On Dynamic Bit-Probe Complexity[*]

Mihai Pătraşcu[†] Corina E. Tarniţă[‡]

Abstract

This work present several advances in the understanding of dynamic data structures in the bit-probe model:

- We improve the lower bound record for dynamic language membership problems to $\Omega((\frac{\lg n}{\lg\lg n})^2)$. Surpassing $\Omega(\lg n)$ was listed as the first open problem in a survey by Miltersen.
- We prove a bound of $\Omega(\frac{\lg n}{\lg\lg\lg n})$ for maintaining partial sums in $\mathbb{Z}/2\mathbb{Z}$. Previously, the known bounds were $\Omega(\frac{\lg n}{\lg\lg n})$ and $O(\lg n)$.
- We prove a surprising and tight upper bound of $O(\frac{\lg n}{\lg\lg n})$ for the greater-than problem, and several predecessor-type problems. We use this to obtain the same upper bound for dynamic word and prefix problems in group-free monoids.

We also obtain new lower bounds for the partial-sums problem in the cell-probe and external-memory models. Our lower bounds are based on a surprising improvement of the classic chronogram technique of Fredman and Saks [1989], which makes it possible to prove logarithmic lower bounds by this approach. Before the work of M. Pătraşcu and Demaine [2004], this was the only known technique for dynamic lower bounds, and surpassing $\Omega(\frac{\lg n}{\lg\lg n})$ was a central open problem in cell-probe complexity.

1 Introduction

The bit-probe model is an instantiation of the cell-probe model with one-bit cells. In this model, memory is organized in cells, and algorithms may read or write a cell in constant time. The number of cell probes is taken as the measure of complexity, and the model allows free nonuniform computation. For formal definitions, see [Mil99]. It should be noted that our upper bounds do not use the power of the model in any unnatural way, and in particular do not use nonuniformity.

Bit-probe complexity can be considered a fundamental measure of computation. When analyzing space-bounded algorithms (branching programs), it is usually preferred to cell-probe complexity with higher cell sizes. In data structures, a cell size of $\Theta(\lg n)$ bits is assumed more frequently, but the machine independence and overall cleanness of the bit-probe measure have made it a persistent object of study since the dawn of theoretical computer science. Nonetheless, many of the most fundamental questions are not yet understood. In this paper, we present better upper or lower bounds for several important problems: maintaining partial sums, dynamic connectivity, the greater-than problem, a few variants of predecessor search, and dynamic word problems.

[*]A preliminary version of this paper appeared in the Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05).

[†]MIT, Computer Science and Artificial Intelligence Laboratory.

[‡]Harvard University, Department of Mathematics. Has previously published as Corina E. Pătraşcu.

Our lower bound of $\Omega((\frac{\lg n}{\lg\lg n})^2)$ for dynamic connectivity also has an important complexity-theoretic significance, as it is the highest known bound for an explicit dynamic language membership problem. The previous record was $\Omega(\lg n)$, shown in [MSVT94]. A survey on cell-probe complexity by Miltersen [Mil99] lists improving this bound as the first open problem among three major challenges for future research. It should be noted that our $\widetilde{\Omega}(\lg^2 n)$ bound[1] is far from a mere echo of a $\widetilde{\Omega}(\lg n)$ bound in the cell-probe model. Indeed, $\Omega(\frac{\lg n}{\lg\lg n})$ bounds in the cell-probe model have been known for one and a half decades (including for dynamic connectivity), but the bit-probe record has remained just the slightly higher $\Omega(\lg n)$. To our knowledge, our bound is the first to show a quasi-optimal $\widetilde{\Omega}(\lg n)$ separation between bit-probe complexity and the cell-probe complexity with cells of $\Theta(\lg n)$ bits, when the cell-probe complexity is superconstant.

Interestingly, our ideas also yield important consequences beyond the bit-probe model. On the upper bound side, our result for the greater-than problem has recently served as inspiration for a novel RAM upper bound for dynamic range reporting in one dimension [MPP05]. This work also extends our upper bound to an optimal trade-off between update and query times.

On the lower bound side, we present a subtle improvement to the classic chronogram technique of Fredman and Saks [FS89], which enables it to prove logarithmic lower bounds in the cell-probe model with cells of $\Theta(\lg n)$ bits. To fully appreciate this development, one must remember that the chronogram technique was virtually the only known approach for proving dynamic lower bounds before the work of [PD06]. At the same time, obtaining a logarithmic bound in the cell-probe model was viewed as one of the most important problems in data-structure lower bounds. It is now quite surprising to find that the answer has always been this close.

We also strengthen the chronogram technique by making it possible to derive lower bound trade-offs in the regime of fast updates and slow queries. Though [PD06] could derive some bounds in this regime, their technique was limited and failed to analyze the partial-sums problem for a higher cell size (the natural nonuniform equivalent of the external-memory model). The present paper does imply such a lower bound, almost matching the bounds achieved by buffer trees, which constitute one of the most important tools for external-memory algorithms.

1.1 The Partial-Sums and Related Problems

Consider an arbitrary group G containing at least 2^δ elements. The partial-sums problem asks to maintain an array $A[1..n]$ of elements from G subject to the following operations:

UPDATE(k, Δ)**:** modifies $A[k] \leftarrow \Delta$.

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

Our lower bounds are specializations of the following theorem, which studies the problem in the most general setting. Note in particular that the theorem does not assume $\delta \le b$ (i.e. that every group element fits into a single cell).

Theorem 1. *Consider an implementation of the partial-sums problem in the cell-probe model with b-bit cells. Let t_u denote the expected amortized running time of an update, and t_q the expected running time of a query. Then, in the average case of an input distribution, the following lower*

[1]We use $\widetilde{\Omega}(f)$ to mean a lower bound of $f/\lg^{O(1)} f$.

bounds hold:

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

The following notation is used in this theorem and the remainder of the paper. First, we define $\lg x = \lceil\log_2(x+2)\rceil$, so that $\lg x \geq 1$ even if $x \in [0,1]$. Regarding the asymptotic notation for several parameters, we say $f(x_1,\ldots,x_t) = \Omega(g(x_1,\ldots,x_t))$ if there exists a constant $\gamma > 0$ such that for all by finitely many tuples $(x_1,\ldots,x_t)$, we have $f(x_1,\ldots,x_t) \geq \gamma\cdot g(x_1,\ldots,x_t)$.

The proof of Theorem 1 appears in Section 2. We now proceed to apply this theorem in three interesting interesting setups, and compare with the best previously known results. Of these, the application to dynamic connectivity is the only one which requires a nontrivial set of ideas.

Higher cell size and buffer trees. Assuming $b = \Omega(\lg n)$ and $\delta \leq b$, our bounds simplify to:

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

The first trade-off was recently obtained by [PD06], who also provided a matching upper bound. Note in particular that this implies $\max\{t_u, t_q\} = \Omega(\lg n)$, which had been a major open problem since [FS89].

The second tradeoff for fast updates is new. The techniques of [PD06] did not apply in this range, and they explicitly discussed this as an interesting open problem. Similarly, epoch-based arguments in the style of [FS89] cannot yield any lower bound when $t_q = \Omega(\lg n)$.

Buffer trees [Arg03] are a general algorithmic paradigm for obtaining fast updates, given a higher cell size. For our problem, this yields a *cell-probe* upper bound of $t_u = O(\left\lceil\frac{\delta+\lg n}{b}\cdot\lg_{t_q/\lg n} n\right\rceil)$, for any $t_q = \Omega(\lg n)$. Thus, we obtain tight bounds when $\delta = \Omega(\lg n)$. (Note that in the cell-probe model, we have a trivial lower bound of $t_u \geq 1$, matching the ceiling in the upper bound.)

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

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

The bit-probe complexity for fixed groups. Setting $b = 1$ and $\delta = O(1)$, our lower bounds simplify to:

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

The folklore solution to the problem achieves the following tradeoffs:

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

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

Previously, the best known lower bound was $\max\{t_u, t_q\} = \Omega(\frac{\lg n}{\lg\lg n})$ achieved by Fredman [Fre82]. This was by a reduction to the greater-than problem, which Fredman introduced specifically for this purpose. As we show below, there is an $O(\frac{\lg n}{\lg\lg n})$ upper bound for this problem, so Fredman's technique cannot yield a better result for partial sums.

Dynamic connectivity and a record bit-probe bound. With $b = 1$ and superconstant δ, Theorem 1 easily implies a nominally superlogarithmic bound on $\max\{t_u, t_q\}$. For instance, for partial sums in $\mathbb{Z}/n\mathbb{Z}$ (i.e. $\delta = \lg n$), we obtain $\max\{t_u, t_q\} = \Omega(\frac{\lg^2 n}{\lg\lg n \cdot \lg\lg\lg n})$. This is a modest improvement over the $\Omega(\frac{\lg^2 n}{(\lg\lg n)^2})$ bound of Fredman and Saks [FS89].

However, it is not particularly relevant to judge the magnitude of such bounds, as we are only proving a hardness of $\widetilde{\Omega}(\lg n)$ per bit in the query output and update input, and we can obtain arbitrarily high nominal bounds. As advocated by Miltersen [Mil99], the proper way to gauge the power of lower bound techniques is to consider problems with a minimal set of operations, and, in particular, decision queries. Specifically, for a language L, we look at the dynamic language membership problem, defined as follows. For any fixed n (the problem size), maintain a string $w \in \{0,1\}^n$ under two operations: flip the i-th bit of w, and report whether $w \in L$.

We prove a lower bound of $\Omega((\frac{\lg n}{\lg\lg n})^2)$ for dynamic connectivity. This problem asks to maintain an undirected graph, under insertion and deletion of edges, and queries asking whether two nodes are in the same connected component. The best upper bound is $O(\lg^2 n \cdot (\lg\lg n)^3)$ [Tho00], so our lower bound is optimal up to doubly logarithmic factors. Our lower bound also holds in the important special case when the graph is guaranteed to be a forest.

Dynamic connectivity can be phrased as a dynamic language membership problem [PD06]. The best previous bound for any explicit problem was $\Omega(\lg n)$ due to [MSVT94], so we obtain an almost quadratic improvement. Our trick for handling decision problems is to use the tradeoffs for slow queries and fast updates, since it is not hard to convert a decision query into one returning a large output, at the price of an appropriate slow down. This is the second time, after the analysis of buffer trees, when our extension of the chronogram technique for the regime of slow queries turns out to be very relevant.

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

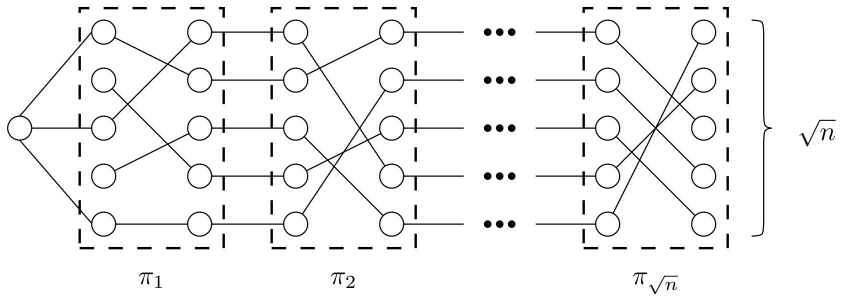

Figure 1: Our graphs can be viewed as a sequence of $\sqrt{n}$ permutation boxes.

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

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

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

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

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

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

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

1.2 The Greater-Than and Related Problems

We begin with a discussion of the greater-than problem. As mentioned already, this was initially considered by Fredman in [Fre82], who used it to deduce a lower bound for the bit-probe complexity of partial sums in $\mathbb{Z}/2\mathbb{Z}$. Consider an infinite memory of bits, initialized to zero. The problem has two stages. In the update stage, the algorithm is given a number $a \in \{1, \ldots, n\}$. After seeing a, the algorithm is allowed to flip T bits in the memory. In the query stage, the algorithm is given $b \in \{1, \ldots, n\}$. Now the algorithm may inspect T bits, and must decide whether or not $a > b$.

Fredman's result stated that $T = \Omega(\frac{\lg n}{\lg \lg n})$. It is quite tempting to believe that one cannot improve past the trivial upper bound $T = O(\lg n)$, since, in some sense, this is the complexity of "writing down" a. If this were true, the problem could be used to obtain a higher lower bound for partial sums. However, we show that Fredman's bound is in fact optimal. As mentioned already, [MPP05] have subsequently extended this result, by considering the possible tradeoffs between the number of bits probes in the update and query stages.

We can obtain the same $O(\frac{\lg n}{\lg \lg n})$ upper bound for the (more natural) dynamic predecessor problem. This asks to maintain a dynamic set $S \subset \{1, \ldots, n\}$ under insertions and deletions, and support queries asking for (some information about) the predecessor in S of a given number. We cannot hope to determine the actual predecessor in $o(\lg n)$ time, because the output itself has this many bits of entropy. However, we can ask for some constant amount of information about the predecessor (a stored "color"), which proves to be enough for many purposes.

Note that lower bounds for the greater-than problem trivially apply to dynamic predecessor, so our result is tight. In the first stage of the greater-than problem, insert the numbers 1, colored red, and a, colored blue. In the second stage, query the color of the predecessor of b, which tells us whether $a > b$.

We finally extend the $O(\frac{\lg n}{\lg \lg n})$ upper bound to two other problems, which are essential for a new upper bound on dynamic word problems that we discuss below. The first problem is a straightforward generalization of the predecessor problem, asking for the colors of the k predecessors of a value, where k is constant. Discussion of the second problem is deferred to Section 3 to avoid a digression into technicalities.

1.3 Dynamic Word Problems

Dynamic prefix problems are defined like the partial sums problem, except that all additions take place in an arbitrary finite monoid. The word problem is identical to the prefix problem, except that queries only ask for the sum of the entire array, not an arbitrary prefix. Such a problem is *defined* by the monoid, so the monoid is considered fixed and constants may depend on it. The aim is to understand the complexity of the problem in terms of the structure of the monoid. This line of research was inspired by the intense study of parallel word problems, which eventually led to a complete classification. Both in the parallel and in the dynamic case, it can be seen that many fundamental problems are equivalent to word and prefix problems for certain classes of monoids. Examples include partial sums modulo a constant, colored predecessor, colored priority queue, and existential range queries in one dimension. In general, we would expect any fundamental problem of a certain one-dimensional flavor to be represented, making word problems an interesting avenue for complexity-theoretic research.

The seminal paper of Frandsen, Miltersen and Skyum [FMS97] achieved tight bounds for many classes of monoids, both in the bit-probe and in the cell-probe models, but the classification is incomplete in both cases. In this paper, we further the classification for the bit-probe model in several directions. Table 1.3 summarizes the old and new bounds. Note that traditionally only the running time of the slowest operation has been considered. We follow this practice, and disregard the tradeoffs between the update and query complexities.

In Section 4.1, we use our solutions for predecessor problems to give an $O(\frac{\lg n}{\lg \lg n})$ upper bound for group-free monoids. This uses the same algebraic toolkit as used by [FMS97] in the cell-probe

	Monoid	New result	Old lower bound	Old upper bound
Prefix problems	group free	$\Theta(\frac{\lg n}{\lg\lg n})$	$\Omega(\frac{\lg n}{\lg\lg n})$	$O(\lg n)$
	contains group	$\Omega(\frac{\lg n}{\lg\lg\lg n})$		
Word problems	commutative group		$\Theta(1)$	
	comm. non-group		$\Theta(\lg\lg n)$	
	contains ENCC	$\Omega(\frac{\lg n}{\lg\lg\lg n})$	$\Omega(\frac{\lg n}{\lg\lg n})$	$O(\lg n)$
	group-free	$\Theta(\frac{\lg n}{\lg\lg n})$		
	other	some are $\Theta(\frac{\lg n}{\lg\lg n})$		

Table 1: Classification of dynamic word and prefix problems in the bit-probe model.

model, but our application needs several interesting algorithmic ideas to handle the idiosyncrasies of the bit-probe model. In particular, while [FMS97] could simply use predecessor queries, we need to invent some queries which gather "enough" information without finding the actual predecessor, thus avoiding the $\Omega(\lg n)$ bottleneck.

On the negative side, our lower bound for partial sums in fixed groups obviously applies to the prefix problem in any monoid containing groups. This creates a separation inside dynamic prefix problems, answering an open problem formulated by [FMS97], who asked whether the bit-probe complexity of prefix queries depends on the monoid at all. Also, we can use [FMS97, Theorem 2.5.1] to imply the same lower bound for the word problem in monoids containing a certain structure, which we call an "externally noncommutative cycle" (ENCC). An ENCC is defined to be a cycle $\{1_a = a^k, a, a^2, \ldots, a^{k-1}\}$ such that there exists b with $1_a ba \neq ab1_a$. This property can be interpreted *loosely* as saying that elements of the cycle don't necessarily commute with elements outside the cycle.

To finish the classification, one would need to strengthen the partial sums lower bound to $\Omega(\lg n)$, which is well motivated independently. From the point of view of algebraic complexity, the only remaining question regards the word problem in monoids containing groups, but no ENCCs. Answering this question seems to require additional insight into the structure of such monoids. The only result we can give is a family of such monoids for which the word problem can be solved in $O(\frac{\lg n}{\lg\lg n})$ time. This is discussed in Section 4.2. On the other hand, we have no example where the partial sums lower bound applies. In fact, we conjecture no such example exists, and the optimal complexity for all monoids in this class is $\Theta(\frac{\lg n}{\lg\lg n})$.

2 Lower Bounds for Partial Sums

We begin by reviewing the chronogram method, at an intuitive level. One first generates a sequence of random updates, ended by one random query. Looking back in time from the query, one partitions the updates into exponentially growing epochs: for a certain r, epoch i contains the r^i updates immediately *before* epoch $i-1$. One then argues that for all i, the query needs to read at least one cell from epoch i with constant probability. This is done as follows. Clearly, information about epoch i cannot be reflected in earlier epochs (those occurred back in time). On the other hand, the latest $i-1$ epochs contain only $O(r^{i-1})$ updates. Assume the cell-probe complexity of each update is bounded by t_u. Then, during the latest $i-1$ epochs, only $O(t^{i-1}t_u b)$ bits are written.

If $r = C \cdot t_u \frac{b}{\delta}$ for a sufficiently large constant C, this number is at most, say, $\frac{1}{10} r^i \delta$. On the other hand, updates in epoch i contain $r^i \delta$ bits of entropy, so all information known outside epoch i can only fix a constant fraction of these updates. If a random query is forced to learn information about a random update from epoch i, it is forced to read a cell from epoch i with constant probability, because the information is not available outside the epoch. This means a query must make $\Omega(1)$ probes in expectation into every epoch, so the lower bound on the query time is given by the number of epochs that one can construct, i.e. $t_q = \Omega(\log_r n) = \Omega(\frac{\lg n}{\lg(t_u b/\delta)})$. A tradeoff of this form was indeed obtained by [AHR98], and is the highest tradeoff obtained by the chronogram method. Unfortunately, even for $\delta = b$, this only implies $\max\{t_u, t_q\} = \Omega(\frac{\lg n}{\lg \lg n})$.

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

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

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

2.1 Formal Framework

We first formalize the overall construction. We consider $2M - 1$ random updates, and insert a random query at a uniformly random position after the M-th update. Now we divide the last M operations before the query into k epochs. Denote the lengths of the epochs by $\ell_1, \dots, \ell_k$, with ℓ_1

being the closest to the query. For convenience, we define $s_i = \sum_{j=1}^{i} \ell_j$.

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

All chronogram lower bounds have relied on an information theoretic argument showing that if epochs 1 up to $i-1$ write too few cells, T_i^{q} must be bounded from below (usually by a constant). As explained above, we instead want to argue that if T_i^{u} is too small, T_i^{q} must be large. Though crucial, this change is very subtle, and the information theoretic analysis follows the same general principles. The following lemma, the proof of which is deferred to Section 2.4, summarizes the results of this analysis:

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

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

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

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

It is easy and instructive to apply Lemma 3 using the ideas of the classic chronogram technique. Define epochs to grow exponentially with rate $r \ge 2$, i.e. $\ell_i = r^i$ and $s_i = O(r^i)$. Assume for simplicity that t_u and t_q are worst-case bounds per operation. Then $T_i^{\mathrm{u}} \le t_u \cdot s_{i-1}$, since the number of probes into epoch i is clearly bounded by the total number of probes made after epoch i. By Lemma 3 we can write $O(\frac{s_{i-1}}{\ell_i} t_u b) + \mathbf{E}[T_i^{\mathrm{q}}]\delta = \Omega(\delta)$, which means $O(\frac{t_u b}{r}) + \mathbf{E}[T_i^{\mathrm{q}}]\delta = \Omega(\delta)$. Setting $r = C t_u \frac{b}{\delta}$ for a sufficiently large constant C, we obtain $\mathbf{E}[T_i^{\mathrm{q}}] = \Omega(1)$. Then $t_q \ge \sum_i \mathbf{E}[T_i^{\mathrm{q}}] = \Omega(\log_r M) = \Omega(\frac{\lg n}{\lg(t_u b/\delta)})$.

As explained before, the key to improving this bound is to obtain a better bound on $\mathbf{E}[T_i^{\mathrm{u}}]$. The next section gives an analysis leading to such a result. Then, Section 2.3 uses this analysis to derive our lower bounds.

2.2 Bounding Probes into an Epoch

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

Lemma 4. *In expectation over a random instance of the problem and a uniformly random $i \in \{1, \dots, k\}$, we have $\mathbf{E}[\frac{T_i^{\mathrm{u}}}{\ell_i}] = O(\frac{\beta}{k} \cdot t_u)$.*

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

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

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

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

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

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

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

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

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

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

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

2.3 Deriving the Tradeoffs of Theorem 1

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

Since the expression of Lemma 3 is increasing in $\mathbf{E}[\frac{T_i^{\mathrm{u}}}{\ell_i}]$ and $\mathbf{E}[T_i^{\mathrm{q}}]$, we can substitute upper bounds for these, obtaining:

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

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

Now let us analyze the tradeoff implied by the first epoch construction. Note that it is valid to substitute the upper bound $\lg k \leq \lg\lg n$ in (1). Also, we use the calculated values for k and $\frac{\beta}{k}$:

$$\frac{t_u}{r \log_r n} \cdot \frac{b + \lg\lg n}{\delta} \; + \; \frac{t_q}{\log_r n} \Big/ \left\lceil \frac{\delta}{b + \lg\lg n} \right\rceil = \Omega(1) \tag{2}$$

We can choose r large enough to make the first term smaller than any constant $\varepsilon > 0$. This is true for r satisfying $\varepsilon \frac{r}{\lg r} > \frac{t_u}{\lg n} \cdot \frac{b+\lg\lg n}{\delta}$, which holds for $\lg r = \Theta(\lg(\frac{t_u}{\lg n} \cdot \frac{b+\lg\lg n}{\delta}))$. For a small enough constant ε, the second term in (2) must be $\Omega(1)$, which implies our tradeoff:

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

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

$$\frac{t_u}{\log_r n} \cdot \frac{b + \lg(r \log_r n)}{\delta} = \Omega(1) \quad \Rightarrow \quad t_u \lg r = \Omega\left(\frac{\delta}{b + \lg(r \log_r n)} \cdot \lg n \right) \tag{3}$$

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

$$\frac{t_q}{r\log_r n} \quad \Big/ \quad \left\lceil \frac{\delta}{b+\lg(r\log_r n)} \right\rceil = \Omega(\varepsilon)$$

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

Plugging into (3), we obtain our final tradeoff:

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

2.4 Proof of Lemma 3

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

$$\frac{\mathbf{E}[T_i^{\mathrm{u}}]}{\ell_i}\left(b + \lg \frac{t_u s_{i-1}}{\mathbf{E}[T_i^{\mathrm{u}}]} \right) + \mathbf{E}[T_i^{\mathrm{q}}] \cdot \min \left\{ \delta, b + \lg \frac{t_q}{\mathbf{E}[T_i^{\mathrm{q}}]} \right\} = \Omega(\delta) \qquad (4)$$

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

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

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

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

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

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

By [PD06, Lemma 5.3], $H(Q^A \mid Q^I, U_i^I, U_{\neg i}^{I\Delta}) = \Omega(\ell_i \delta)$, where H denotes conditional binary entropy. This result is very intuitive. We expect the set of query indices Q^I to interleave with the set of update indices U_i^I in $\Omega(\ell_i)$ places. Each interleaving gives a query that extracts δ bits of information about U_i^Δ (it extract a partial sum linearly independent from the rest). Thus, the set of query answers has $\Omega(\ell_i \delta)$ bits of entropy. The cited lemma assumes our condition $s_i \le \sqrt[3]{n}$, because we do not want updates after epoch i to overwrite updates from epoch i. If there are at most $\sqrt[3]{n}$ updates in epoch i and later, they all touch distinct indices with probability $1 - o(1)$.

We now propose an encoding for Q^A given Q^I and $U_{\neg i}^{I\Delta}$. Comparing the size of this encoding with the previous information lower bound, we will obtain the conclusion of Lemma 3. Consider the following random variables:

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

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

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

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

Lemma 5. *There exists an encoding for Q^A given Q^I and $U_{\neg i}^{I\Delta}$ whose size in bits is:*

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

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

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

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

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

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

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

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

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

To bound $T_{<i}^{\mathrm{u}}$, note that it is the sum of s_{i-1} random variables X_j, each giving the number of probes made by the j-th update before the query. By definition of t_u, the total number of

probes made by all $2M-1$ updates is in expectation at most $(2M-1)t_u$. Our query is inserted randomly in one of M possible positions, so the update described by X_j is chosen randomly among M possibilities. Then, $\mathbf{E}[X_j] \le \frac{(2M-1)t_u}{M} < 2t_u$, and by linearity of expectation $\mathbf{E}[T^{\mathrm{u}}_{<i}] = O(t_u s_{i-1})$. Then, using convexity as before, we can bound:

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

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

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

Here c is a positive constant. This is the desired (4), except that the first term in the min is $\delta + \lg\frac{1}{T^{\mathrm{q}}_i}$ instead of δ. We now show that this makes no difference up to constant factors. First of all, when the second term in the min is smaller, the expressions are obviously identical. Otherwise, pick a constant $c' > 0$ such that $c' \lg\frac{1}{c'} \le \frac{c}{2}$. If $\mathbf{E}[T^{\mathrm{q}}_i] \le c'$, we have $\mathbf{E}[T^{\mathrm{q}}_i \lg\frac{1}{T^{\mathrm{q}}_i}] \le \frac{c}{2}$. Then, moving the offending term to the right hand side, we obtain a lower bound of $c\delta - \frac{c}{2} = \Omega(\delta)$. Finally, assume $\mathbf{E}[T^{\mathrm{q}}_i] > c'$. Then (4) is trivially true if the constant in the Ω notation is at most c', because just the term $\mathbf{E}[T^{\mathrm{q}}_i\delta]$ is larger than the lower bound.

3 The Greater-Than and Related Problems

3.1 The Greater-Than Problem

The solution is remarkably easy. Consider a balanced tree with branching factor $B = \Theta(\frac{\lg n}{\lg\lg n})$, and with n leaves representing the values $\{1, \dots, n\}$. In the update stage, we mark the root-to-leaf path leading to a, taking time $O(\log_B n) = O(\frac{\lg n}{\lg\lg n})$. In the query stage, we first find the lowest common ancestor of a and b. This can be done by scanning b's path upwards, until we find the first marked node, taking time $O(\log_B n)$. Next, we examine the children of the lowest common ancestor. We find the marked child (corresponding to a), and determine whether it is to the left or to the right of the child corresponding to b. Thus, the total time for the query is $O(\log_B n + B) = O(\frac{\lg n}{\lg\lg n})$.

3.2 The Colored k-Predecessors Problem

As before, we use a B-ary tree whose leaves represent the universe. Let $S \subset \{1, \dots, n\}$ be the current set stored by the data structure. A node of the tree is called *active* if any leaf under it is in S. Active nodes are labeled as such. We also attach distinct labels to the minimum and maximum active children of each node. We first show that these labels can be maintained efficiently.

For insertions, traverse the leaf-to-root path looking for an active node. Along the way, label nodes as simultaneously active, minimum and maximum among siblings. When we find the lowest active ancestor, we scan all its children, and relabel the minimum and maximum, as these might have changed. No labels higher in the tree need to be changed. Deletes are symmetric. Climb up the tree from the leaf. As long as the current node is both a minimum and a maximum, we mark its parent inactive. When this is no longer the case, the parent remains active. We end by relabeling the minimum and maximum children at this level. Both operation take time $O(\lg_B n)$ for the climbing phase and $O(B)$ for examining the children on one node, which is $O(\frac{\lg n}{\lg\lg n})$ in total.

We now show how to use the labels to solve the k-predecessors problem. Active nodes will hold the colors of the largest k elements from S under them, or of all elements if there are less than k. Remember that $k = O(1)$ so this takes $O(1)$ bits per node.

Using this information, a query proceeds as follows. Find the lowest active ancestor of the query leaf, by scanning the leaf-to-root path. Then, examine all the left siblings of the child leading to the query. If there are at least k colors stored in these nodes, we return the rightmost k. Otherwise, we continue scanning towards the root. If we ascend from a node marked as the minimum among its siblings, we simply go on. Otherwise, we scan left siblings again, collecting more predecessors. We continue climbing upwards until we find k predecessors, or reach the root. Climbing requires $O(\lg_B n)$ time in total. In addition to this, we make at most $k+1$ scans of the siblings, taking time $O(kB)$. This is so because we scan the siblings the first time we reach an active node, and then only when we have not ascended from a node marked as minimum. Thus, for each scan, we collect at least one additional predecessor. Therefore, the total running time is $O(\lg_B n + kB) = O(\frac{\lg n}{\lg\lg n})$.

We now implement insertions. Up until the lowest active ancestor, we attach a list with one color to every node. Then, we recompute the color list of this ancestor, by examining its children. We now update the color lists on the remaining path to the root according to the following rules:

- if a node is the only active child (it is labeled as both the minimum and maximum), we just copy its color list to the parent, in $O(1)$ time per level.
- if a node has less than k colors in its list, we recalculate the list of the parent by traversing all its children. This takes $O(B)$ time, but we do it at most k times, since the color list of the current node grows each time.
- otherwise, we are only interested in right siblings (because left siblings cannot add to an already full list). If the node is marked as the maximum child, we just copy its list to the parent, in $O(1)$ time.
- otherwise, we recalculate the list of the parent by traversing the right siblings. Each time we do this, the position of the update shifts by at least one to the left, because our node was not maximum among siblings, so we picked up at least one more color. Thus, we can stop updating after we've applied this step k times.

By the analysis done in each case, the running time is again $O(\lg_B n + kB) = O(\frac{\lg n}{\lg\lg n})$. Deletes are almost identical. We begin by updating the labels. Then, we recompute the color list of the lowest active ancestor by scanning its children. Finally, we apply the same algorithm as for an insertion, which updates the lists of all nodes above.

3.3 Finding Segment Representatives

This problem deals with a more unusual stabbing query. We have to maintain a dynamic set $S = \{b_1, b_2, \dots\}$ under the following query operation: given $j \in [b_i, b_{i+1})$, the query determines a value in $[b_i, b_{i+1})$, which is only a function of b_i and b_{i+1}, but not of j or i. Imagine that the elements of S break $\{1, \dots, n\}$ into segments. The query must then produce a representative for the segment stabbed by j, which is inside the segment, but is independent of the actual choice of

j. Adding or removing an element merges or splits segments. The representatives of these affected segments may change arbitrarily, but those of any other segments must remain the same (because they are only functions of the end-points). The segment representative has $\lg n$ bits, so it may be quite surprising that one can be found in $O(\frac{\lg n}{\lg\lg n})$ time. Of course, the query cannot actually write down the representative, but the representative is determined by the query's input and its bit probes.

We use the same B-ary tree and the active labels from before. Updates only need to maintain these labels. It remains to implement queries. The main idea is to find the lowest common ancestor of b_i and b_{i+1}. Once this is known, it is easy to determine a canonical representative. First, take the bits corresponding to the common ancestor, which are common to j, b_i and b_{i+1}. Then, take the next $\lg B$ bits of b_{i+1}, from which we subtract one. Finally, pad with ones. This conceptual computation of the representative takes zero time, since all values involved are already known.

To find the lowest common ancestor of the predecessor and successor, we proceed as follows. Traverse the leaf-to-root path from the query until an active node is found. Scan the children of the node. If there is both an active node to the left and one to the right of the query, we have found the prefixes of b_i and b_{i+1}, which give the answer. Otherwise, assume by symmetry there is an active node to the left, so we have only found the prefix of b_i. We continue searching up the tree, until we find a node which is not marked as maximum among its siblings. At that point, we scan the right siblings, finding the next active one, which corresponds to b_{i+1}.

4 Dynamic Word Problems

4.1 Group-Free Monoids

We now show an $O(\frac{\lg n}{\lg\lg n})$ upper bound for the prefix problem in group-free monoids. We use a corollary of the Krohn-Rhodes decomposition [KR65], which was also used in the cell-probe case [FMS97]. Unfortunately, our application of it is considerably more involved since we can't find exact predecessors.

Theorem 6 ([KR65]). *Let M be a finite nontrivial group-free monoid. One of the following holds:*

1. $M \setminus \{1\} = \langle a \rangle = \{a, a^2, \ldots, a^k = a^{k+1}\}$;
2. *M is left simple, i.e.* $\forall a, b \in M \setminus \{1\}, ab = a$;
3. *$M = V \cup T$, with V, T proper submonoids of M, and $T \setminus \{1\}$ a left ideal of M (i.e. $(\forall)a \in M, b \in T : ab \in T$).*

We prove our upper bound by induction on the size of M. However, we need a stronger induction hypothesis, which assumes a solution for a slightly harder problem. We call this the prefix problem with breakpoints. Consider an array $A[1..n]$, in which an element can either be an element of M, or a breakpoint, denoted ⓑ. The update operation can change any position of A to any element from $M \cup \{$ⓑ$\}$. A query on an arbitrary position i must return the composition of $A[j+1], \ldots, A[i]$, where j is the predecessor of i in the set of breakpoints. For uniformity, we say $A[0] = A[n+1] =$ ⓑ. Also, we assume breakpoints do not appear in consecutive positions. This can easily be arranged by doubling the array, and inserting an identity at every other position.

Assume by induction that the predecessor problem with breakpoints in all group-free monoids of size less than $|M|$ has complexity $O(\frac{\lg n}{\lg\lg n})$. Now apply the decomposition theorem.

Case 1 – $M = \{1_M, a, a^2, \dots, a^k = a^{k+1}\}$. The solution is simple based on the k-predecessor structure. We maintain such a structure on the set $\{i \mid A[i] \neq 1_M\}$ (this includes breakpoints). A query can always be answered based on the values of the k preceding non-identity elements. Specifically, we compose these elements from right to left until either we hit a $\textcircled{b}$, or we reach a^k.

Case 2 – $\forall a, b \in M \setminus \{1_M\}, ab = a$. In this case, a partial sum is determined by the value of the first non-identity element following a breakpoint. Our data structure has the following components:

- a structure for finding segment representatives in the dynamic set $B = \{i \mid A[i] = \textcircled{b}\}$. Denote the elements $B = \{b_1, b_2, \dots\}$.
- a colored predecessor structure, on the set $C = \{i \mid A[i] \neq 1_M\}$ (note that this includes breakpoints). The "color" of i is $A[i]$.
- an array $D[1..n]$; for any segment (b_i, b_{i+1}), the value of $D[\text{repr}(b_i + 1)]$ is equal to the value of the first non-identity element from the segment, or 1_M if none exists. The values of D at positions which are not representatives are ignored.

We first implement a query to position i. First, find the color of the predecessor of i in C. If the predecessor is a breakpoint, return 1_M. Otherwise, the answer is $B[\text{repr}(i)]$.

We implement an update to position i in two steps: first, we change $A[i]$ to 1_M, and then change it to the new value. Updating B and C is trivial. For updating D, we have the following cases:

- replacing $\textcircled{b}$ by 1_M. This merges the two segments around i. The new segment's representative value, $B[\text{repr}(i)]$, will be obtained either from the first segment, i.e. $B[\text{repr}(i-1)]$ before i was removed, or from the second segment ($B[\text{repr}(i+1)]$) if $B[\text{repr}(i-1)] = 1_M$ (so the first segment contained only 1_M's).
- replacing 1_M by $\textcircled{b}$. This splits a segment in two. We obtain the representative of the right segment, $B[\text{repr}(i+1)]$, by querying the successor of i in C. For the left segment, we query the predecessor of i. If it is a breakpoint, $B[\text{repr}(i-1)] = 1_M$; otherwise, it is the old representative of the unsplit segment.
- replacing an element of $M \setminus \{1_M\}$ by 1_M. First, we query the predecessor of i; if it is not a breakpoint, we have nothing to do. Otherwise, we query the successor of i, and update $B[\text{repr}(i)]$ accordingly.
- replacing 1_M by an element of $M \setminus \{1_M\}$. First, we query the predecessor of i in C; if it is not a breakpoint, we have nothing to do. Otherwise, we set $B[\text{repr}(i)]$ to the new value $A[i]$.

Case 3 – $M = T \cup V$. Since $|T|, |V| \leq |M| - 1$, we have, by induction, solutions for the prefix problem with breakpoints, both running in $O(\frac{\lg n}{\lg \lg n})$. To obtain a data structure for M, we use the following components:

- a prefix structure for V, built over the array $A^V[1..n]$. The values of A^V are defined as follows: if $A[i] \in V$, $A^V[i] = A[i]$; otherwise ($A[i] \in T \setminus V$ or $A[i] = \textcircled{b}$), let $A^V[i] = \textcircled{b}$.
- a structure for finding segment representatives in the dynamic set $B = \{i \mid A^V[i] = \textcircled{b}\}$. Denote the elements $B = \{b_1, b_2, \dots\}$.

- a prefix structure for T, built over the array A^T. If $A[i] = ⓑ$, let $A^T[i] = ⓑ$. The other elements are represented in a less straightforward way. For any segment (b_i, b_{i+1}), let $A^T[\text{repr}(b_i + 1)] = \bigoplus_{j=b_i+1}^{b_{i+1}} A[j]$. Note that all but $A[b_{i+1}]$ are elements from V, but the composition is in T because of the ideal property. All elements which are not segment representatives are 1_M.

- a simple array C. For any segment (b_i, b_{i+1}), $C[\text{repr}(b_i + 1)] = A[b_{i+1}]$. Other values of C are ignored.

We claim that we can support the following operation: given any element $j \in (b_i, b_{i+1})$, compute the composition of the entire segment in A^V. Clearly, we can recover the composition of the elements between $b_i + 1$ and $j - 1$, by running a prefix query in A^V. In addition, we can define a monoid W, which is "the reverse" of V: $(\forall)a, b : a \oplus_W b = b \oplus_V a$. Since $|W| = |V| \le m - 1$, we can construct a prefix structure on W by the induction hypothesis. We maintain $A^W[n - i + 1] = A^V[i], (\forall)i$. Now, by running a prefix query in A^W at position $n - j + 1$, we are effectively running a suffix query in A^V. By composing the prefix and the suffix, we obtain the composition of the entire segment.

We can now easily support a prefix query to position i. First, we run a prefix query in A^T up to $\text{repr}(i) - 1$. This will give the desired partial sum up to the predecessor of i in B. To get the last part of the prefix, which consists only of elements from V', we simply ask for the prefix up to i in A^V. This works because the predecessor from B of i is the most recent breakpoint in A^V.

Updates are done in two steps: first we change the old value to 1_M, and then we change this to the new value. We distinguish the following cases:

- both the old and new values are in V (possibly 1_M). We update $A^V[i]$, and compute the composition of the entire segment containing i. Then, we compose this with $C[\text{repr}(i)]$, and we update $A^T[\text{repr}(i)]$ to this new value.

- an element from $T \setminus V$ is replaced by 1_M. We update A^V and remove i from B, which merges two segments. We remove the old segment representatives from A^T (set $A^T[\text{repr}(i - 1)] = A^T[\text{repr}(i + 1)] = 1_M$). Then, we add the representative for the new segment. The value of $B[\text{repr}(i)]$ is given by the old value $B[\text{repr}(i + 1)]$, corresponding to the right segment; $A^T[\text{repr}(i)]$ is obtained by recomputing the composition of i's segment, as above.

- 1_M is replaced by an element in $T \setminus V$. We update $A^V[i]$ to ⓑ, and insert i into B, thus splitting a segment. We first remove the old segment's representative, setting $A^T[\text{repr}(i)] = 1_M$, and then we add the two new representatives. The values in B are obvious: $B[\text{repr}(i - 1)]$ gets the new value of $A[i]$, and $B[\text{repr}(i + 1)]$ gets the old $B[\text{repr}(i)]$ from the unsplit segment. The values in A^T are obtained by recomputing the compositions of the two segments.

- ⓑ is replaced by 1_M. We update $A^T[i]$ to 1_M. Changing the other structures is identical to the case when an element from $T \setminus V$ is removed.

- 1_M is replaced by ⓑ. We propagate the change to $A^T[i]$. Changing the other structures is identical to the case when an element from $T \setminus V$ is added.

4.2 A Family of Monoids without ENCCs

We now construct monoids which contain only "externally-commutative" groups, i.e. they contain groups, but no ENCCs. Let $M = \{1_M, a, a^2, \ldots, a^k = a^{k+1}\}$ be a monoid, and G be a commutative group. Now consider $M \times G$ with the operation $(x, y) \circledast (z, w) = (x \circledast_M z, y \circledast_G w)$. It follows trivially that $M \times G$ is a monoid, and it contains groups, such as $\{1_M\} \times G$.

We now show $M \times G$ does not contain ENCCs. Consider some element (x, y) generating a cycle. Since x to the cycle length must be equal to x, we have $x \in \{1_M, a^k\}$, which means $x^2 = x$. Also, y generates a cycles in G, which contains the identity of G (since G is a group). Then the identity of the cycle generated by (x, y) must be $(x, 1_G)$. For arbitrary (z, w), we have $(x, 1_G) \circledast (z, w) \circledast (x, y) = (xzx, wy) = (x, y) \circledast (z, w) \circledast (x, 1_G)$, so one cannot find an ENCC.

Finally, observe that there is a trivial solution for the word problem in $M \times G$. We can dissociate the components from M and the components from G altogether, and solve the word problems in M and G independently. Since G is a commutative group, the word problem can be solved in constant time. An $O(\frac{\lg n}{\lg \lg n})$ solution for M follows by the previous section.

Acknowledgments. We are grateful to Thore Husfeldt and an anonymous referee for the conference version, for many helpful suggestions in the presentation of the lower bound proof. Some of the results in this paper originate in a project of the first author for a research course at Harvard. She is grateful to Nick Rogers for useful comments made on that occasion.

References

[AHR98] Stephen Alstrup, Thore Husfeldt, and Theis Rauhe. Marked ancestor problems. In *Proc. 39th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 534–543, 1998.

[Arg03] Lars Arge. The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, 2003. See also WADS'95.

[FMS97] Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. Dynamic word problems. *Journal of the ACM*, 44(2):257–271, 1997. See also FOCS'93.

[Fre82] Michael L. Fredman. The complexity of maintaining an array and computing its partial sums. *Journal of the ACM*, 29(1):250–260, 1982.

[FS89] Michael L. Fredman and Michael E. Saks. The cell probe complexity of dynamic data structures. In *Proc. 21st ACM Symposium on Theory of Computing (STOC)*, pages 345–354, 1989.

[KR65] Kenneth Krohn and John Rhodes. Algebraic theory of machines I. Prime decomposition theorem for finite semigroups and machines. *Transactions of the American Mathematical Society*, 116:450–464, 1965.

[Mil99] Peter Bro Miltersen. Cell probe complexity - a survey. In *19th Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 1999. Advances in Data Structures Workshop.

[MPP05] Christian Worm Mortensen, Rasmus Pagh, and Mihai Pătraşcu. On dynamic range reporting in one dimension. In *Proc. 37th ACM Symposium on Theory of Computing (STOC)*, pages 104–111, 2005.

[MSVT94] Peter Bro Miltersen, Sairam Subramanian, Jeffrey S. Vitter, and Roberto Tamassia. Complexity models for incremental computation. *Theoretical Computer Science*, 130(1):203–236, 1994. See also STACS'93.

[PD06] Mihai Pătraşcu and Erik D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4):932–963, 2006. See also SODA'04 and STOC'04.

[Tho00] Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In *Proc. 32nd ACM Symposium on Theory of Computing (STOC)*, pages 343–350, 2000.