identical to the given permutation π. A macro-operation can thus be implemented by $O(\sqrt{n})$ elementary data-structure operations of the same type (update or query).

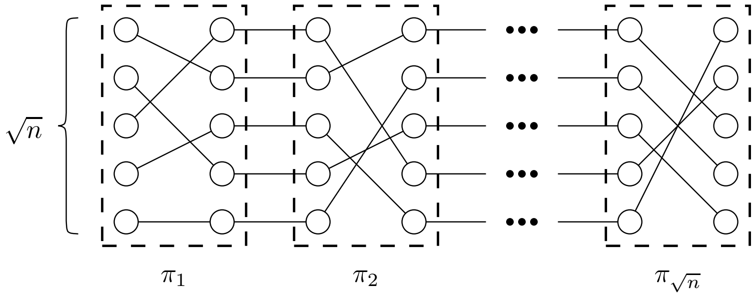

Figure 1: Our graphs can be viewed as a sequence of permutation boxes (dashed). The horizontal edges between boxes are in fact contracted in the actual graphs.

The sequence of macro-operations we consider is generated by a memoryless random process. For each macro-operation, we toss a fair coin to decide whether the operation is a query or an update. Then we choose the index of a permutation box uniformly at random. If the macro-operation is an update, we select a random permutation and execute the update. If the macro-operation is a query, we pass in the permutation formed by the composition of the permutations in all the boxes to the left. This means that the data structure will be asked to prove a tautology, involving the composition of all permutation to the left of some box.

Similar to the framework discussion, we consider a balanced binary tree but with one leaf per macro-operation. For every node that is a right child of its parent, we analyze the read instructions executed in its subtree that access cells with a chronogram in the subtree of its left sibling. Let L be the number of leaves in each of these two subtrees. (For simplicity, we assume that the two subtrees are of equal size, because we can always ignore operations from the larger subtree.) The rest of this section is devoted to proving lower bounds for any pair of adjacent intervals of length L.

The lower bound for two adjacent intervals of operations depends on the interleaving between the indices of the boxes accessed in the two intervals. More precisely, we care about the indices $a_1, a_2, \ldots$ of the boxes updated during the left interval of time, and the indices $b_1, b_2, \ldots$ of the boxes queried during the right interval of time. By suitable relabeling and discarding of duplicates, assume that these indices form sorted sets: $a_1 < a_2 < \cdots$ and $b_1 < b_2 < \cdots$. We define the *interleaving factor* l to be the number of indices i such that, for some index j, $a_i < b_j \leq a_{i+1}$. In words, the interleaving factor measures the number of transitions from runs of a's to runs of b's when merging the two lists of indices.

LEMMA 1. *Consider two adjacent intervals of operations, each of length L. If $L \leq \frac{1}{4}\sqrt{n}$ and the operations are generated by the random process described above, then the interleaving between the two intervals satisfies $E[l] = \Theta(L)$, and, with constant probability, no box is updated twice.*

We omit the proof, because we shall later prove a generalization in Lemma 5 below. We are now ready to state the relation between the interleaving factor and the lower bound we are seeking:

LEMMA 2. *Consider two adjacent intervals of operations, generated by the random process described above, and condition on the same box not being updated twice within these intervals. Let l be the interleaving factor between these intervals. Let w be the number of write instructions executed by the data structure during the first interval, and let r be the number of read instructions executed during the second interval. Finally let c be the number of read instructions executed during the second interval that read cells last written during the first interval. Then $E[c] = \Omega\left(E[l]\sqrt{n} - \frac{E[m]}{\lg n}\right)$, where $m = \lg\binom{r+w}{r}$.*

For the purposes of this section, it will suffice to note that $m \leq r + w$. The tighter bound given by the lemma will become relevant for our trade-off lower bound in Section 4. Before we embark on a proof of this key lemma, we describe its application to our lower bound for dynamic connectivity:

THEOREM 3. *Any data structure for dynamic connectivity must perform $\Omega(\lg n)$ cell probes where each cell has size $O(\lg n)$. This lower bound holds in the average case of a certain probability distribution, even if amortized over a sequence of at least n operations, and even if the graph is always a disjoint family of paths.*

PROOF. Consider a sequence of $k \geq \frac{1}{2}\sqrt{n}$ macro-operations, and let T be the total running time of the data structure. We construct a balanced binary tree over these macro-operations, and analyze the $\frac{1}{2}\lg n - 1$ bottommost levels. Consider a node that is a right child, and let L be the number of leaves in its subtree. Because $L \leq 2^{(1/2)\lg n - 2} = \frac{1}{4}\sqrt{n}$, we can apply Lemma 1, and obtain that the interleaving between the node and its left sibling is, on average, $\Theta(L)$. Then, by Lemma 2, we have $E[c] = \Omega(L\sqrt{n} - \frac{E[m]}{\lg n})$, where c is the number of cell probes associated with this node. However, Lemma 2 applies only if we condition on the same box not being updated twice. Fortunately, by Lemma 1, this event happens with constant probability, so the lower bound can decrease by only a constant factor. As explained above, m is bounded by the number of read instructions in the node's subtree, plus the number of write instructions in the subtree of node's left sibling. Summing for all nodes on a level, m counts all read and write instructions at most once, so we obtain $E[\sum c_i] = \Omega(k\sqrt{n} - \frac{E[T]}{\lg n})$. As explained in the framework discussion, we can sum up the lower bounds for each level to obtain a lower bound on $E[T]$. We obtain that $E[T] = \Omega\left(k\sqrt{n} \cdot \lg n - \lg n \cdot \frac{E[T]}{\lg n}\right)$, which means that $E[T] = \Omega(k\sqrt{n} \cdot \lg n)$. This result implies an average-case amortized lower bound per elementary operation of $\frac{E[T]}{k\sqrt{n}} = \Omega(\lg n)$. $\square$

3.1 An Encoding Argument

This section proves Lemma 2. We consider two adjacent intervals of macro-operations, the first spanning macro-operations $[i, j-1]$ and the second spanning macro-operations $[j, k]$. First we condition on arbitrary choices for the types of the macro-operations in the two intervals. Next we condition on arbitrary assignments of box indices to macro-operations in $[i, k]$, subject to the already imposed condition in the lemma that the box indices of updates are all distinct. After conditioning on these quantities, the interleaving factor l becomes a deterministic quantity. The macro-operations occurring outside the time interval $[i, k]$ will be irrelevant to our analysis, so we can also condition on some arbitrary choices for these operations. Finally, the updates from the interval $[j, k]$ will be irrelevant, so we condition on an arbitrary choice of input permutation for each of these macro-updates. Because the lower bound we obtain does not depend on any of our choices for conditioning in any other way than through l, we can remove the conditioning by the law of iterated expectation: $E[x] = E_Y[E[x \mid y = Y]]$.

Crucial to our analysis is that the permutations passed to the macro-updates in the interval $[i, j-1]$ are chosen independently and uniformly at random. Let U denote the set of indices for the boxes modified by these macro-updates. By the definition of the interleaving factor l between the left and right interval, there must exist l queries $q_1 < q_2 < \cdots < q_l$ in the right interval such that $U \cap [q_{t-1}+1, q_t] \neq \emptyset$ for each $t > 1$.

Now let us look at the state of the data structure immediately before operation j. Define P_t to be the composition of all permutations in the boxes $q_{t-1}+1$ through q_t. Some of these permutations were set before time i or during $[j, k]$, so they are fixed quantities. However, in each P_t, there is at least one random factor from a macro-update that occured during time $[i, j-1]$. An update for a box in the range $q_{t-1}+1$ through q_t must have occured in order to define the interleaving, and this random permutation remains in the expression of P_t because, by assumption, no box was updated twice. In conclusion, each P_t is a composition of some independent and uniformly random permutations (updates from time $[j, k]$) and optionally some fixed permutations, in any order. It follows easily that each P_t is a uniformly random permutation, because conditioning on all but one of the random factors still leaves the result uniformly random. In addition, the P_t's are independent because the random permutations in the compositions are disjoint: the ranges $q_{t-1}+1$ to q_t are disjoint. Therefore, the information-theoretic complexity (the conditional Kolmogorov complexity) of the sequence of l permutations P_t is $\lg\left((\sqrt{n}!)^l\right) - O(1) = l(\sqrt{n} \cdot \lg\sqrt{n} - O(\sqrt{n}))$. In other words, any encoding for the sequence of P_t's must use $\Omega(l\sqrt{n} \cdot \lg n)$ bits on average.

Let w be the number of write instructions executed during the interval $[i, j-1]$. Let r be the number of read instructions executed during the interval $[j, k]$. Finally, let c be the number of read instructions performed by the data structure in the interval $[j, k]$ that access cells last written during the interval $[i, j-1]$. Note that all of w, r, and c are random variables, because the data structure can behave differently depending on the permutations passed to the updates. We give an encoding for the sequence of P_t's that uses $O(\lg n) + c \cdot O(\lg n) + O(m)$ bits, where $m = \lg\binom{r+w}{r}$. Because the expected size of our encoding must be $\Omega(l\sqrt{n} \cdot \lg n)$, we obtain that $E[c] + \frac{E[m]}{O(\lg n)} = \Omega(l\sqrt{n})$ and therefore $E[c] = \Omega\left(l\sqrt{n} - \frac{E[m]}{O(\lg n)}\right)$.

Our encoding consists of three parts. The first part stores the quantities r, w, and c using $O(\lg n)$ bits. The second part encodes the *interesting* cell probes. For each read instruction executed during $[j, k]$ that reads a cell last written during $[i, j-1]$, we encode the address probed and the value read from that cell. This encoding uses $c \cdot O(\lg n)$ bits. For this bound, we assume that addresses fit in a word, so they have $O(\lg n)$ bits; a more careful analysis can observe that the state of the graph depends only on the past $n^{O(1)}$ updates with high probability, and avoid this restriction. Finally, the third part of the encoding is concerned with the uninteresting cell probes. Let A be the set of cells written during $[i, j-1]$ that are never read during $[j, k]$. Let B be the set of cells read during $[j, k]$ that were last written before time i. The information in the third part certifies that A and B are disjoint. To efficiently encode this fact, we need the following result:

LEMMA 4. *For any integers a, b, u satisfying $0 < a \leq b$ and $a + b \leq u$, there exists a system of sets $\mathbb{S}$ with $\lg|\mathbb{S}| = O(a\lg(b/a) + \lg\lg u)$ such that, for all $A, B \subset \{1, 2, \ldots, u\}$ with $|A| = a, |B| = b, A \cap B = \emptyset$, there exists an $S \in \mathbb{S}$ satisfying $A \subset S$ and $B \subset \bar{S}$.*

PROOF. We use the probabilistic method to show that such a set system exists. Select a set S randomly, by letting every element $x \in \{1, 2, \ldots, u\}$ be in the set with probability $p = \frac{a}{a+b}$. Then, for any pair A, B, the probability that $A \subset S$ and $B \subset \bar{S}$ is $p^a(1-p)^b$. The system $\mathbb{S}$ will be formed of sets chosen independently at random, so the probability that there is no good S for some A and B is $\left(1 - p^a(1-p)^b\right)^{|\mathbb{S}|} \leq \exp\left(-p^a(1-p)^b|\mathbb{S}|\right)$. The number of choices for A and B is $\binom{u}{a}\binom{u-a}{b} \leq u^{a+b}$. So the probability that there is no good set in $\mathbb{S}$ for any A, B is at most $u^{a+b}\exp\left(-p^a(1-p)^b|\mathbb{S}|\right) = \exp\left((a+b)\ln u - p^a(1-p)^b|\mathbb{S}|\right)$. As long as this probability is less than 1, such a system $\mathbb{S}$ exists. So we want $(a+b)\ln u < p^a(1-p)^b|\mathbb{S}| = \left(\frac{a}{a+b}\right)^a\left(\frac{b}{a+b}\right)^b|\mathbb{S}|$. We want to choose a system of size greater than $\frac{(a+b)^{a+b+1}\ln u}{a^a b^b}$. Then $\lg|\mathbb{S}| = \Theta((a+b+1)\log_2(a+b) + \lg\lg u - a\log_2 a - b\log_2 b) = \Theta\left(\lg\lg u + a\log_2(b/a) + a \cdot \frac{b}{a}\log_2\left(1 + \frac{a}{b}\right)\right)$. Elementary calculus shows that $\frac{b}{a}\log_2\left(1 + \frac{a}{b}\right) = \Theta(1)$, so our result simplifies to $\lg|\mathbb{S}| = \Theta(\lg\lg u + a\lg(b/a))$. □

We apply this lemma with $a = \min(|A|, |B|)$, $b = \max(|A|, |B|)$, and $u = n^{O(1)}$. The encoding and decoding algorithms can simply iterate through all possible systems $\mathbb{S}$, and choose the first good one. Given this unique choice of a system, the proof that A and B are disjoint is the index of an appropriate set in the system. This index will occupy $O(a\lg(b/a) + \lg\lg u)$ bits, where $\min(r, w) \in [a, a+c]$ and $\max(r, w) \in [b, b+c]$. Elementary asymptotics shows that this quantity is $O(m + \lg\lg n)$.

It remains to show that this information is enough to encode the sequence of P_t's. Remember that all the remaining randomness consists of the random permutations passed to updates in the interval $[i, j-1]$. The decoding algorithm proceeds by testing all possible choices of random permutations for these updates, and simulating the data structure for every case. For each such possible computation history, the decoder tests whether the encoding that would result is identical to the encoding it receives as input. Once it finds such a possible computation history, it can compute the sequence P_t based on this particular instance. Assume for contradiction that two computation histories H and H' generate the same encoding and yet the sequences of P_t's differ. Take the sequence of macro-operations $[i, j-1]$ from H', and append the sequence of macro-operations $[j, k]$ from H. We argue by induction that the data structure has the same behavior during time $[j, k]$ of this hybrid execution as during execution H. For every read instruction executed during time interval $[j, k]$, we can have three cases:

1. The cell probed was written after time j. By the induction hypothesis, the contents of the cell are the same as during execution H.
2. The cell probed was written during time $[i, j-1]$. The cell appears in the encoding, and the encoding is the same for H and H', so nothing changes.
3. The cell probed was written before time i. The third part of the encoding is the same for both H and H'. This means that the set of cells read by H, which were written before time i, is disjoint from the set of cells written by H' during $[i, j-1]$. So the cell probed was not overwritten by the H' segment of this hybrid execution, so the contents are still what was written before time i.

Remember that, during a normal execution, all queries should be answered in the affirmative. Because the data structure behaves in the same way for the hybrid instance of the problem, all queries from time $[j, k]$ continue to get affirmative answers. However, if the sequences of P_t's for H and H' differ, then the compositions of the permutations to the left of some queried box actually change. This means that the data structure behaves erroneously for an instance of the problem, which is our contradiction.

4. TRADE-OFF LOWER BOUNDS FOR DYNAMIC CONNECTIVITY

We now show how our lower-bound framework can be used to derive trade-off lower bounds. In a nutshell, we bias the random process to generate the cheaper operation more frequently, so that the total cost of queries matches the total cost of updates. Then, we analyze the sequence of operations by considering a tree with a higher branching factor. To prove our lower bound, we assume there exists a data structure with amortized running times bounded by t_u for updates and t_q for queries.

We begin by modifying the random process generating the hard instances of the problem. Instead of letting every macro-operation be a query or an update with equal probability, we define the probability of a macro-operation being an update as $p = \frac{t_q}{t_q+t_u}$. Remember that a macro-update is implemented using $2\sqrt{n}$ elementary updates (insertion and deletion of edges) and a macro-query is implemented using $\sqrt{n}$ elementary queries. Then the amortized average-case cost of a macro-operation must be $p(t_u \cdot 2\sqrt{n}) + (1-p)(t_q\sqrt{n}) = \frac{t_u t_q}{t_u+t_q} \cdot 3\sqrt{n}$. We we will prove below that the amortized cost of a macro-operation must also be $\Omega(\sqrt{n}\log_{1/p(1-p)} n)$. This implies that $\frac{t_u t_q}{t_u+t_q} = \Omega(\log_{1/p(1-p)} n) = \Omega\left(\lg n / \lg\left(\frac{(t_u+t_q)^2}{t_u t_q}\right)\right)$. This is equivalent to $\min(t_u, t_q) \lg\left(\frac{\max(t_u,t_q)}{\min(t_u,t_q)}\right) = \Omega(\lg n)$, which is our desired trade-off lower bound.

To show our lower bound, we develop the following stronger version of Lemma 1:

Lemma 5. *Consider two adjacent intervals of operations, randomly generated such that any particular operation is an update with probability $0 < p$. Assume the left interval has length L and the right interval has length R, satisfying $p \cdot L = (1-p)R$ and $L + R \le \frac{1}{2}\sqrt{n}$. Then the interleaving factor between the two intervals satisfies $E[l] = \Theta(p \cdot L)$, and, with constant probability, no box is updated twice.*

Proof. The number of indices touched by the operations is $L+R \le \frac{1}{2}\sqrt{n}$, so the probability that all indices are unique is at least $\frac{1}{2}$. If we show that $E[l \mid \text{unique}] = \Theta(p \cdot L)$, we will have proved our claim, because $l \in [0, \min(L, R)]$ and $\min(L, R) \le 2pL$. Condition on there being $U \le L$ updates in the left interval, and $Q \le R$ queries in the right interval. Further condition on the set of indices being an arbitrary set of size $U+Q$. What remains is to randomly designate Q of these indices to be queries. Then the interleaving l is the number of transitions from updates to queries, as we read the indices in order. The probability that a transition happens on any fixed position is $\frac{U}{U+Q} \cdot \frac{Q}{U+Q-1}$, so by linearity of expectation $E[l \mid Q, U] = (U+Q-1)\frac{U}{U+Q} \cdot \frac{Q}{U+Q-1} = \frac{UQ}{U+Q}$. But Q is a binomial with probability $1-p$, so we have $Q = \Theta((1-p)R) = \Theta(pL)$ with high probability (in $(1-p)R = pL$). This claim remains true even after we condition on all indices being unique, because that event happens with constant probability. Similarly, U is a binomial with probability p, so $U = \Theta(pL)$ with high probability (still in pL). Thus, $E[l] = \Theta(pL)$ with high probability. □

To use this lemma, consider a balanced tree with branching factor $B = \frac{2}{p(1-p)}$ in which the leaves correspond to the macro-operations in order. We will only analyze nodes on the $\frac{1}{2}\log_B n$ bottommost levels, ensuring that there are at most $B^{\frac{1}{2}\log_B n - 1} = \frac{1}{2}\sqrt{n}$ leaves in a node's subtree, as required by Lemma 5.

Let us now analyze the case $t_u \ge t_q$, or $p \le \frac{1}{2}$. In this case, we are interested in the information transfer between a node and its left siblings. The subtree of the node defines the right interval of macro-operations, and the union of the subtrees of all left siblings defines the left interval. We make a claim only regarding nodes that are in the right half of their parent's children. This means that the sizes of the two intervals are related by $L \ge \frac{B}{2}R$, which means $pL \ge (1-p)R \ge R/2$. Thus, by Lemma 5, the expected interleaving is $\Omega(R)$ – clearly, having more indices in the left interval never decreases the interleaving factor. Now we apply Lemma 2, and get that the number of cell probes associated with this node is $\Omega(R\sqrt{n} - \frac{E[m]}{\lg n})$. As before $m = \lg\binom{r+w}{r}$, where r is the number of cells read in the right interval, and w is the number of cells written in the left interval.

We want to sum the lower bounds for all nodes on a certain level. To do that, we must first understand $\sum m_i = \lg \prod \binom{r_i+w_i}{r_i}$ in terms of T, the total running time for the entire sequence of operations. Note that $\sum r_i \le T$, because each read instruction is counted at most once, on a level of the tree. Also note that $\sum w_i \leq T \cdot B$, because each write instruction is counted at most once for every sibling of the node it is under. The quantity $\prod \binom{r_i+w_i}{r_i}$ counts the total numbers of ways to choose r_i elements from a set $r_i + w_i$, where we have a different set for each i. This is bounded from above by the number of ways to choose $\sum r_i$ elements from a single set of $\sum(r_i + w_i)$ objects. Because $T \leq \frac{1}{2}(B+1)T$, we can use the unimodality of the binomial coefficients, and obtain the upper bound $\sum m_i \leq \binom{(B+1)T}{T} = O(T \lg B)$. Because this upper bound holds in any random instance, it also holds in expectation: $E[\sum m_i] = O(E[T] \lg B)$. Now we can sum the lower bounds for all nodes on a level and obtain that the number of cell probes associated with that level is $\Omega(k\sqrt{n} - \frac{E[T]\lg B}{\lg n})$.

For the case $t_u < t_q$ ($p \geq \frac{1}{2}$), we apply a symmetric argument. We analyze the information transfer between any node, giving the left interval, and all its right siblings, giving the right interval. For nodes in the first half of their parent's children, $R \geq \frac{B}{2}L$, which means $(1-p)R \geq pL \geq L/2$. By Lemma 5, the expected interleaving is $\Omega(L)$. By Lemma 2, the number of cell probes associated with this node is $\Omega(L\sqrt{n} - \frac{E[m]}{\lg n})$. The same bound for $\sum m_i$ holds, with the roles of r_i and w_i are switched. Thus, the number of cell probes associated with a level is again $\Omega(k\sqrt{n} - \frac{E[T]\lg B}{\lg n})$. Summing for all levels, we get the lower bound $E[T] = \Omega(k\sqrt{n} \cdot \log_B n - \frac{E[T]\lg B}{\lg n}\log_B n) = \Omega(k\sqrt{n}\log_B n - E[T])$ and, rearranging terms, $E[T] = \Omega(k\sqrt{n} \cdot \log_B n)$. Combined with the analysis from the beginning of this section, this completes the proof of our trade-off lower bound.

THEOREM 6. *Any data structure supporting insertion and deletion of edges in amortized time t_u and supporting dynamic connectivity queries in amortized time t_q must satisfy $t_q \lg(t_u/t_q) = \Omega(\lg n)$ and $t_u \lg(t_q/t_u) = \Omega(\lg n)$.*

Acknowledgments

The authors would like to thank Faith Fich, Mikkel Thorup, and Peter Bro Miltersen for many helpful discussions.

5. REFERENCES

[1] P. Beame and F. E. Fich. Optimal bounds for the predecessor problem and related problems. *Journal of Computer and System Sciences*, 65(1):38–72, 2002.

[2] P. F. Dietz. Optimal algorithms for list indexing and subset rank. In *Proc. 1st Workshop on Algorithms and Data Structures (WADS)*, pages 39–46, 1989.

[3] D. Eppstein, G. F. Italiano, R. Tamassia, R. E. Tarjan, J. R. Westbrook, and M. Yung. Maintenance of a minimum spanning forest in a dynamic planar graph. *Journal of Computer and System Sciences*, 13(1):33–54, 1992.

[4] P. M. Fenwick. A new data structure for cumulative frequency tables. *Software: Practice and Experience*, 24(3):327–336, 1994.

[5] M. L. Fredman. The complexity of maintaining an array and computing its partial sums. *Journal of the ACM*, 29(1):250–260, 1982.

[6] M. L. Fredman and M. R. Henzinger. Lower bounds for fully dynamic connectivity problems in graphs. *Algorithmica*, 22(3):351–362, 1998.

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

[8] H. Hampapuram and M. L. Fredman. Optimal biweighted binary trees and the complexity of maintaining partial sums. *SIAM Journal on Computing*, 28(1):1–9, 1998.

[9] M. R. Henzinger and V. King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *Journal of the ACM*, 46(4):502–516, 1999.

[10] J. Holm, K. de Lichtenberg, and M. 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.

[11] W.-K. Hon, K. Sadakane, and W.-K. Sung. Succinct data structures for searchable partial sums. In *Proc. 14th Annual International Symposium on Algorithms and Computation (ISAAC)*, pages 505–516, 2003.

[12] T. Husfeldt and T. Rauhe. New lower bound techniques for dynamic partial sums and related problems. *SIAM Journal on Computing*, 32(3):736–753, 2003.

[13] P. B. Miltersen. Cell probe complexity - a survey. In *Advances in Data Structures Workshop, Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 1999.

[14] P. B. Miltersen, S. Subramanian, J. S. Vitter, and R. Tamassia. Complexity models for incremental computation. *Theoretical Computer Science*, 130(1):203–236, 1994.

[15] M. Pătraşcu and E. D. Demaine. Tight bounds for the partial-sums problem. In *Proc. 15th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 20–29, 2004.

[16] R. Raman, V. Raman, and S. S. Rao. Succinct dynamic data structures. In *Proc. 7th Workshop on Algorithms and Data Structures (WADS)*, pages 426–437, 2001.

[17] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

[18] M. Thorup. Near-optimal fully-dynamic graph connectivity. In *Proc. 32nd ACM Symposium on Theory of Computing (STOC)*, pages 343–350, 2000.

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

APPENDIX

A. RESULTS FOR THE PARTIAL-SUMS PROBLEM

The partial-sums problem has been considered in several models of computation, and open problems remained in most models. Our techniques yield a complete understanding in most of these cases, with the notable exception of the bit-probe model. We include a discussion of our results below, and defer proofs to the final version of this paper. We trust the interested reader will not have problems filling in these gaps, by combining the proofs from [15] with the new ideas from this paper.

Algebraic models of computation are a natural model for the partial-sums problem. In such models, elements of the array are members of an abstract group or semigroup, and the complexity of an algorithm is measured only in the number of algebraic operation performed. Despite a long history of results, tight lower bounds have been obtained only recently. Hampapuram and Fredman [8] showed an $\Omega(\lg n)$ lower bound of for the semigroup model, and the authors [15] showed a similar bound for the more powerful group model. In [8], the question of trade-offs between `update` and `sum` was also raised. Using our techniques, we can show a trade-off lower bound of the form: $t_q \lg(t_u/t_q) = \Omega(\lg n)$ and $t_u \lg(t_q/t_u) = \Omega(\lg n)$. A matching upper bound for all cases can be obtained by augmenting a B-tree with a variable branching factor.

Hampapuram and Fredman [8] also considered the problem of adaptive bounds, assuming accesses are generated by a random process. For the semigroup model, they characterized the optimal solution in terms of optimal biweighted binary trees, and gave an algorithm for computing the optimal cost if the access probabilities are known. However, this leaves one to desire a characterization of the optimal cost in terms of some concise and meaningful parameter. If accesses to any element are equally likely to be queries or updates, we can show that the entropy of the access sequence is a lower bound in the more powerful group model. Our result is especially interesting given that splay trees achieve an upper bound in terms of the entropy, without actually knowing the access probabilities. This shows an interesting similarity to searching in the comparison model (though the lower-bound techniques differ widely). The proof of our claim is fairly easy, and is based on the following replacement of Lemma 1:

LEMMA 7. *Consider two adjacent intervals of operations, each of length L. Assume the indices accessed by each operation are generated by sampling a probability distribution with entropy H_0. If $L \leq O(\sqrt{H_0})$, then the interleaving factor l between the two intervals satisfies $E[l] = \Theta(L)$.*

Another natural setting for the problem is the cell-probe model of computation. In this setting, the elements of the array are arbitrary b-bit integers, where b is the number of bits in a machine word and we have $b \geq \lg n$ (so that the memory can be addressed by words). Here it is customary to restrict the update values Δ to be δ-bit integers, with $\delta \leq b$; this is motivated by practical applications, where δ is usually much smaller than b. For this model, the authors [15] gave a data structure where each operation takes $O(\lg n/\lg(b/\delta))$, and proved a matching lower bound for sequences of `sum` and `select`. This improved a lower bound by Fredman and Saks [7], an upper bound for `update` and `sum` due to Dietz [2], and an upper bound for `select` due to Raman et al. [16].

A natural question, posed by [15], is whether we can also prove a matching lower bound for sequences of `update` and `select`. We note that even though lower bounds on `sum` proved easier to derive, it is the `select` operation that is crucial for the applications listed above. In addition, this problem is quite interesting from a theoretical perspective, because it is a combination of two major data-structure problems, namely the predecessor and the partial-sums problem. In fact, the problem has already been approached from both directions. Hon et al. [11] noticed that the lower bound of Beame and Fich [1] for the predecessor problem must also hold for `select`, if $\delta = b$. A stronger bound of $\Omega(\lg n/\lg b)$, which applies for any $\delta \geq 1$ has been established by Husfeldt and Rauhe [12], by extending the chronogram technique of Fredman and Saks [7]. Combining the techniques of this paper with the proof from Section 5.2 of [15] (which gives a similar lower bound for `update` and `sum`) yields a tight lower bound of $\Omega(\lg n/\lg(b/\delta))$ for `update` and `select`.

Another natural question is what trade-offs are possible between updates and queries. This question is motivated by the observation that, in many applications, updates are less frequent than queries. This question was asked repeatedly in [8, 16, 11], and partial answers were given in [7, 16, 11]. The techniques of this paper can be used to show a trade-off lower bound of the form $t_q \cdot (\lg(b/\delta) + \lg(t_u/t_q)) = \Omega(\lg n)$, where t_u is the running time of `update` and t_q is the running time of a query (`select` or `sum`). We cannot prove a symmetric tradeoff which is relevant in the case $t_q > t_u$. When $t_q < t_u$, a matching upper bound can be obtained for `update` and `sum`, based on the data structure presented by the authors in [15]. An analogous approach yields a trade-off for `update` and `select`, but it is tight only for restricted ranges of the parameters. The cases where this approach is not tight correspond roughly to the cases where the hardness of answering predecessor queries becomes more relevant than the hardness of maintaining partial sums. We believe that the lower bound needs to be strengthened in these cases; however, this is beyond reach of current techniques.