

On the Cell Probe Complexity of Dynamic Membership

Ke Yi* Qin Zhang†

Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong, China
{yike, qinzhang}@cse.ust.hk

Abstract

We study the dynamic membership problem, one of the most fundamental data structure problems, in the cell probe model with an arbitrary cell size. We consider a cell probe model equipped with a cache that consists of at least a constant number of cells; reading or writing the cache is free of charge. For nearly all common data structures, it is known that with sufficiently large cells together with the cache, we can significantly lower the amortized update cost to $o(1)$. In this paper, we show that this is not the case for the dynamic membership problem. Specifically, for any deterministic membership data structure under a random input sequence, if the expected average query cost is no more than $1 + \delta$ for some small constant δ , we prove that the expected amortized update cost must be at least $\Omega(1)$, namely, it does not benefit from large block writes (and a cache). The space the structure uses is irrelevant to this lower bound. We also extend this lower bound to randomized membership structures, by using a variant of Yao’s minimax principle. Finally, we show that the structure cannot do better even if it is allowed to answer a query mistakenly with a small constant probability.

1 Introduction

We study one of the most fundamental data structure problems, *dynamic membership*, in the *cell probe model* [22]. In this model, a data structure is a collection of b -bit cells, and the complexity of any operation on the data structure is just the number of cells that are read and/or changed. It is arguably the strongest computation model one can conceive for data structures; in particular it is at least as powerful as the RAM with any operation set. The cell size b is an important model parameter, and various b ’s have been considered, often leading to models with dramatically different characteristics. The case

$b = 1$ (a.k.a. the *bit probe model*) yields a clean combinatorial model, but it is mainly of theoretical interest. The most studied case is $b = \log u$, where u is the universe size, so that every cell stores one element from the universe. In recent years, there have been a lot of interests in studying much larger cell sizes, potentially going all the way to $b = n^\epsilon$ for some small constant ϵ . This is motivated by the fact that in modern memory hierarchies, data is transferred in larger and larger blocks to amortize the high memory transfer costs. In this paper, we will work with any cell size b , though our result is more meaningful for large b ’s.

In the *membership* problem, we want to build a data structure for a set $S \subset [u]$, $|S| = n \leq u/2$, such that we can decide if $x \in S$ efficiently for any $x \in [u]$. In the dynamic version of the problem, we also need to update the data structure under insertions and deletions of elements in S . As we are mostly interested in lower bounds in this paper, we will only consider insertions. A closely related and more general problem is the *dictionary* problem, in which each element $x \in S$ is in addition associated with a piece of data. If the answer to the membership query on x is “yes”, this piece of data should also be returned. Both problems have been extensively studied in the literature, especially the dictionary problem. With $\log u$ -bit cells, comparison based dictionaries have $\Theta(\log n)$ cost per operation; various hashing techniques can achieve expected $O(1)$ cost. There are also data structures that are specifically designed for membership queries, such as *Bloom filters* [4]. For the dynamic versions of these two problems, the two most important measures are the query time t_q and the (amortized) update time t_u . In this paper, we will study the inherent tradeoff between t_q and t_u for the dynamic membership problem. It turns out that the space the structure uses is irrelevant to our tradeoffs.

In all the membership and dictionary data struc-

*Supported in part by Hong Kong DAG grant (DAG07/08).

†Supported by Hong Kong CERG Grant 613507.

tures (except the trivial one using $\Omega(u)$ space), an operation always starts by first probing a cell (or a constant number of cells) at a fixed location, which stores for example the root of a search tree or the description of a hash function, and then adaptively probe other cells. Thus it is convenient, and actually realistic, to exclude the cost of the first fixed probe, by introducing a *cache* that consists of at least a constant number of cells which can be accessed for free. Note that when we consider a general cache size of m bits for $m \geq b$, the cell probe model essentially becomes the *external memory model* [1] where the cache is the “main memory” and a cell is a “block”. In the cell probe literature this assumption is sometimes not made explicitly. However, when b is large, the availability of a cache, even with only a constant number of cells, could make updates much faster. In this case, $\Omega(1)$ is not a lower bound on t_u any more since it is possible to update b bits with one probe. This is evident from the vast literature on external memory data structures [19]. Using various buffering techniques, for most problems the update cost can be reduced to just slightly more than $O(1/b)$, typically $O(\frac{\text{poly log}(n,u)}{b})$ (see e.g. [3, 5, 8]) without affecting t_q very much. Note that this could be much smaller than 1 for typical values of b of interests in the external memory setting. This line of study has also resulted in a lot of practical data structures that support fast updates, which are especially useful for managing archival data where there are much more updates (mostly insertions) than queries, e.g., network traffic logs, database transaction records [10, 14].

However, no effective buffering technique is known for the dictionary problem. It has been conjectured that the update cost must be $\Omega(1)$ if a constant query time is desired, that is, a dictionary does not benefit from large block writes (and a cache), unlike other external memory data structures. This conjecture has been floating around in the external memory community for quite a while, and was recently stated explicitly by Jensen and Pagh [9]. In this paper, we make the first progress towards proving this conjecture by establishing its correctness for an expected average query time $t_q = 1 + \delta$ for some small constant δ . We will formally state our results after setting up the context. Our lower bound holds for the membership problem, hence also for the more general dictionary problem.

Previous results. We will only review the relevant results on dynamic membership and dictionary structures for $b \geq \log u$; the static case and the bit-probe complexity are considered in [15] and the ref-

erences therein. The most widely used dictionary structure is a *hash table*. Knuth [11] showed that using some standard collision resolution strategies such as linear probing or chaining, a hash table achieves $t_q = t_u = 1 + 1/2^{\Omega(b/\log u)}$, which is extremely close to 1 as b gets large. Here t_q is the expected query cost assuming uniformly random inputs (or equivalently using a truly random hash function), and averaged over all queried keys in the universe; t_u is the expected amortized update cost over a sequence of random insertions. If the random input assumption is lifted and t_q is required to be worst-case, *cuckoo hashing* [16] achieves $t_q = 2$, but $t_u = O(1)$ is still expected. It is believed that t_q and t_u cannot both be made worst-case $O(1)$ (with near-linear space), but there has not been a formal proof. Some super-constant lower bounds on the worst-case $\max\{t_q, t_u\}$ are given in [7, 12, 18], but they use a model either more restrictive than or incomparable to the cell probe model.

Intuitively, membership should be easier than the dictionary problem, but we do not have any membership structure that does strictly better. The *Bloom filter* [4] solves the membership problem with only $O(n)$ bits of space, but querying and updating the structure needs more probes, and it also has a probability of false positives.

There are very few lower bounds for the two problems in the cell probe model. Pagh [15] proved that for static membership, the worst-case t_q is at least 2 with linear space (for $b = \log u$). However, when it comes to hashing, people are generally more interested in its expected performance under uniformly random inputs, since with a reasonably good hash function, real-world inputs indeed appear to be uniformly random; some theoretical explanations have been recently put forward for this phenomenon [13]. Under random inputs, Knuth [11] showed that t_q approaches 1 exponentially quickly in b , just using a standard hash table, so there is little left to do in terms of query performance since $t_q \geq 1 - o(1)$ trivially (the $o(1)$ term is due to elements being stored in the cache). However, as argued above, there is no reason why t_u cannot go below 1, especially when b is large, but currently there is no lower bound on t_u yet except the trivial one $\Omega(1/b)$. In [20], a tradeoff between t_q and t_u is given on the dictionary problem, but there t_q is defined as the expected query cost averaged over all the elements currently in the dictionary, while elements not in the dictionary (i.e., unsuccessful queries) are not considered. In this case, it is proved that if $t_q = 1 + O((b/\log u)^{-c})$ for any $c \geq 1$, any dictionary must have an expected amortized up-

date cost $t_u = \Omega(1)$; if $t_q = 1 + O((b/\log u)^{-c})$ for any $c < 1$, then it is possible to achieve $t_u = o(1)$ provided $b = \Omega(\log^{1/c} n \log u)$. The latter exploits the fact that only the average successful query cost needs to be small; unsuccessful ones require longer time to decide. So these results do not apply to the membership problem.

The tradeoff between t_q and t_u has been considered for other dynamic data structure problems in the cell probe model (with a cache), for example the *marked ancestor problem* [2], *partial sums* [17], and *range reporting* [23]. The lower bounds on t_u , with respect to the range of t_q considered for these problems, are all $o(1)$ (for sufficiently large b) but higher than $\Omega(1/b)$, showing that buffering is still effective but only to a certain extent.

Our results. In this paper, we study the tradeoff between t_q and t_u of any dynamic membership data structure in the cell probe model with any cell size b and a cache of size m , where t_q is the expected query cost averaged over all $x \in [u]$ and t_u is the expected amortized update cost, under a uniformly random sequence of insertions. Our main result is that if $t_q \leq 1 + \delta$ for some small constant δ (any $\delta < \frac{1}{2}$ works for our analysis, but we have not attempted to optimize this constant), then $t_u = \Omega(1)$. The lower bound holds as long as $n = \Omega(mb)$ and $u = \Omega(n)$ for sufficiently large hidden constants. Our result rules out the possibility of achieving $t_q = 1 + o(1)$ and $t_u = o(1)$ simultaneously. Compared with the results of [20], it also shows that when both successful and unsuccessful queries are considered, the problem indeed becomes harder. In addition, our lower bound holds irrespective of the size of the data structure.

One of the main difficulties in proving a lower bound for the membership problem is that the query answer is binary. So we cannot use the *indivisibility* assumption as in the lower bounds for dictionaries [7, 18, 20], which says that for a successful query, a cell storing the element (or one of its copies) has to be probed. A membership data structure may indeed “divide” an element into multiple pieces, as in a Bloom filter. To overcome this difficulty, we take a *functional* view of any (deterministic) membership data structure in the cell probe model, which gives us a clean, combinatorial picture of the problem. We define our model in Section 2, followed by the proof of the lower bound for deterministic data structures in Section 3.

In Section 4 we extend our lower bound to randomized data structures. It turns out that we only need a smaller constant δ so as to make the lower

bound hold. To prove our randomized lower bound, we first give a version of *Yao’s minimax principle* [21], which connects the update cost of a randomized data structure with that of a deterministic data structure on a random update sequence following any distribution. Since our deterministic lower bound already assumes a uniformly random update sequence, with this principle it easily results in a randomized lower bound. The result is actually quite intuitive: as the inputs are already random, a data structure should not be able to improve by using further internal randomization.

Finally in Section 5, we extend our lower bound to data structures that may err with a probability ϵ when answering membership queries, thus incorporating any Bloom filter-type structures, but we in addition allow both false positives and false negatives. We show that as long as δ and ϵ are constants small enough, the update time still has to be $\Omega(1)$.

2 The Model

In this section, we define the our model for any dynamic deterministic data structure, which is at least as strong as the cell probe model.

We will treat computation as the evaluation of functions. Let $[u]$ be the universe and $S \subseteq [u]$ be a dynamic set of cardinality at most n that is maintained by the data structure \mathcal{D} . At any time, \mathcal{D} should be able to evaluate a function g_S , following the procedures that we will specify shortly. In the following we will omit the subscript S from g_S when the context is clear. For membership queries, the goal is to evaluate

$$g(x) = \begin{cases} 1, & x \in S; \\ 0, & x \notin S. \end{cases}$$

To evaluate g , \mathcal{D} will employ a few families of functions. The first family, Ψ , consisting of 2^m functions $\psi_1, \dots, \psi_{2^m} : [u] \rightarrow \{0, 1\}$, represents all possible functions computable entirely within the cache. Since there are $\binom{u}{n}$ different g ’s, some g ’s must not be captured by Ψ . To evaluate these g ’s, we need to read one or more memory cells. Let us focus on the case where only one probe is allowed. By reading one cell, we have b “fresh bits”. Together with the m bits in the cache, we can index a larger family of functions. Let \mathcal{F} be a family of $2^m \times 2^b$ functions $f_{M,B} : [u] \rightarrow \{0, 1\}$ for $M = 1, \dots, 2^m, B = 1, \dots, 2^b$. This does not appear to increase the size of the set of computable functions by much, but the key is that the query algorithm is allowed to choose which cell to read after seeing the queried element x . Thus the

b new bits read by this probe could potentially differ for different queries. On the other hand, note that the cache content has to be the same for all queries.

Realizing this, we need to introduce a third family of functions Π , which consists of 2^m functions $\pi_1, \dots, \pi_{2^m} : [u] \rightarrow \{0\} \cup \mathbb{Z}^+$. Each π_M is called a *cell selector*, which will select the cell to probe depending on the queried item and the current cache content M . When $\pi_M(x) = 0$, the cell selector directs the query algorithm to no cell, namely, we will use the cache-resident function ψ_M to evaluate x . Putting everything together, when the cache content is M and the memory cells store the bit strings B_1, B_2, \dots , upon a queried element $x \in [u]$, the data structure \mathcal{D} will evaluate

$$(2.1) \quad \mathcal{D}(x) = \begin{cases} \psi_M(x), & \text{if } \pi_M(x) = 0; \\ f_{M, B_{\pi_M(x)}}(x), & \text{otherwise.} \end{cases}$$

The query cost of evaluating $\mathcal{D}(x)$ is defined to be 0 for any x such that $\pi_M(x) = 0$, and 1 otherwise.

We should stress that under our model, all the function families Ψ, \mathcal{F}, Π have to be predetermined for a deterministic data structure. What changes as elements are inserted into or deleted from S is the cache content M and the cells B_1, B_2, \dots . These form the *state* of \mathcal{D} . This naturally defines the *update cost* of \mathcal{D} : Every time after S changes, M is allowed to switch to an arbitrary state with no cost, while changing any B_i costs 1.

Our model as defined above only allows the query algorithm to visit one memory cell. We can extend the model to visiting multiple memory cells by cascading the basic scheme: A cache-indexable cell selector is first used to select the first cell to read, then a second cell selector, indexed by both the cache and the content of the first cell, is used to select the second cell, and so forth. This complicates the model significantly. Since in this paper, we are interested in a $t_q = 1 + \delta$ query bound for some small constant δ , we can relax the model in an easier way that avoids these complications.

We introduce a special symbol $*$, and redefine \mathcal{F} to be a family of $2^m \times 2^b$ functions $f_{M,B} : [u] \rightarrow \{0, 1, *\}$. The evaluation procedure remains the same as (2.1), but now we only require $\mathcal{D}(x) \approx g(x)$ for all $x \in [u]$, where we define $* \approx 0$ as well as $* \approx 1$. Conceptually, when $\mathcal{D}(x)$ returns $*$, the data structure is declaring that it cannot determine $g(x)$ by just visiting one memory cell and more probes are needed (note that \mathcal{D} is not allowed to return incorrect answers; we will discuss data structures that may err in Section 5). For lower bound purposes, we say the query cost is 2 for any x such that $\mathcal{D}(x) = *$; for any

x where $\mathcal{D}(x) \neq *$, the query cost is defined the same way as before.

To better understand this “functional” model, let us consider how the standard hash table instantiates in this model. It uses $O(n/(b \log u))$ cells, and for each cell, B_i simply stores all the keys hashed into it; if B_i is full, the extra ones are discarded. The cell selector $\pi_M(x)$ is simply the hash function used. The cache-resident function ψ_M is irrelevant. The function $f_{M,B}$ (which actually does not depend on M) is

$$f_{M,B}(x) = \begin{cases} 1, & \text{if } x \in B; \\ 0, & \text{if } x \notin B \text{ and } B \text{ is not full;} \\ *, & \text{if } x \notin B \text{ and } B \text{ is full.} \end{cases}$$

When a key is inserted or deleted, we simply update the corresponding B_i , with cost 1.

3 Deterministic Data Structures

In this section, we first prove a lower bound for deterministic data structures over an insertion sequence where each element is inserted independently, uniformly at random from $[u]$.

Let \mathcal{D} be as defined in Section 2. We assume that the expected average query cost t_q of \mathcal{D} is no more than $1 + \delta$ at any time for some small constant δ to be determined later, and try to bound the expected amortized update cost t_u from below. We set parameters $\rho = \frac{4b}{n\sigma^2}$ and $s = \frac{\sigma^2}{2\rho} = \frac{n\sigma^4}{8b}$ where $\sigma = \min\{\frac{1-2\delta}{11}, \frac{\delta}{2}\}$.

We neglect the insertion cost for the first σn elements. For the rest of the insertions, we divide them into rounds, with each containing s elements, and then try to lower bound the insertion cost of each round.

Consider any particular round R , and let t_R be the ending time of R , i.e., the number of inserted elements by the end of R . Note that according to our construction of rounds, $t_R \geq \sigma n$. Let M be the state of the cache at time t_R . Let $A_i = \{x \mid \pi_M(x) = i\}$ and $\alpha_i = |A_i|/u$ (note that A_i and α_i are both determined by M , but we omit the subscript M for ease of presentation). Let B_i^{pre} and B_i^{post} be the states of cell i at the beginning and the end of R , respectively. Our notations will refer to the time snapshot t_R except B_i^{pre} . The goal will be to show that many cells i have $B_i^{\text{pre}} \neq B_i^{\text{post}}$, thus those cells must be modified in round R .

Preparatory lemmas. To pave the road to the main proof, we will first formalize some intuitive observations. We first eliminate the effects of the

memory-resident function ψ_M . More precisely we show that with high probability, α_0 has to be small, meaning that there cannot be too many elements whose membership queries can be answered directly by ψ_M .

LEMMA 3.1. *At time t_R , $\alpha_0 \leq \sigma$ with probability at least $1 - 1/2^{\Omega(b)}$.*

Proof: Let k be the number of elements that have been inserted by time t_R ($k = t_R \geq \sigma n$). We show that for any M such that $\alpha_0 > \sigma$, with high probability, ψ_M will not evaluate the corresponding A_0 correctly. Consider a particular M , suppose the multiset $\{\psi_M(x) \mid x \in A_0\}$ contains y “0”, z “1” with $y + z = \alpha_0 u$ (note that $\psi_M(x)$ cannot be “*” for any $x \in A_0$). Let K be the set of the k randomly inserted elements. In order to correctly answer membership queries for all $x \in A_0$, the data structure has to guarantee that

$$\psi_M(x) = \begin{cases} 1, & \text{if } x \in A_0 \cap K; \\ 0, & \text{if } x \in A_0 - K. \end{cases}$$

We can assume $z \leq k$, otherwise ψ_M must not be correct. For every $x \in A_0$ such that $\psi_M(x) = 0$, x cannot be inserted in the first k insertions for ψ_M to be correct. Therefore the probability that ψ_M is a valid evaluating function at the snapshot is no more than (for $u = \Omega(n)$)

$$\begin{aligned} \left(1 - \frac{y}{u}\right)^k &\leq \left(1 - \left(\alpha_0 - \frac{k}{u}\right)\right)^k \\ &\leq \left(1 - \frac{\sigma}{2}\right)^{\sigma n} \leq e^{-\frac{1}{2}\sigma^2 n}. \end{aligned}$$

Since there are at most 2^m states of M , the probability that there is one M with $\alpha_0 > \sigma$ that works is at most

$$2^m \cdot e^{-\frac{1}{2}\sigma^2 n} \leq 1/2^{\Omega(b)}$$

for $n = \Omega(mb)$, i.e., with probability at least $1 - 1/2^{\Omega(b)}$, M has to be one such that $\alpha_0 \leq \sigma$. \square

Define $\delta_* = |\{x \mid \mathcal{D}(x) = *\}|/u$, i.e., the number of “*” (as a fraction of the universe) returned by the data structure \mathcal{D} (at time t_R). Recall that ψ_M does not return any “*”, so each “*” must be contributed by some cell. Since we require $t_q \leq 1 + \delta$, there cannot be too many “*”. More formally:

LEMMA 3.2. *With probability at least $1/3 - 1/2^{\Omega(b)}$, $\delta_* \leq 2\delta + \sigma$.*

Proof: Since we require that at any time the expected average query time of \mathcal{D} over a random input is no more than $1 + \delta$, and for any data structure and a random input, the expected query time for any element is at least $(1 - 1/2^{\Omega(b)})(1 - \sigma)$ (queries answered by ψ_M cost 0 and by Lemma 3.1, $\alpha_0 \leq \sigma$ with probability at least $1 - 1/2^{\Omega(b)}$), we have that at time t_R , with probability at least $1/3$, the average query time of \mathcal{D} is no more than $1 + 2\delta$. By Lemma 3.1 and since answering the query for x with $\mathcal{D}(x) = *$ costs 2, it is easy to see that $\delta_* \leq 2\delta + \sigma$ with probability at least $1/3 - 1/2^{\Omega(b)}$. \square

The basic idea of the proof. By Lemma 3.1, we know that for the majority of the elements, querying them needs to probe a cell. In particular, cell i is responsible for the elements of A_i . We will classify all the cells into five *zones* according to their characteristics: the *bad zone* \mathcal{B} , the *easy zone* \mathcal{E} , the *old zone* \mathcal{O} , the *strong zone* \mathcal{S} , and the *weak zone* \mathcal{W} . For any zone \mathcal{X} , define $A_{\mathcal{X}} = \bigcup_{i \in \mathcal{X}} A_i$, and $\alpha_{\mathcal{X}} = \sum_{i \in \mathcal{X}} \alpha_i$. We also define

$$\delta_{\mathcal{X}} = |\{x \in A_{\mathcal{X}} \mid \mathcal{D}(x) = *\}|/u,$$

i.e., the number of “*” (as a fraction of the universe) returned by zone \mathcal{X} for the elements \mathcal{X} is responsible for. Note that $\sum_{\mathcal{X}} \delta_{\mathcal{X}} = \delta_*$.

We first consider the bad zone, and defer the definitions of the other zones to later. The *bad zone* \mathcal{B} contains the set of cells $\{i \mid \alpha_i > \rho\}$, namely, those cells that are each responsible for a lot of elements. The basic idea of our proof is the following: We will first show that cells in the bad zone altogether can only handle a small fraction of the universe. Then the majority of the queries will be allocated to the other zones, in which each cell is only responsible for a small number of elements. Since the s elements inserted in this round are randomly drawn from $[u]$, they will possibly cause changes in many cells of these zones.

The bad zone. We first show that the bad zone can only handle a small fraction of elements.

LEMMA 3.3. *At time t_R , $\alpha_{\mathcal{B}} \leq \delta_{\mathcal{B}} + \sigma$ with probability at least $1 - 1/2^{\Omega(b)}$.*

Proof: Let k be the number of elements that have been inserted by time t_R ($k = t_R \geq \sigma n$). We first consider a particular M at t_R . We show that if $\alpha_{\mathcal{B}} > \delta_{\mathcal{B}} + \sigma$ under this M , then with high probability, \mathcal{D} cannot evaluate $g(x)$ for all $x \in A_{\mathcal{B}}$ correctly.

Suppose the multiset $\{\mathcal{D}(x) \mid x \in A_{\mathcal{B}}\}$ contains w “*”, y “0”, z “1” with $w + y + z = \alpha_{\mathcal{B}}u$. We have $w = \delta_{\mathcal{B}}u$ and $z \leq k$. Consider a random input of k elements. Since there are at most $1/\rho$ cells B_i with $\alpha_i > \rho$, there are at most $2^{1/\rho \cdot b}$ possible states for answering the membership queries of the set $A_{\mathcal{B}}$. Similar to the proof of Lemma 3.1, the probability that all $x \in A_{\mathcal{B}}$ can be answered correctly at $t_{\mathcal{R}}$ is no more than (by the union bound)

$$\begin{aligned} & 2^{1/\rho \cdot b} \left(1 - \left(\alpha_{\mathcal{B}} - \delta_{\mathcal{B}} - \frac{k}{u} \right) \right)^k \\ & \leq 2^{1/\rho \cdot b} \left(1 - \frac{\sigma}{2} \right)^{\sigma n} \\ & \leq 2^{1/\rho \cdot b} \cdot e^{-\frac{1}{2}\sigma^2 n}. \end{aligned}$$

Since there are at most 2^m different cache states, we conclude that with probability at most

$$2^{1/\rho \cdot b} \cdot e^{-\frac{1}{2}\sigma^2 n} \cdot 2^m \leq 1/2^{\Omega(b)},$$

there is one M with $\alpha_{\mathcal{B}} > \delta_{\mathcal{B}} + \sigma$ that works, i.e., with probability at least $1 - 1/2^{\Omega(b)}$, M has to be one such that $\alpha_{\mathcal{B}} \leq \delta_{\mathcal{B}} + \sigma$. \square

The other four zones. We have shown that a large number of queries must be answered by probing the other four zones. Below we will argue that a lot of cells in these zones have to change in order to handle this large number of queries. Let $I_{\mathcal{R}}$ be the set of elements inserted before \mathcal{R} starts. These four zones are defined as follows.

1. The *easy zone* \mathcal{E} contains all cells i that are not in \mathcal{B} and for which

$$\left| \left\{ x \mid x \in A_i, x \notin I_{\mathcal{R}}, f_{M, B_i^{\text{pre}}}(x) = 1 \right\} \right| \geq 1.$$

2. The *old zone* \mathcal{O} contains all cells i that are not in the previous zones and for which

$$\left| \left\{ x \mid x \in A_i, x \in I_{\mathcal{R}}, f_{M, B_i^{\text{pre}}}(x) = 1 \right\} \right| \geq \frac{n}{\sigma \cdot 1/\rho}.$$

3. The *strong zone* \mathcal{S} contains all cells i that are not in the previous zones and for which

$$\left| \left\{ x \mid x \in A_i, f_{M, B_i^{\text{pre}}}(x) = * \right\} \right| \geq (1 - 2\sigma)\alpha_i u.$$

4. The *weak zone* \mathcal{W} contains the rest of the cells.

Note that these zones are defined at the end snapshot $t_{\mathcal{R}}$ by looking back $f_{M, B_i^{\text{pre}}}(x)$, namely, how

the cell would respond under the current cache status M if the cell content B_i stayed the same as the start of the round \mathcal{R} . If any of the s insertions in \mathcal{R} conflicts with $f_{M, B_i^{\text{pre}}}(x)$, then cell i has to change. Obviously, the number of cells changed is a lower bound on the insertion cost. We will show that many cells have to change for any M that satisfies the conditions in Lemma 3.1, 3.2, and 3.3.

Expected total insertion cost of \mathcal{R} . Before going to the main proof, we first introduce a special bin-ball game which will be used later.

In an (s, p, β) *bin-ball game*, we throw s balls into r (for any $r \geq 1/p$) bins independently at random, following an arbitrary distribution, but the probability that any ball goes to any particular bin is no more than p . After a ball falling into a bin, with probability at most β it will disappear. The cost of the game is defined to be the number of nonempty bins at the end of the process. We have the following lemma with respect to such a game.

LEMMA 3.4. *If $sp + \beta < 1$, then for any $\mu > 0$, with probability at least $1 - e^{-\frac{\mu^2(1-sp-\beta)s}{2}}$, the cost of an (s, p, β) bin-ball game will be at least $(1 - \mu)(1 - sp - \beta)s$.*

Proof: Imagine that we throw the s balls one by one. Let X_j be the indicator variable denoting the event that the j -th ball is thrown into an empty bin and the ball does not disappear. The number of nonempty bins in the end is thus $X = \sum_{j=1}^s X_j$. These X_j 's are not independent, but no matter what has happened previously for the first $j - 1$ balls, we always have $\Pr[X_j = 0] \leq sp + \beta$. This is because at any time, at most s bins are nonempty. Let Y_j ($1 \leq j \leq s$) be a set of independent variables such that

$$Y_j = \begin{cases} 0, & \text{with probability } sp + \beta; \\ 1, & \text{otherwise.} \end{cases}$$

Let $Y = \sum_{j=1}^s Y_j$. Each Y_j is stochastically dominated by X_j , so Y is stochastically dominated by X . We have $\mathbf{E}[Y] = (1 - sp - \beta)s$ and we can apply Chernoff inequality on Y :

$$\Pr[Y < (1 - \mu)(1 - sp - \beta)s] < e^{-\frac{\mu^2(1-sp-\beta)s}{2}}.$$

Therefore with probability at least $1 - e^{-\frac{\mu^2(1-sp-\beta)s}{2}}$, we have $X \geq (1 - \mu)(1 - sp - \beta)s$. \square

LEMMA 3.5. *With probability at least $1/5$, at least $\Omega(s)$ cells in the union of \mathcal{E}, \mathcal{S} and \mathcal{W} have to change their contents during the s random insertions in \mathcal{R} .*

Proof: Let \mathcal{M} be the collection of M such that $\alpha_0 \leq \sigma$, $\alpha_{\mathcal{B}} \leq \delta_{\mathcal{B}} + \sigma$ and $\delta_{\mathcal{B}} + \delta_{\mathcal{S}} \leq \delta_* \leq 2\delta + \sigma$. By Lemma 3.1, 3.2, and 3.3, we know that \mathcal{D} has to use some $M \in \mathcal{M}$ at time $t_{\mathcal{R}}$ with probability at least $1/3 - 1/2^{\Omega(b)} - 1/2^{\Omega(b)} \geq 1/4$.

Consider any particular $M \in \mathcal{M}$. We will first show that the old zone \mathcal{O} is small. Remember that each cell in \mathcal{O} has many elements already inserted before \mathcal{R} . We claim that $\alpha_{\mathcal{O}} \leq \sigma$. Indeed, there are at most n elements inserted before \mathcal{R} , meaning that there are at most $n / \left(\frac{n}{1/\rho \cdot \sigma}\right) = 1/\rho \cdot \sigma$ cells in \mathcal{O} , thus covering at most a $1/\rho \cdot \sigma \cdot \rho = \sigma$ fraction of membership queries over the universe (recall that any cell $i \notin \mathcal{B}$ has $\alpha_i \leq \rho$). We next analyze the cost of the s insertions in two cases depending on the size of the easy zone \mathcal{E} .

Case A: $\alpha_{\mathcal{E}} > \sigma$. In this case, we only consider the easy zone. Intuitively, a cell in \mathcal{E} is “predicting” some elements to be inserted. More precisely, each $i \in \mathcal{E}$ contains at least one $x \in A_i$ that has not been inserted at the beginning of the round and $f_{M, B_i^{\text{pre}}}(x) = 1$. The probability that after s random insertions, x still has not been inserted is no less than $1 - s/u$. If this happens, in order to correctly answer all queries, we should set $f_{M, B_i^{\text{post}}}(x) = 0$ or $*$, meaning that with probability at least $1 - s/u$, cell B_i has to change in the round. By the Chernoff inequality, we know that with probability at least

$$1 - e^{-2 \cdot \left(\frac{1}{2}\right)^2 \cdot \frac{\sigma}{\rho}} \geq 1 - e^{-\Omega(s)},$$

the total number of cells that have to change during the round is at least

$$(1 - s/u - 1/2) \cdot \frac{\sigma}{\rho} \geq \Omega(s).$$

Case B: $\alpha_{\mathcal{E}} \leq \sigma$. In this case, we neglect the cost of \mathcal{E} , and only consider the strong zone \mathcal{S} or the weak zone \mathcal{W} . By definition, the cells in \mathcal{S} each have a lot of “*”, while those in \mathcal{W} have fewer. So intuitively a cell in \mathcal{S} could handle many insertions without changing its content. But since overall we do not have too many “*”, the capacity of \mathcal{S} is limited anyway, unless we are willing to change many cells in it. Thus, many elements have to be handled by the weak zone \mathcal{W} . Since a cell in \mathcal{W} has few “*”, a random insertion is very likely to force it to change. We formalize this intuition below by considering two subcases.

B.1 $\alpha_{\mathcal{S}} > \delta_{\mathcal{S}} + 4\sigma$. In this subcase we focus on \mathcal{S} . First note that by our definition, at $t_{\mathcal{R}}$,

only $\delta_{\mathcal{S}}u$ queries in \mathcal{S} could be answered by “*”. Let i_1, i_2, \dots, i_l be those cells in the \mathcal{S} whose contents are preserved during \mathcal{R} , that is, $B_{i_t}^{\text{pre}} = B_{i_t}^{\text{post}}$ ($t = 1, 2, \dots, l$), then we must have

$$\sum_{t=1}^l \alpha_{i_t} \leq (1 + 3\sigma)\delta_{\mathcal{S}},$$

otherwise the number of “*” answers will be more than $(1 - 2\sigma)(1 + 3\sigma)\delta_{\mathcal{S}}u > \delta_{\mathcal{S}}u$. Therefore, at least

$$\frac{\alpha_{\mathcal{S}} - (1 + 3\sigma)\delta_{\mathcal{S}}}{\rho} \geq \frac{\sigma}{\rho} \geq s$$

cells in \mathcal{S} have changed their contents during \mathcal{R} .

B.2 $\alpha_{\mathcal{S}} \leq \delta_{\mathcal{S}} + 4\sigma$. In this subcase, we neglect zone \mathcal{S} and only consider zone \mathcal{W} . Now the fraction of elements that will be directed to \mathcal{W} is at least $1 - (\alpha_0 + \alpha_{\mathcal{B}} + \alpha_{\mathcal{E}} + \alpha_{\mathcal{O}} + \alpha_{\mathcal{S}}) \geq 1 - (2\delta + \sigma) - 8\sigma \geq 2\sigma$ ¹. We have the following observations. First, for the s random insertions, by the Chernoff bound, we know that with probability at least $1 - e^{-\Omega(s)}$, at least $(2\sigma \cdot s)/2 = \sigma s$ elements will be directed to zone \mathcal{W} . Second, for a random element x and a particular cell B_i in \mathcal{W} , conditioned upon that x falls into \mathcal{W} , the probability that x being directed to B_i is at most $\rho/(2\sigma)$. Third, it is easy to see that the number of changed cells in \mathcal{W} at the end of the round is at least the number of cells B_i that contains at least one new inserted element x with $f_{M, B_i^{\text{pre}}}(x) = 0$, for the reason that if $f_{M, B_i^{\text{pre}}}(x) = 0$, then cell B_i must change because $f_{M, B_i^{\text{post}}}(x)$ must not be 0 (it can be either 1 or *).

Now we bound the number of changed cells in \mathcal{W} , thus the insertion cost of the round. Consider any cell i in \mathcal{W} . Since i is in neither the old zone nor the strong zone, it is not hard to see that for a random x , conditioned upon $\pi(x) = i$, with probability at most $\frac{n/(\sigma \cdot 1/\rho)}{u} + (1 - 2\sigma) \leq 1 - \sigma$, we have $f_{M, B_i^{\text{pre}}}(x) = *$ or 1. For a new inserted element, if $f_{M, B_i^{\text{pre}}}(x) = *$ or 1, we say it disappears after insertion. Therefore, the number of changed cells in \mathcal{W} is at least the cost of the $(\sigma s, \frac{\rho}{2\sigma}, 1 - \sigma)$ bin-ball game with probability at least $1 - e^{-\Omega(s)}$. By Lemma 3.4 (setting $\mu = 1/2$), with probability at least

¹If $\sigma = \frac{1-2\delta}{11} < \frac{\delta}{2}$, the inequality is obvious. Otherwise $\frac{\delta}{2} \leq \frac{1-2\delta}{11}$, then $\delta \leq \frac{2}{15}$, and consequently $\alpha_{\mathcal{W}} \geq \frac{2}{15}$, which is still at least $\delta = 2\sigma$.

$1 - e^{-\Omega(s)}$, the cost of the bin-ball game is at least

$$\frac{1}{2} \cdot \left(1 - \sigma s \cdot \frac{\rho}{2\sigma} - (1 - \sigma)\right) \cdot \sigma s \geq \Omega(s).$$

To sum up, the analysis for either case holds with probability $1 - e^{-\Omega(s)}$ for any particular $M \in \mathcal{M}$. Since there are at most 2^m different M in \mathcal{M} , we know that the analysis holds with probability at least $1 - 2^m \cdot e^{-\Omega(s)}$ for all $M \in \mathcal{M}$. Finally, as argued earlier, \mathcal{D} has to use such an $M \in \mathcal{M}$ at t_R with probability at least $1/4$, we conclude that with probability at least $(1 - 2^m \cdot e^{-\Omega(s)} - 1/4) \geq 1/5$, the total insertion cost of the round will be at least $\Omega(s)$. \square

Lemma 3.5 directly implies that the expected cost of the round would be at least $1/5 \cdot \Omega(s) = \Omega(s)$.

Amortized insertion cost. Now we are in the position to bound the amortized insertion cost. We know that in total there are $(1 - \sigma)n/s$ rounds, thus the amortized cost per insertion is at least

$$\Omega(s) \cdot (1 - \sigma)n/s \cdot 1/n \geq \Omega(1).$$

One can verify that the analysis above works for any $\delta < 1/2$, so we have the following.

THEOREM 3.1. *Suppose we insert a sequence of n random elements into any deterministic, initially empty data structure in the cell probe model with cell size b and cache size m . If the expected total cost of these insertions is $n \cdot t_u$, and the data structure is able to answer a membership query with expected average t_q probes at any time, then we have the following tradeoff: If $t_q < 1 + \frac{1}{2}$, then $t_u \geq \Omega(1)$, provided that $n = \Omega(mb)$ and $u = \Omega(n)$.*

4 Randomized Data Structures

In this section, we will first show a transformation similar to Yao's *minimax principle* [21] that connects the lower bound of a randomized data structure to that of a deterministic data structure on random inputs with respect to the update cost. The main difference between our transformation and Yao's minimax principle is that in the data structure setting, the query guarantees do not directly carry over, that is, the randomized data structure has an expected query time t_q does not mean that the corresponding deterministic one has to have the same t_q .

A minimax principle for data structures. Consider a dynamic data structure problem. Let \mathcal{I} be the

set of all possible update sequences, and \mathcal{D} be the set of all deterministic data structures. Let $Q(\mathcal{D}, I, t)$ be the average query time (over all possible queries) of a data structure \mathcal{D} on an update sequence I at time t (assuming one update per time unit). Let $C_{\geq t_0}(\mathcal{D}, I)$ be the total update cost of \mathcal{D} on I after time t_0 . A randomized data structure can be viewed as a probability distribution \mathbf{q} over \mathcal{D} ; we also consider a probability distribution \mathbf{p} on \mathcal{I} . Let $I_{\mathbf{p}}$ denote a random input chosen according to \mathbf{p} and $\mathcal{D}_{\mathbf{q}}$ denote a random data structure chosen according to \mathbf{q} . The *minimax principle for data structures* states:

THEOREM 4.1. *Let α, β be any sufficiently small constants. Let \mathbf{p} be any probability distribution on \mathcal{I} , and let t_0 be any time step. Suppose $\mathbf{E}_{\mathbf{p}}[Q(\mathcal{D}, I_{\mathbf{p}}, t)] \geq l$ for all $\mathcal{D} \in \mathcal{D}$ and all $t \geq t_0$. Consider any probability distribution \mathbf{q} on \mathcal{D} such that for all $t \geq t_0$,*

$$(4.2) \quad \mathbf{E}_{\mathbf{p}}\mathbf{E}_{\mathbf{q}}[Q(\mathcal{D}_{\mathbf{q}}, I_{\mathbf{p}}, t)] \leq l + \mu.$$

Let $\mathcal{D}' \subseteq \mathcal{D}$ be the set of data structures \mathcal{D} on which the following holds for at least a $(1 - \beta)$ -fraction of $t \in [t_0, n]$:

$$\mathbf{E}_{\mathbf{p}}[Q(\mathcal{D}, I_{\mathbf{p}}, t)] \leq l + \mu/(\alpha\beta).$$

Then we have

$$\mathbf{E}_{\mathbf{p}}\mathbf{E}_{\mathbf{q}}[C_{\geq t_0}(\mathcal{D}_{\mathbf{q}}, I_{\mathbf{p}})] \geq (1 - \alpha) \min_{\mathcal{D} \in \mathcal{D}'} \mathbf{E}_{\mathbf{p}}[C_{\geq t_0}(\mathcal{D}, I_{\mathbf{p}})].$$

Proof: From (4.2) we have

$$\begin{aligned} & \frac{1}{n - t_0 + 1} \sum_{t=t_0}^n \mathbf{E}_{\mathbf{p}}\mathbf{E}_{\mathbf{q}}[Q(\mathcal{D}_{\mathbf{q}}, I_{\mathbf{q}}, t)] \\ &= \mathbf{E}_{\mathbf{q}} \left[\frac{1}{n - t_0 + 1} \sum_{t=t_0}^n \mathbf{E}_{\mathbf{p}}[Q(\mathcal{D}_{\mathbf{q}}, I_{\mathbf{p}}, t)] \right] \\ &\leq l + \mu. \end{aligned}$$

Combined with the condition $\mathbf{E}_{\mathbf{p}}[Q(\mathcal{D}_{\mathbf{q}}, I_{\mathbf{p}}, t)] \geq l$, we know that with probability at least $(1 - \alpha)$, $\mathcal{D}_{\mathbf{q}}$ satisfies

$$\frac{1}{n - t_0 + 1} \sum_{t=t_0}^n \mathbf{E}_{\mathbf{p}}[Q(\mathcal{D}_{\mathbf{q}}, I_{\mathbf{p}}, t)] \leq l + \mu/\alpha.$$

For each such $\mathcal{D}_{\mathbf{q}}$, we have $\mathbf{E}_{\mathbf{p}}[Q(\mathcal{D}_{\mathbf{q}}, I_{\mathbf{p}}, t)] \leq l + \mu/(\alpha\beta)$ for at least $(1 - \beta)$ fraction of time steps $t \geq t_0$. Therefore with probability at least $1 - \alpha$, $\mathcal{D}_{\mathbf{q}} \in \mathcal{D}'$. It follows that $(1 - \alpha) \min_{\mathcal{D} \in \mathcal{D}'} \mathbf{E}_{\mathbf{p}}[C_{\geq t_0}(\mathcal{D}, I_{\mathbf{p}})]$ is a lower bound on the expected cost of $\mathcal{D}_{\mathbf{q}}$ over the random input $I \in \mathcal{I}$ chosen according to \mathbf{p} after time t_0 . \square

The principle only looks at the data structure after t_0 . This is because at the very beginning of the update sequence, all information of the updates can be kept in the cache, therefore, l has to be 0, weakening the applicability of the theorem. With this minimax principle, to prove a lower bound on the update cost of a randomized data structure \mathbf{q} on a random input (hence also on the worst-case input), we can fix an arbitrary update sequence distribution \mathbf{p} , and derive a lower bound on the expected update cost of any deterministic data structure that holds a weaker query guarantee for most time steps.

The lower bound. Now we use Theorem 4.1 and the lower bound for deterministic data structures in Section 3 to derive a lower bound for randomized data structures. We only count the update cost of the randomized data structure after time $t_0 = \sigma n$.

The input distribution \mathbf{p} is still uniformly random. Suppose that the randomized data structure $\mathcal{D}_{\mathbf{q}}$ fulfills the constraint that for any time $t \geq \sigma n$ and any random input sequence $I_{\mathbf{p}}$,

$$(4.3) \quad \mathbf{E}_{\mathbf{p}}\mathbf{E}_{\mathbf{q}}[Q(\mathcal{D}_{\mathbf{q}}, I_{\mathbf{p}}, t)] \leq 1 + \delta.$$

In Section 3 we have shown that $\mathbf{E}_{\mathbf{p}}[Q(\mathcal{D}, I_{\mathbf{p}}, t)] \geq (1 - 2^{\Omega(b)})(1 - \sigma) \geq 1 - \delta$ for any $\mathcal{D} \in \mathfrak{D}$ and any time step $t \geq \sigma n$. Setting $\alpha = \beta = 1/2$ in Theorem 4.1, we have

$$(4.4) \quad \mathbf{E}_{\mathbf{p}}\mathbf{E}_{\mathbf{q}}[C_{\geq \sigma n}(\mathcal{D}_{\mathbf{q}}, I_{\mathbf{p}})] \geq \frac{1}{2} \min_{\mathcal{D} \in \mathfrak{D}'} \mathbf{E}_{\mathbf{p}}[C_{\geq \sigma n}(\mathcal{D}, I_{\mathbf{p}})],$$

where $\mathfrak{D}' \subseteq \mathfrak{D}$ is the set of deterministic data structures such that for any $\mathcal{D} \in \mathfrak{D}'$, $\mathbf{E}_{\mathbf{p}}[Q(\mathcal{D}, I_{\mathbf{p}}, t)] \leq 1 + 7\delta$ holds for at least half of $t \in [\sigma n, n]$.

Now we try to use the results in Section 3 to bound the the RHS of (4.4). Theorem 3.1 requires the query guarantee for all t , but it is easy to deal with the additional condition that only for half of time steps, the query constraint is met. We modify the construction of the rounds as follows: Instead of constructing the rounds at fixed time instances, we construct s groups of rounds by shifting the original group of rounds $0, 1, \dots, s-1$ time steps to the right, respectively. By the pigeon hole principle, we know that there are at least one group such that at least half of the end snapshots of its rounds meet the query constraint $t_q \leq 1 + 7\delta$. Then we conclude that $\min_{\mathcal{D} \in \mathfrak{D}'} \mathbf{E}_{\mathbf{p}}[C_{\geq \sigma n}(\mathcal{D}, I_{\mathbf{p}})] = \Omega(n)$ for any constant $\delta < 1/14$.

THEOREM 4.2. *Suppose we insert a sequence of n uniformly random items into any randomized, initially empty data structure in the cell probe model*

with cell size b and cache size m . If the expected total update cost is $n \cdot t_u$, and the data structure is able to answer a membership query with expected average t_q probes at any time, then we have the following trade-offs: If $t_q < 1 + \frac{1}{14}$, then $t_u \geq \Omega(1)$, provided that $n \geq \Omega(mb)$ and $u = \Omega(n)$.

5 Randomized Data Structures with Errors

The model with errors. We can easily extend our model in Section 2 by allowing an error probability ϵ . Formally, we say a randomized data structure $\mathcal{D}_{\mathbf{q}}$ answers queries with error probability ϵ if at any time t , for any $x \in U = [u]$, $\mathcal{D}_{\mathbf{q}}(x) \neq g(x)$ with probability at most ϵ . Note that here we have made an implicit relaxation that if \mathcal{D} returns $*$ for some x , the query will always be answered correctly by the second probe. We show in this section that allowing a small constant probability of error does not strengthen the model.

The lower bound. Set $\sigma = \min\{\frac{1-2\delta}{11}, \frac{\delta}{2}\}$ as before. Below we will prove a lower bound for any randomized data structure with error probability at most $\epsilon = \sigma^2/40$. We as previously divide a sequence of n random insertions into rounds of size s . Consider any particular round R and its end time snapshot t_R , let K ($|K| = k \geq \sigma n$) be the set of elements already inserted by time t_R and T be the set of elements inserted in round R . Let $\mathfrak{D}'' \subseteq \mathfrak{D}$ be the set of data structures \mathcal{D} such that any $\mathcal{D} \in \mathfrak{D}''$ answers membership queries mistakenly for at most

1. 9ϵ fraction of elements in U ;
2. 9ϵ fraction of elements in K ; and
3. 9ϵ fraction of elements in T .

Since we require that the randomized data structure $\mathcal{D}_{\mathbf{q}}$ answer the membership queries for any $x \in U$ correctly with probability at least $1 - \epsilon$ at any time, thus at time t_R for the universe U , with probability at most $1/9$, $\mathcal{D}_{\mathbf{q}}$ errs for more than a 9ϵ fraction of elements in U . The same argument holds for sets K and T . Therefore, with probability at least $1 - 3 \cdot 1/9 = 2/3$, $\mathcal{D}_{\mathbf{q}} \in \mathfrak{D}''$. Let $\mathfrak{D}^* \subseteq \mathfrak{D}''$ be the set of data structures \mathcal{D} on which $\mathbf{E}_{\mathbf{p}}[Q(\mathcal{D}, I_{\mathbf{p}}, t)] \leq 1 + 11\delta$ holds for at least a $1/2$ -fraction of $t \in [\sigma n, n]$. By similar arguments in the proof of Theorem 4.1, we can show that with probability at least $1/3$, $\mathcal{D}_{\mathbf{q}} \in \mathfrak{D}^*$.

Having these at hand, we can focus on those deterministic data structures $\mathcal{D} \in \mathfrak{D}^*$, and try to lower bound their amortized update costs. Similar arguments as in Section 4 will give us the lower bound

for randomized structures. We will prove a similar result as Theorem 3.1 by making some modifications to the proof in Section 3. Below we only discuss places where modifications are needed.

We consider the effects of the error term on the cache and the five zones one by one. Consider the cache and the bad zone \mathcal{B} first. We will show that Lemma 3.1 and Lemma 3.3 still hold. Let $y = |\{x \mid x \in A_0, \psi_M(x) = 0\}|$ and $z = |\{x \mid x \in A_0, \psi_M(x) = 1\}|$. We show again that for any M such that $\alpha_0 > \sigma$, with high probability, ψ_M will not evaluate the corresponding A_0 correctly. We can assume that $z \leq 10\epsilon u$, otherwise the fraction of erroneous elements in U must be more than $10\epsilon - k/u > 9\epsilon$, contradicting our choice of $\mathcal{D} \in \mathfrak{D}^*$. Now for every $x \in A_0$ such that $\psi_M(x) = 0$, the probability that x is inserted in the first k insertions is at least $y/u \geq \alpha_0 - 10\epsilon \geq \sigma/2$. Applying the Chernoff bound, we have that with probability at least $1 - e^{-\Omega(\sigma^2 n)}$, the number of erroneous elements is at least $\sigma k/4$, which is more than $9\epsilon k$, the maximum number of erroneous elements allowed for set K if $\mathcal{D} \in \mathfrak{D}^*$. The lemma follows by applying a union bound for all M . Using essentially the same argument, Lemma 3.3 also holds in the presence of errors.

Second, for the easy zone \mathcal{E} , if $\alpha_{\mathcal{E}} \leq \sigma$, we still neglect it. Otherwise, applying the arguments in Section 3 we have that with high probability, at least $s/3$ cells should be modified in the current round if \mathcal{D} answers all membership queries correctly. Note that any $\mathcal{D} \in \mathfrak{D}^*$ allows at most $9\epsilon s$ erroneous elements in T , we know that the update cost is at least $s/3 - 9\epsilon s \geq \Omega(s)$.

Third, it is not difficult to notice that the presence of errors will not affect the analysis for zones \mathcal{O} and \mathcal{S} . Since by definition, $\alpha_{\mathcal{O}} \leq \epsilon$ and by the same arguments in Section 3, we have that if $\alpha_{\mathcal{S}} > \delta_{\mathcal{S}} + 4\sigma$, the cost of round R will be at least $\Omega(s)$.

Finally, if $\alpha_0 \leq \sigma, \alpha_{\mathcal{B}} \leq \delta_{\mathcal{B}} + \sigma, \alpha_{\mathcal{O}} \leq \sigma, \alpha_{\mathcal{E}} \leq \sigma, \alpha_{\mathcal{S}} \leq \delta_{\mathcal{S}} + 4\sigma$, we consider the weak zone \mathcal{W} (now $\alpha_{\mathcal{W}} \geq 2\sigma$). The same arguments as in Section 3 show that for any M , the update cost is at least

$$\frac{1}{2} \cdot \left(1 - \sigma s \cdot \frac{\rho}{2\sigma} - (1 - \sigma)\right) \cdot \sigma s \geq \frac{\sigma^2}{4} s$$

with high probability if all the membership queries of elements directed to \mathcal{W} should be answered correctly. Since any $\mathcal{D} \in \mathfrak{D}^*$ allows at most $9\epsilon s$ erroneous elements in T , we know that the update cost is at least $\frac{\sigma^2}{4} s - 9\epsilon s \geq \Omega(s)$.

THEOREM 5.1. *Let ϵ be some constant small enough. Suppose we insert a sequence of n random elements*

into any randomized, initially empty data structure in cell probe model with cell size b bits and cache size m bits. Let t_u and t_q be defined as before. If we require that at any time, for any $x \in U$ the data structure has to answer its membership query correctly with probability at least $1 - \epsilon$, then we have the following tradeoffs: if $t_q < 1 + \frac{1}{22}$, then $t_u \geq \Omega(1)$, provided that $n \geq \Omega(mb)$ and $u = \Omega(n)$.

6 Concluding Remarks

We have made the first step towards a long-standing conjecture in external memory, that any membership data structure (hence any dictionary) does not benefit from larger block writes. If one considers the case $m = \Theta(b)$, our result holds for all block sizes up to $b = O(\sqrt{n})$. Even with such large blocks, we show that any membership structure has to perform one block write for every constant number of updates, if an average query performance of $t_q = 1 + \delta$ is to be guaranteed. In this paper our journey stopped at δ being a small constant. Although it seems small, its significance can be appreciated if we compare it with a standard hash table, which has δ exponentially small in b . Nevertheless, there is still a long way to the conjecture that supposedly holds for any constant (even possibly some super-constant) t_q .

We imagine that proving the conjecture in general could be difficult. A weaker version is to prove so for nonadaptive membership structures [6]. A nonadaptive structure first probes the cache and then decides the locations of all the other probes solely by cache and the queried item. Such structures are especially interesting when parallel accesses are possible in systems with multiple disks or multi-cores. Bloom filters [4] and cuckoo hashing [16] are both well-known examples of nonadaptive structures. We believe that our model and techniques could be useful for proving lower bounds for such structures.

References

- [1] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.
- [2] S. Alstrup, T. Husfeldt, and T. Rauhe. Marked ancestor problems. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 534–543, 1998.
- [3] L. Arge. The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, 2003.
- [4] B. H. Bloom. Space/time trade-offs in hash coding

- with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
- [5] G. S. Brodal and R. Fagerberg. Lower bounds for external memory dictionaries. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 546–554, 2003.
- [6] H. Buhrman, P. B. Miltersen, J. Radhakrishnan, and S. Venkatesh. Are bitvectors optimal? In *Proc. ACM Symposium on Theory of Computing*, pages 449–458, 2000.
- [7] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R. E. Tarjan. Dynamic perfect hashing: upper and lower bounds. *SIAM Journal on Computing*, 23:738–761, 1994.
- [8] R. Fadel, K. V. Jakobsen, J. Katajainen, and J. Teuhola. Heaps and heapsort on secondary storage. *Theoretical Computer Science*, 220(2):345–362, 1999.
- [9] M. S. Jensen and R. Pagh. Optimality in external memory hashing. *Algorithmica*, 52(3):403–411, 2008.
- [10] C. Jermaine, A. Datta, and E. Omiecinski. A novel index supporting high volume data warehouse insertion. In *Proc. International Conference on Very Large Databases*, pages 235–246, 1999.
- [11] D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, 1973.
- [12] K. Mehlhorn, S. Naher, and M. Rauch. On the complexity of a game related to the dictionary problem. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 546–548, 1989.
- [13] M. Mitzenmacher and S. Vadhan. Why simple hash functions work: Exploiting the entropy in a data stream. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, 2008.
- [14] P. O’Neil, E. Cheng, D. Gawlick, and E. O’Neil. The log-structured merge-tree (LSM-tree). *Acta Informatica*, 33(4):351–385, 1996.
- [15] R. Pagh. On the cell probe complexity of membership and perfect hashing. In *Proc. ACM Symposium on Theory of Computing*, pages 425–432, 2001.
- [16] R. Pagh and F. F. Rodler. Cuckoo hashing. *Journal of Algorithms*, 51:122–144, 2004.
- [17] M. Patrăşcu and E. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4):932–963, 2006.
- [18] R. Sundar. A lower bound for the dictionary problem under a hashing model. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 612–621, 1991.
- [19] J. S. Vitter. *Algorithms and Data Structures for External Memory*. Now Publishers, 2008.
- [20] Z. Wei, K. Yi, and Q. Zhang. Dynamic external hashing: The limit of buffering. In *Proc. ACM Symposium on Parallelism in Algorithms and Architectures*, 2009.
- [21] A. C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proc. IEEE Symposium on Foundations of Computer Science*, 1977.
- [22] A. C. Yao. Should tables be sorted? *Journal of the ACM*, 28(3):615–628, 1981.
- [23] K. Yi. Dynamic indexability and lower bounds for dynamic one-dimensional range query indexes. In *Proc. ACM Symposium on Principles of Database Systems*, 2009.