Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 3 (2020)

Tight Static Lower Bounds for Non-Adaptive Data Structures

Giuseppe Persiano* Kevin Yeo†

Abstract

In this paper, we study the *static* cell probe complexity of *non-adaptive* data structures that maintain a subset of n points from a universe consisting of $m = n^{1+\Omega(1)}$ points. A data structure is defined to be non-adaptive when the memory locations that are chosen to be accessed during a query depend only on the query inputs and not on the contents of memory. We prove an $\Omega(\log m/\log(sw/n\log m))$ static cell probe complexity lower bound for non-adaptive data structures that solve the fundamental dictionary problem where s denotes the space of the data structure in the number of cells and w is the cell size in bits. Our lower bounds hold for all word sizes including the bit probe model ($w = 1$) and are matched by the upper bounds of Boninger *et al.* [FSTTCS'17].

Our results imply a sharp dichotomy between dictionary data structures with one round of adaptive and at least two rounds of adaptivity. We show that $O(1)$, or $O(\log^{1-\epsilon}(m))$, overhead dictionary constructions are only achievable with at least two rounds of adaptivity. In particular, we show that many $O(1)$ dictionary constructions with two rounds of adaptivity such as cuckoo hashing are optimal in terms of adaptivity. On the other hand, non-adaptive dictionaries must use significantly more overhead.

Finally, our results also imply static lower bounds for the non-adaptive predecessor problem. Our static lower bounds peak higher than the previous, best known lower bounds of $\Omega(\log m/\log w)$ for the dynamic predecessor problem by Boninger *et al.* [FSTTCS'17] and Ramamoorthy and Rao [CCC'18] in the natural setting of linear space $s = \Theta(n)$ where each point can fit in a single cell $w = \Theta(\log m)$. Furthermore, our results are stronger as they apply to the static setting unlike the previous lower bounds that only applied in the dynamic setting.

*giuper@gmail.com. Università di Salerno.
†kwlyeo@google.com. Google LLC.

ISSN 1433-8092

1 Introduction

The goal of a data structure is to maintain a dataset while enabling efficient queries to information about the stored dataset. For any long-term data storage system such as a file system or database, data structures are integral building blocks that enable efficient access to the large amounts of data maintained by the storage system. Data structures also form the basis of several fundamental algorithms that potentially may process large amount of data such as efficient sorting algorithms [FW93] and IP-routing algorithms [DBCP97] both of which heavily rely on the predecessor data structure.

Data structures may be characterized depending on the supported operations. *Static* data structures consider the problem of maintaining a static or fixed dataset while enabling efficient queries to information about the stored dataset. For static data structures, the main tradeoffs consist of the amount of storage used by the data structure and the running time of the query algorithm. On one end, the static data structure can use large space and simply store the answer for every possible query so that queries are extremely efficient. On the opposite end, the static data structure can simply store the input dataset without any processing and queries will compute its answer by processing the entire input dataset. Therefore, the goal of static data structure design is to find the best tradeoff. *Dynamic* data structures consider the problem of maintaining datasets with operations that can both query and update the underlying dataset. In this case, the tradeoffs consist of the efficiency of the query and update operations. In particular, the goal is to minimize the maximum of the query and update running times.

We may also characterize data structures depending on the manner which they perform their operations. *Adaptive* data structures are able to perform their operations by adaptively interacting with the underlying memory system. In particular, adaptive data structures may request some memory, look at its contents and then decide the next part of memory to request. On the other hand, *non-adaptive* data structures must interact with the underlying memory system in a non-interactive manner. Non-adaptive data structures must specify all the memory contents needed to perform an operation before seeing the contents of any memory. After receiving all requested memory, the non-adaptive data structure must perform the operation and return the result without the ability to request any more memory.

At first, it seems clear that designing adaptive data structure is significantly easier than non-adaptive data structures. So, a natural question is: why even consider non-adaptive data structures? In many practical scenarios, non-adaptive data structures perform operations with significantly smaller latency compared to the their adaptive counterparts. This efficiency arises from the fact that non-adaptive data structures can parallelize all their memory requests at the beginning of a query. In contrast, adaptive data structures must perform their memory requests in sequential order as the contents of a previously requested memory cell must be used to compute the next memory request. With data sets continuing to grow in size, larger portions of data must be stored in secondary memory or distributed across several storage devices. The costs of cache misses or read requests across multiple storage devices may become the bottleneck for query latency. The ability to parallelize memory requests when executing algorithms can lead to significantly smaller latency costs. Therefore, it is important to investigate the costs of fundamental data structure problems in the non-adaptive setting.

The dictionary data structure (also known as perfect hashing, sparse tables or set membership in various scenarios) is a foundational problem that asks to maintain a subset S of n items from the universe $[m]$ and answer simple queries of whether an item q appears in S or not. Tarjan and Yao [TY79] presented a static dictionary construction with constant query time using universal

hashing [CW79] when the universe size m was polynomially bounded by n ($m = n^{O(1)}$). Fredman, Komlós and Szemerédi [FKS82] present a static dictionary scheme with constant query overhead for all universe sizes. This construction is famously known as the FKS hashing scheme. A dynamic variant of the dictionary problem support amortized expected constant time insertions and deletions is presented in [DKM$^+$94]. There has been a long line of other work studying this problem. As an example of some of these works, see [AN96, Mil98, ABKU99, Hag99, Pag00, PR04, PPR07, PT11] and references therein. Importantly, we note that all these constant dictionary data structures require at least two rounds of communication with memory for query operations. For example, dictionary schemes based on hashing [FKS82, PR04] use at least two rounds of adaptivity as they must first retrieve the key to their hash functions and then retrieve memory locations based on hash function evaluations. A natural question is: what is the best query running time possible with only one round of communication with memory (non-adaptivity)?

Another classic data structure problem is the predecessor problem which asks to return the largest element in a subset S that is no larger than the queried item. In contrast, the successor problem asks to return the smallest element in the subset S that is no smaller than the queried item. As the two problems are equivalently hard in any conceivable model, we focus only on the predecessor problem. The classic way to solve the static predecessor problem is to store the subset S in sorted order and simply perform a binary search in $O(\log n)$ time. The van Emde Boas tree [vEB77] uses $O(\log \log m)$ time but required $O(m)$ storage. Y-fast tries [Wil83] reduce the storage to $O(n)$. Fusion trees [FW93] were able to solve the predecessor problem in $O(\log_w n)$ time and $O(n)$ storage where w is cell size in bits. Beame and Fich [BF02] improve the query time with respect to the universe size m to $O(\log \log m / \log \log \log m)$ in polynomial space. Using the construction by Beame and Fich and fusion trees, one can achieve query time of $O(\sqrt{\log n / \log \log n})$. By applying generic transformations from static to dynamic [And96, AT00], dynamic predecessor can be solved in linear space with an additional $O(\log \log n)$ query overhead. We note that all these efficient constructions are highly adaptive and use many rounds of communication with memory. Once again, we can ask: what is the best running time in the non-adaptive case?

Another important aspect of data structure studies are lower bounds. Typically, lower bounds are proved in the *cell probe* model by Yao [Yao81]. Memory is considered to consist of cells of w bits. The only cost in the cell probe model is accessing cells of memory while computation can be performed with no cost. By using such a weak cost model, the resulting lower bounds are significantly stronger as they still apply in more realistic models where computation is charged. There is a long line of works that prove cell probe lower bounds for various problems such as [FS89, PD06, PT06, Lar12a, Lar12b, WY16, LWY18].

Focusing on the predecessor problem, several works present lower bounds [Ajt88, Mil94, MNSW98, BF02, Sen03] for adaptive data structures. Beame and Fich present $\Omega(\log \log m / \log \log \log m)$ lower bounds which are tight with static constructions using polynomial space. Pătraşcu and Thorrup [PT06] present an $\Omega(\log \log m)$ lower bound for linear space static data structures. Recently, Boninger *et al.* [BBK17] and Ramamoorthy and Rao [RR18] present non-adaptive lower bounds of $\Omega(\log m / \log w)$ and $\Omega(\log m / (\log \log m + w))$ that only apply to the dynamic predecessor problem. Boninger *et al.* [BBK17] present a non-adaptive predecessor data structure with $O(\log m / \log(w / \log m))$ overhead in the dynamic setting. Note there is a gap between the upper and lower bounds for the non-adaptive predecessor problem. In this paper, we will resolve this gap along with all the questions previously presented above.

1.1 Our Results

Our main results are static cell probe lower bounds for non-adaptive data structures. In particular, our lower bounds peak at $\Omega(\log m)$ in the natural setting of linear space and where a point from the universe $[m]$ can fit in a single cell which is higher than all previous non-adaptive data structure lower bounds.

Theorem 1 (Informal)**.** *Let* **DS** *be any static data structure that maintains n points from the universe $[m]$ where $m = n^{1+\Omega(1)}$ with non-adaptive dictionary queries. If $w \geq 1$ denotes the cell size of* **DS** *and s denotes the storage size of* **DS** *in number of cells, then* **DS** *must have expected running time $\Omega(\log m / \log(sw/n \log m))$.*

As one can solve the dictionary problem using exactly one predecessor query, our results also imply lower bounds for non-adaptive predecessor data structures. Recent works on non-adaptive lower bounds have focused on the *dynamic* variant of the predecessor problem where both queries and updates may be performed. Boninger *et al.* [BBK17] and Ramamoorthy and Rao [RR18] present non-adaptive query lower bounds for the dynamic predecessor problem of $\Omega(\log m / \log w)$ by and $\Omega(\log m / (\log \log m + \log w))$ respectively. In the natural setting of linear space $s = O(n)$ and each point in the universe fits into a single cell $w = \Theta(\log m)$, our lower bounds peak at $\Omega(\log m)$ which is higher than both of the previous lower bounds. Our lower bounds are tight with the non-adaptive (dynamic) predecessor construction by Boninger *et al.* [BBK17] in the linear space regime.

Unlike the previous results, our lower bounds apply to static data structures. Note that proving static lower bounds has traditionally been much more difficult than dynamic lower bounds. For example, the best adaptive dynamic lower bounds are $\tilde{\Omega}(\log^2 n)$ [Lar12a] while the best adaptive static lower bounds are $\tilde{\Omega}(\log n)$ [Lar12b]. In fact, there is some evidence that proving super-logarithmic static lower bounds might be very difficult [DGW18, Vio19] in various settings. Furthermore, we note that our lower bounds match the highest known static lower bounds in [Lar12b].

Finally, we note that our lower bounds are able to separate the dictionary problem for data structure with one round of adaptivity as opposed to two rounds. There exist linear space dictionary schemes such as cuckoo hashing [PR04] that use only two rounds of adaptivity and achieve $O(1)$ query overhead (in fact, just three probes). On the other hand, our lower bounds show that any non-adaptive linear dictionary must have query running times of $\Omega(\log m / \log(w / \log m))$ matched by the construction of Boninger *et al.* [BBK17]. In other words, we show that two rounds of adaptivity is necessary to achieve constant or even $o(\log m)$ query overhead implying that many construction that guarantee constant query complexity, such as cuckoo hashing [PR04], are indeed optimal in terms of adaptivity.

1.2 Technical Overview

To understand the techniques used in our paper, we start by describing the *cell sampling* technique that is used to prove static cell probe lower bounds for data structures.

Cell Sampling Techniques [PTW10, Lar12a]. Panigrahy *et al.* [PTW10] introduced the cell sampling technique to prove static cell probe lower bounds for the approximate-near-neighbor search problem in the adaptive setting. At a high level, [PTW10] samples a small number of memory cells S greedily in several rounds corresponding to the number of probes performed. In

each round, the cells with the highest probability of being probed by a query chosen uniformly at random from some query set Q are selected. If the number of probes (and, thus, rounds) is small, it can be shown that there exists a large subset of *resolved* queries $Q(S) \subseteq Q$ that only probe cells from the cell set S. We note that this technique shares similarities with the round elimination technique (see [Ajt88, Mil94, MNSW98]). In the next step of the proof, it is shown that the information revealed by answers of resolved queries in $Q(S)$ about the underlying random dataset $\mathbf{D}$ is larger than the maximum information that can be stored in the sampled cells S. As a result, a contradiction is presented showing that such a data structure with a small number of cell probes cannot exist.

Larsen [Lar12b] introduced a "single-round" variant of the cell sampling technique to prove stronger lower bounds for polynomial evaluation. In particular, the "single-round" cell sampling technique will pick all the sampled cells in S uniformly at random. Since all cells are picked in one round, the number of cells sampled is smaller than the multi-round variant used by [PTW10] resulting in higher lower bounds.

We note that the cell sampling technique has been used in several prior other works [Lar12a, Lar12b, LMWY19] together with the chronogram [FS89] to prove $\tilde{\Omega}(\log^2 n)$ dynamic lower bounds.

Our Cell Sampling Technique for Non-Adaptive Data Structures. We now show our crucial modifications to the cell sampling technique to enable proving higher lower bounds for non-adaptive data structures in the static setting. From now on, we will use cell sampling to refer to the one-round cell sampling variant of Larsen [Lar12b]. To do this, we carefully examine the qualifiers used in the cell sampling technique for adaptive data structures.

First, let us fix a query set Q. The cell sampling technique (both the multi-round and single-round variants) shows that, if a data structure is more efficient than a certain bound, then for every dataset D, there exists a small subset of cells $S(D)$ such that a large subset of resolved queries $Q(S(D)) \subseteq Q$ exist. In particular, the small subset of cells $S(D)$ is a function of the dataset D. While cell sampling shows such a subset of cells exist for every possible input dataset, the subsets of cells are not necessarily identical for each dataset. Formally, for two different datasets $D_1 \neq D_2$, there is no guarantee that $S(D_1)$ and $S(D_2)$ will be the same. In fact, we show that it is easy to design adaptive data structures such that $S(D_1)$ and $S(D_2)$ will be different. Recall in fact that adaptive data structures are allowed to probe a single cell and use the contents of the probed cell to determine which cell to probe next. Since the contents of cells can be highly correlated with the underlying dataset, the probed cells can also be highly correlated with the underlying dataset. Furthermore, the probed cells can change drastically by changing the underlying stored dataset. As an extreme example, consider a data structure that stores the w-bit string 0^w in cell location 0 for D_1 and 1^w for D_2. For each query, the data structure reads the cell location 0. If it sees 0^w, it will only query cells with locations at most $s/2$. On the other hand, if it sees 1^w, the data structure queries cells with locations strictly larger than $s/2$. In this case, the small subset of cells for each of D_1 and D_2 will necessarily be different, $S(D_1) \neq S(D_2)$. However, it seems that this property arises due to the adaptivity of the data structure.

Moving back to the case of non-adaptive data structures, the cells probed when answering a query are dependent only on the query input. In particular, the probed cells are completely independent of any cell content and, thus, completely independent of the stored dataset. Using this fact, we can switch the qualifiers of the cell sampling technique for non-adaptive data structures. Formally, we show that there exists a small subset of cells S that resolves a large subset of queries

$Q(S) \subseteq Q$ for all datasets D. This modification enables us to separate the techniques for adaptive and non-adaptive data structures. For the next step, we can now construct a random, hard dataset distribution $\mathbf{D}$ for the data structure using the knowledge of the resolved query subset $Q(S)$. In more detail, we can construct the random dataset distribution $\mathbf{D}$ such that the answers of the resolved queries reveal significant information about the random dataset. To finish the proof, we will show that the information revealed by resolved queries is larger than the maximum amount of information that may be stored in the sampled cell subset S.

To compare the adaptive and non-adaptive cell sampling techniques, we note that big difference is how the hard dataset distribution may be constructed. For the adaptive case, the hard dataset distribution $\mathbf{D}$ must be chosen without any knowledge of the resolved queries $Q(S)$. Then, it must be shown that for any resolved query subset $Q(S)$, it is possible to extract significant information about the random dataset $\mathbf{D}$. On the other hand, our non-adaptive cell sampling technique may construct the hard dataset distribution with knowledge of the resolved queries and the explicit goal of ensuring resolved queries extract large information about $\mathbf{D}$. We leverage these new ideas to prove higher static lower bounds for non-adaptive data structures.

1.3 Related Work

We note that several previous works have studied the cost of non-adaptivity in various setting. Katz and Trevisan [KT00] presented lower bounds for locally decodable codes where decoding queries for any bit of the original message must be performed non-adaptively and succeed even when some fraction of the encoded message is corrupted. Several works [AF09, RSS10, LMNR14, GR15, GR17] present non-adaptive bit probe upper and lower bounds for the static dictionary problem when the number of probes is very small. Brody and Larsen [BL12] prove polynomial non-adaptive lower bounds for dynamic data structures. In particular, they prove a lower bound for the multiphase problem (a dynamic version of the set disjointness problem) which was shown to imply several important graph data structure problems by Pătraşcu [Pat10]. Boninger *et al.* [BBK17] presented dynamic cell probe lower bounds for the non-adaptive predecessor problem. In parallel, Ramamoorthy and Rao [RR18] present dynamic cell probe lower bounds for the non-adaptive data structures with median, minimum or predecessor queries. In particular, they are able to show stronger tradeoffs for insertion/deletion operations.

2 Preliminaries

In this section, we formally define the notion of a randomized non-adaptive data structure and formally define the classical dictionary problem that maintains a subset of n points from the universe $[m]$.

2.1 Non-Adaptive Data Structures

A static, randomized, non-adaptive data structure consists of three randomized algorithms: the $\mathsf{preprocess}$, probe and $\mathsf{compute}$ algorithms. The $\mathsf{preprocess}$ algorithm takes a data set D and determines the content of the memory cells denoted by $M[1], \ldots, M[s]$ that encodes the data set D where s denotes the number of memory cells used by $\mathsf{preprocess}$. Algorithm probe receives the query q and outputs a set of cell addresses $\mathsf{probe}(q) \subseteq [s]$ whose contents will be probed. Finally, algorithm $\mathsf{compute}$ receives the query q as well as the contents of the memory cells chosen by $\mathsf{probe}(q)$,

$\{M[i] \mid i \in \mathsf{probe}(q)\}$, and computes the answer to the query q. The non-adaptive query algorithm consists of executing probe to choose the probed cells and then running compute to compute the answer using the contents of probed cells. Note, as probe only has access to the query q and not the memory M, the algorithm is forced to pick all probed cells non-adaptively.

For the dictionary problem, the data set is a subset of the universe $D \subseteq [m]$. A query q is an element from the universe $q \in [m]$ and the answer to query q is 1 if and only if $q \in D$.

Definition 2. *A static, randomized, non-adaptive data structure consists of the algorithms* preprocess *and* query $= (\mathsf{probe}, \mathsf{compute})$ *used in the following manner:*

1. $M \leftarrow \mathsf{preprocess}(D)$*: Preprocess a data set D to construct the memory cells to be used by the* query *algorithm.*

2. $\mathsf{compute}(q, \{M[i] \mid i \in \mathsf{probe}(q)\}) \leftarrow \mathsf{query}(q, M)$*: The query algorithm runs* $\mathsf{probe}(q) \subseteq [|M|]$ *to choose a subset of memory cells in M to probe. The* compute *algorithm uses the contents of probed memory to compute an answer to the query q.*

Next, we define the expected query cost for a randomized, static, non-adaptive data structure. We denote by $S_{\mathbf{DS}}(q, D, R)$ the set of cells probed by **DS** for query q over data set D using R as the fixed coin tosses. As we only consider non-adaptive data structures **DS**, the set of cells is independent from the data set D. Therefore, we drop D from the notation and denote the set of probed cells as $S_{\mathbf{DS}}(q, R)$ for a query q and random coin tosses R for **DS**. This set of probed cells is identical for all data sets D. We use $T_{\mathbf{DS}}(q, R)$ to denote the number of cells probed by **DS** for query q when using R as random coin tosses. Therefore, $T_{\mathbf{DS}}(q, R) = |S_{\mathbf{DS}}(q, R)|$.

We denote $\mathbf{R}$ as the random distribution used to draw the each coin toss uniformly at random used by **DS**. Thus, with a slight abuse of notation, $S_{\mathbf{DS}}(q, \mathbf{R})$ is the distribution of $S_{\mathbf{DS}}(q, R)$ with R chosen according to $\mathbf{R}$ and it coincides with the distribution of the output of algorithm probe on input q. Similarly, we use $T_{\mathbf{DS}}(q, \mathbf{R})$ as the distribution of the number of probes chosen by probe on input query q.

Finally, we define *expected query cost* by

$$T_{\mathbf{DS}} := \max_q \mathbb{E}_{\mathbf{R}}[T_{\mathbf{DS}}(q, \mathbf{R})].$$

In other words, $T_{\mathbf{DS}}$ is the maximum over all queries q of the expected number of cell probes executed by **DS** on query q where the expectation is taken over the random coin tosses of **DS**. We prove our lower bound on $T_{\mathbf{DS}}$.

Definition 3. *The expected query cost of a static non-adaptive data structure $T_{\mathbf{DS}}$ is defined as the maximum over any query q of the expected number of cells probed when answering q over the randomness of the internal coin tosses $\mathbf{R}$ of the data structure.*

2.2 Dictionary Problem

The dictionary problem is a fundamental data structure problem that has been well-studied in the past. The goal of the static version of the dictionary problem is to preprocess a subset D of n *points* from the universe $[m]$ so to enable efficient handling of *membership* queries; that is, given point $q \in [m]$, the dictionary query on q returns whether q appears in D or not.

Definition 4 (Static Dict_m)**.** *The static dictionary problem over the set* $[m]$*, denoted by* Dict_m*, asks to design a data structure* **DS** *to maintain a subset* $D \subseteq [m]$ *with the following two operations:*

1. $\mathsf{preprocess}(D), D \subseteq [m]$: *Preprocess a subset of* n *items* $D \subseteq [m]$ *by writing an* encoding *of* D *into* s *cells* $M[1], \ldots, M[s]$*;*

2. $\mathsf{query}(q), q \in [m]$: *Return* 1 *if and only if* $q \in D$*.*

   We consider the $\mathsf{query}(q)$ *algorithm as consisting of two steps. First, the* $\mathsf{probe}(q)$ *algorithm selects the cells to probe. Next, the* compute *algorithm, on input* q *and the contents of probed cells* $\{M[i] \mid i \in \mathsf{probe}(q)\}$*, computes the answer to the query.*

3 Non-Adaptive Dictionary Lower Bound

In this section, we present lower bounds for static data structures using non-adaptive queries that solve the dictionary problem. We will denote by n the number of points taken from universe $[m]$ where $m = n^{1+\Omega(1)}$. Note, we make no assumption on the cell size and thus our results also apply in the bit probe model.

We will present a lower bound for a *non-adaptive* randomized data structure **DS** that solves the *dictionary problem* over a *static* set of data. In the next theorem, we formally state our lower bound for the expected query cost $T_{\mathbf{DS}}$ of a non-adaptive **DS**. An informal statement of the same result is given in the introduction as Theorem 1.

Theorem 5 (Main result)**.** *Let* **DS** *be a non-adaptive randomized data structure that implements the static dictionary problem for a data set of* n *elements from the universe* $[m]$ *over* w*-bit cells and that uses* s *cells of memory. If* $m = n^{1+\Omega(1)}$*, then*

$$T_{\mathbf{DS}} = \Omega\left(\frac{\log m}{\log\left(\frac{sw}{n \log m}\right)}\right).$$

In the natural setting where $m = \mathsf{poly}(n)$, $w = \Theta(\log m)$ and **DS** uses $s = \Theta(n)$ cells of storage, we obtain $T_{\mathbf{DS}} = \Omega(\log n)$.

To prove our lower bound, we use a modification of the *cell sampling* technique introduced by Panigrahy *et al.* [PTW10] that is suitable for use with non-adaptive data structures. In particular, we will use a "single-round" variant of cell sampling that was introduced by Larsen [Lar12a] where sampling occurs in a single-round as opposed to multiple rounds as in [PTW10]. Roughly speaking, the proof will show that, for a data structure with small expected query cost, there must exist a small subset of server cell addresses $S^\star$ such that many of the queries only probe server cells with addresses in the set $S^\star$. Thus, we can encode the answer to these queries using only the content of the cells with addresses in $S^\star$. If the replies to these queries have high entropy and $S^\star$ is small, we will reach a contradiction.

We now present our variant of cell sampling, inspired by the ideas of [PTW10] and [Lar12a], that is designed specifically for non-adaptive data structures. For a set S of cells and a fixed choice of coin tosses R, we let $Q_{\mathbf{DS}}(S, R)$ be the set of queries q that are *resolved* by the set of cells S when **DS** uses R as coin tosses. A query q is resolved by S if all cells probed by **DS** on query q are in the set S. Formally, $q \in Q_{\mathbf{DS}}(S, R)$ if and only if $S_{\mathbf{DS}}(q, R) \subseteq S$.

Lemma 1 (Non-Adaptive Cell Sampling.)**.** *Assume there exists a* **DS** *with expected query cost*

$$T_{\mathbf{DS}} = o\left(\frac{\log m}{\log \frac{ws}{n \log m}}\right).$$

For all constants $0 < \alpha$*, there exists a fixed choice of coin tosses* R *and set of cell addresses* $S^\star$ *of size at most* $\alpha \cdot \frac{n \log m}{w}$ *such that*

$$|Q_{\mathbf{DS}}(S^\star, R)| \geq \frac{|Q|}{m^{o(1)}}.$$

Before we present the proof of Lemma 1, we will describe its implication. Lemma 1 shows that for **DS** that beat the bound there exists a small set $S^\star$ of at most $\alpha n \log m/w$ server cells that can be used to resolve a large subset of a set Q of $|Q|/m^{o(1)}$ queries. We stress that this set $S^\star$ of cells is independent from the data set D by critically using the fact that **DS** is non-adaptive.

Let us suppose that we had chosen our data sets according to a random distribution **D** that generates n elements from $[m]$ uniformly and independently at random. Since **D** is chosen uniformly at random, we know that the entropy of **D** is $H(\mathbf{D}) = n \log m$ bits. Yet, the queries in $Q_{\mathbf{DS}}(S^\star, R)$ only have access to at most

$$|S^\star| w \leq \alpha n \log m < n \log m = H(\mathbf{D})$$

bits of information of **D** by choosing $\alpha < 1$. As a result, we would be able to derive a contradiction.

However, we had just implicitly assumed that the answers to the set of resolved queries would reveal all n elements in **D**. It is not clear that this will always be the case. As an example, suppose that the set of resolved queries never intersect any of the n elements chosen randomly from **D**. In this case, none of the n elements drawn from **D** are revealed. In other words, the answers to the set of resolved queries in $Q_{\mathbf{DS}}(S^\star, R)$ do not reveal enough information about **D** to derive a contradiction. Therefore, we have to design our hard distribution carefully to ensure that resolved queries retrieve large amounts of information about **D**.

3.1 Proof of Theorem 5

Let us now proceed more formally. We start by our hard distribution over data sets that will be stored by the dictionary data structure. Towards a contradiction, we will suppose that there exists a non-adaptive data structure **DS** that solves the static dictionary problem such that the expected cost of **DS** is

$$T_{\mathbf{DS}} = o\left(\frac{\log m}{\log\left(\frac{sw}{n \log m}\right)}\right).$$

Next we will show that, based on **DS**, it is possible to encode a random data set from the data set distribution by using fewer bits than its entropy, thus reaching a contradiction. Therefore, for any **DS**, we have a lower bound on the expected query cost with respect to a query and data set distribution. This implies a lower bound on $T_{\mathbf{DS}}$ as it is the worst case over all choices of queries.

Hard Distribution. First, we apply Lemma 1 to the set $Q := [m]$ of all queries thus obtaining a fixed choice of random coin tosses R and cell addresses $S^\star$. Specifically, by Lemma 1, there exist a small subset of server cell addresses $S^\star$ and coin tosses R such that the set of resolved

queries $Q_{\mathbf{DS}}(S^\star, R)$ contains at least least $m^{1-o(1)}$ of the m queries of Q. Afterwards, we partition $Q_{\mathbf{DS}}(S^\star, R)$ into n equal parts, which we denote by $Q_1, \ldots, Q_n$. We know that each part contains at least $m^{1-o(1)}/n \geq m^{1-o(1)-1/(1+\epsilon)}$, where we write $m = n^{1+\epsilon}$, for some $\epsilon = \Omega(1)$. Since $\epsilon = \Omega(1)$, there exists a positive constant $\rho := \Theta(1 - o(1) - 1/(1+\epsilon))$ such that each of the n parts of $Q_{\mathbf{DS}}(S^\star, R)$ contains at least m^ρ queries. We partition $Q_{\mathbf{DS}}(S^\star, R)$ by placing the smallest m^ρ elements into Q_1, the second smallest m^ρ elements into Q_2 and so forth. These $n \cdot m^\rho$ queries in $Q_1, \ldots, Q_n$ will be our queries of interest used in our proof. We note that these resolved queries $Q_1, \ldots, Q_n$ do not depend on the underlying data set stored by **DS**. $Q_1, \ldots, Q_n$ are derived using only the fixed coin tosses R and cell addresses $S^\star$ from Lemma 1, which critically relies on the fact that **DS** is non-adaptive. As a result, $Q_1, \ldots, Q_n$ are always resolved regardless of the underlying data set stored by **DS**.

Next, we define the hard distribution of data sets $\mathbf{D}$ as the product of n independent distributions $\mathbf{D} := \mathbf{D}_1 \times \ldots \times \mathbf{D}_n$, where distribution $\mathbf{D}_i$ selects an element from the query set Q_i uniformly at random. Note that given the answers to all queries in Q_i, one will be able to uniquely decode the random element chosen by Q_i. Exactly one query q from Q_i will return 1 while all the other queries will return 0. Then, we know that q was the random element drawn by $\mathbf{D}_i$.

Using our choice of hard data set distribution $\mathbf{D}$ and queries of interest $q_1, \ldots, q_n$, we can now present a contradiction that the contents of server cells with addresses in $S^\star$ contains far too little bits of information about $\mathbf{D}$ given that **DS** has small expected query cost. To formally prove this statement, we will use a one-way communication protocol between an encoder (Alice) and a decoder (Bob) using **DS**. Both Alice and Bob will receive the set of cell addresses, $S^\star$, as well as the fixed coin tosses R chosen by Lemma 1. Alice will attempt to efficiently communicate to Bob a set of n items chosen according to $\mathbf{D}$ that Bob does not receive. By Shannon's source coding theorem, we know that the size of any encoding by Alice must be at least the entropy of $\mathbf{D}$ conditioned on the Bob's input, that is, $H(\mathbf{D} \mid S^\star, R)$. We show that by assuming a very efficient **DS** along with Lemma 1, we will construct an impossible encoding. Before doing so, we present a lemma analyzing the entropy of $\mathbf{D}$ conditioned on Bob's input, $S^\star$ and R.

Lemma 2. *The entropy of* $\mathbf{D}$ *conditioned on* $S^\star$ *and* R *is* $H(\mathbf{D} \mid S^\star, R) = \rho \cdot n \log m$.

Proof. Since $\mathbf{D}$ is a product distribution, we know that

$$H(\mathbf{D} \mid S^\star, R) = \sum_{i=1}^{n} H(\mathbf{D}_i \mid S^\star, R).$$

Note that each $\mathbf{D}_i$ chooses a random element uniformly from the set of Q_i that contains m^ρ elements. The set of cells addresses $S^\star$ only defines the set Q_i that $\mathbf{D}_i$ is choosing the uniformly random element. As a result, $H(\mathbf{D}_i \mid S^\star, R) = \log(m^\rho) = \rho \cdot \log m$. □

We now complete the proof of Theorem 5 by presenting our encoding scheme. Note that since the addresses of the sampled cells $S^\star$ are given as input to both Alice and Bob, we can prove lower bounds in the bit probe model. Previous cell sampling proofs [Lar12a, Lar12b] required Alice to communicate the addresses of $S^\star$ which required the assumption that the size of cells were not smaller than the number of bits needed to communicate the cell addresses. In other words, these proofs required $w = \Omega(\log n)$ such that each cell could fit one cell address meaning they could not work for the bit-probe model unlike our proof.

Alice's Encoding. Alice will receive both the fixed coin tosses R and the set of cell addresses $S^\star$ whose existence is guaranteed by Lemma 1. Additionally, Alice receives the data set $D = (d_1, \ldots, d_n)$ drawn from $\mathbf{D}$.

1. Alice executes the preprocessing algorithm of **DS** to compute the set of memory cells $\mathsf{C}(D, R)$ used by **DS** for queries.
2. Alice encodes the contents of $\mathsf{C}(D, R)$ at the addresses in $S^\star$ using $w \cdot |S^\star|$ bits.

Bob's Decoding. Bob will receive Alice's encoding. In addition, Bob receives the same fixed coin tosses R drawn and the set of cell addresses $S^\star$ whose existence is guaranteed by Lemma 1 as Alice. Bob will attempt to decode the items $(d_1, \ldots, d_n)$ drawn from $\mathbf{D}$.

1. Bob computes the set $Q_{\mathbf{DS}}(S^\star, R)$ of queries resolved by $S^\star$. Bob iterates through every possible query $q \in [m]$, determines the set of cells addresses that would be probed by **DS** to execute query q using R as source of randomness, $S_{\mathbf{DS}}(q, R)$, and checks if it is a subset of $S^\star$. Note that this step does not require knowledge of the data set D as **DS** is non-adaptive. As a result, Bob computes the set $Q(S^\star, R)$.
2. Afterwards, Bob partitions $Q(S^\star, R)$ into n equal parts $Q_1, \ldots, Q_n$ by placing the smallest m^ρ elements of $Q(S^\star, R)$ into Q_1, the second smallest m^ρ elements of $Q(S^\star, R)$ into Q_2 and so forth.
3. Bob decodes the contents of $\mathsf{C}(D, R)$ at the addresses in $S^\star$ using Alice's encoding. For each $i = 1, \ldots, n$, Bob executes all queries in Q_i according to **DS** using R as source of randomness. Note that for any query $q \in Q_i$, the contents of all cells probed by **DS** to answer query q_i were encoded by Alice as $Q_i \subset Q_{\mathbf{DS}}(S^\star, R)$. As a result, Bob will decode d_i using the answers to the queries in Q_i. Therefore, Bob is able to successfully recover $(d_1, \ldots, d_n)$.

Analysis. We now analyze the length of Alice's encoding. Alice's encodes the contents of cells at addresses $S^\star$ which takes $|S^\star|w$ bits. Using the fact that $|S^\star| < \rho n \log m/w$ from Lemma 1 and by setting $\alpha < \min\{\rho, 1\}$, Alice's encoding length is at most

$$|S^\star(\mathbf{R})|w < \rho n \log m$$

which contradicts Shannon's source coding theorem.

Therefore, all that remains to complete the proof of Theorem 5 is to prove the cell sampling Lemma 1, which we do below.

3.2 Proof of Lemma 1

Let X_q be the 0-1 random variable that is 1 if and only if $q \in Q(S^\star, R)$. We will show that, when we choose $\mathbf{S}^\star$ uniformly at random from all subsets of exactly $\alpha n \log m/w$ cells from the s cells used by **DS** as storage, then $\mathbb{E}_{\mathbf{R},\mathbf{S}^\star}[X_q] \geq 1/m^{o(1)}$. Then, by linearity of expectation, we have

$$\mathbb{E}_{\mathbf{R},\mathbf{S}^\star}\left[|Q_{\mathbf{DS}}(\mathbf{S}^\star, \mathbf{R})|\right] = \sum_{q \in Q} \mathbb{E}_{\mathbf{R},\mathbf{S}^\star}\left[X_q\right] \geq \frac{|Q|}{m^{o(1)}}.$$

As a result, there exists a fixed choice of R and $S^\star$ such that $Q_{\mathbf{DS}}(S^\star, R)$ is size at least $|Q|/m^{o(1)}$.

We now prove that $\mathbb{E}_{\mathbf{R},\mathbf{S}^\star}[X_q] \geq 1/m^{o(1)}$. For convenience, we will use r as a shorthand for $\alpha \cdot \frac{n \log m}{w} = |\mathbf{S}^\star|$ and t as a shorthand for $T_{\mathbf{DS}}(q, \mathbf{R})$. For randomly chosen $\mathbf{R}$ and $\mathbf{S}^\star$ we have that

$$\mathbb{E}_{\mathbf{R},\mathbf{S}^\star}[X_q] = \frac{\binom{s-t}{r-t}}{\binom{s}{r}} \geq \frac{r \cdot (r-1) \cdots (r-t+1)}{s \cdot (s-1) \cdots (s-t+1)} \geq \left(\frac{r-t}{s}\right)^t.$$

As $r = \alpha n \log m/w$ and $t = T_{\mathbf{DS}}(q, \mathbf{R}) \leq \max_q \mathbb{E}_{\mathbf{R}} T_{\mathbf{DS}}(q, \mathbf{R}) = T_{\mathbf{DS}} = o(\log m/\log(sw/n \log m))$, we know that $t \leq r/2$ for sufficiently large n. So, we get that

$$\mathbb{E}_{\mathbf{R},\mathbf{S}^\star}[X_q] \geq \left(\frac{r}{2s}\right)^t.$$

Hence,

$$\mathbb{E}_{\mathbf{R},\mathbf{S}^\star}[X_q] \geq \mathbb{E}_{\mathbf{R},\mathbf{S}^\star}\left[\left(\frac{r}{2s}\right)^{T_{\mathbf{DS}}(q,\mathbf{R})}\right] \geq \left(\frac{r}{2s}\right)^{\mathbb{E}_{\mathbf{R}}[T(q,\mathbf{R})]}$$

where the last inequality follows by Jensen's inequality. Also note that $n \log m \leq s \cdot w$ and thus $r \leq s$, for every $\alpha \leq 1$. Since $r \leq s$, we can write

$$\begin{aligned}
\mathbb{E}_{\mathbf{R},\mathbf{S}^\star}[X_q] &\geq \left(\frac{r}{2s}\right)^{\max_q \mathbb{E}_{\mathbf{R}}[T_{\mathbf{DS}}(q,\mathbf{R})]} \\
&= \left(\frac{r}{2s}\right)^{T_{\mathbf{DS}}} \\
&= \left(\frac{\alpha}{2} \cdot \frac{n \log m}{sw}\right)^{o\left(\frac{\log m}{\log \frac{sw}{n \log m}}\right)} \\
&\geq \frac{1}{m^{o(1)}}.
\end{aligned}$$

4 Conclusions

In this paper, we essentially resolve the static cell probe complexity for non-adaptive data structures solving the fundamental dictionary problem. In particular, we present a dichotomy with respect to adaptivity. For data structures that are able to perform two rounds of adaptive retrievals from storage, there exists many well-known constructions with $O(1)$ number of cell probes such as cuckoo hashing [PR04]. On the other hand, as soon as one wishes to use exactly one round of retrieval from storage, dictionary data structures must use logarithmic number of probes matched by the constructions of Boninger *et al.* [BBK17].

As one can solve the dictionary problem using the predecessor problem, our results imply the same lower bounds for the predecessor problem. To our knowledge, all past non-adaptive lower bounds for predecessor only applied to dynamic setting. Our lower bounds improve on this dimension by applying to static data structures while also peaking higher than all previous predecessor lower bounds [BBK17, RR18] in the natural setting of linear space.

Our results can also be shown to apply to other important problems such as one-dimensional range search problems (such as sum, count, emptiness, reporting, etc.) by simple reductions through the dictionary problem. Additionally, our lower bounds have implications to depth-2 circuit size lower bounds as shown in [BL12].

Acknowledgements

The authors would like to thank Omri Weinstein for a helpful discussion.

References

[ABKU99] Yossi Azar, Andrei Z Broder, Anna R Karlin, and Eli Upfal. Balanced allocations. *SIAM journal on computing*, 29(1):180–200, 1999.

[AF09] Noga Alon and Uriel Feige. On the power of two, three and four probes. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 346–354. Society for Industrial and Applied Mathematics, 2009.

[Ajt88] Miklós Ajtai. A lower bound for finding predecessors in yao's cell probe model. *Combinatorica*, 8(3):235–247, 1988.

[AN96] Noga Alon and Moni Naor. Derandomization, witnesses for boolean matrix multiplication and construction of perfect hash functions. *Algorithmica*, 16(4-5):434–449, 1996.

[And96] Arne Andersson. Faster deterministic sorting and searching in linear space. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 135–141. IEEE, 1996.

[AT00] Arne Andersson and Mikkel Thorup. Tight (er) worst-case bounds on dynamic searching and priority queues. In *ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING*, volume 32, pages 335–342. Citeseer, 2000.

[BBK17] Joe Boninger, Joshua Brody, and Owen Kephart. Non-adaptive data structure bounds for dynamic predecessor search. 2017.

[BF02] Paul Beame and Faith E Fich. Optimal bounds for the predecessor problem and related problems. *Journal of Computer and System Sciences*, 65(1):38–72, 2002.

[BL12] Joshua Brody and Kasper Green Larsen. Adapt or die: Polynomial lower bounds for non-adaptive dynamic data structures. *arXiv preprint arXiv:1208.2846*, 2012.

[CW79] J Lawrence Carter and Mark N Wegman. Universal classes of hash functions. *Journal of computer and system sciences*, 18(2):143–154, 1979.

[DBCP97] Mikael Degermark, Andrej Brodnik, Svante Carlsson, and Stephen Pink. *Small forwarding tables for fast routing lookups*, volume 27. ACM, 1997.

[DGW18] Zeev Dvir, Alexander Golovnev, and Omri Weinstein. Static data structure lower bounds imply rigidity. *arXiv preprint arXiv:1811.02725*, 2018.

[DKM+94] Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auF der Heide, Hans Rohnert, and Robert E Tarjan. Dynamic perfect hashing: Upper and lower bounds. *SIAM Journal on Computing*, 23(4):738–761, 1994.

[FKS82] Michael L Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with o (1) worst case access time. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 165–169. IEEE, 1982.

[FS89] Michael Fredman and Michael Saks. The cell probe complexity of dynamic data structures. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 345–354. ACM, 1989.

[FW93] Michael L Fredman and Dan E Willard. Surpassing the information theoretic bound with fusion trees. *Journal of computer and system sciences*, 47(3):424–436, 1993.

[GR15] Mohit Garg and Jaikumar Radhakrishnan. Set membership with a few bit probes. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 776–784. Society for Industrial and Applied Mathematics, 2015.

[GR17] Mohit Garg and Jaikumar Radhakrishnan. Set membership with non-adaptive bit probes. In *34th Symposium on Theoretical Aspects of Computer Science (STACS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[Hag99] Torben Hagerup. Fast deterministic construction of static dictionaries. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 414–418. Society for Industrial and Applied Mathematics, 1999.

[KT00] Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *STOC*, pages 80–86. Citeseer, 2000.

[Lar12a] Kasper Green Larsen. The cell probe complexity of dynamic range counting. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 85–94. ACM, 2012.

[Lar12b] Kasper Green Larsen. Higher cell probe lower bounds for evaluating polynomials. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 293–301. IEEE, 2012.

[LMNR14] Moshe Lewenstein, J Ian Munro, Patrick K Nicholson, and Venkatesh Raman. Improved explicit data structures in the bitprobe model. In *European Symposium on Algorithms*, pages 630–641. Springer, 2014.

[LMWY19] Kasper Green Larsen, Tal Malkin, Omri Weinstein, and Kevin Yeo. Lower bounds for oblivious near-neighbor search. *CoRR*, abs/1904.04828, 2019.

[LWY18] Kasper Green Larsen, Omri Weinstein, and Huacheng Yu. Crossing the logarithmic barrier for dynamic boolean data structure lower bounds. In *STOC 2018 Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, 2018.

[Mil94] Peter Bro Miltersen. Lower bounds for union-split-find related problems on random access machines. In *STOC*, volume 94, pages 625–634, 1994.

[Mil98] Peter Bro Miltersen. Error correcting codes, perfect hashing circuits, and deterministic dynamic dictionaries. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 556–563. Society for Industrial and Applied Mathematics, 1998.

[MNSW98] Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences*, 57(1):37–49, 1998.

[Pag00] Rasmus Pagh. Faster deterministic dictionaries. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 487–493. Society for Industrial and Applied Mathematics, 2000.

[Pat10] Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 603–610. ACM, 2010.

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

[PPR07] Anna Pagh, Rasmus Pagh, and Milan Ruzic. Linear probing with constant independence. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 318–327. ACM, 2007.

[PR04] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004.

[PT06] Mihai Pătraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 232–240. ACM, 2006.

[PT11] Mihai Patrascu and Mikkel Thorup. The power of simple tabulation hashing. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 1–10. ACM, 2011.

[PTW10] Rina Panigrahy, Kunal Talwar, and Udi Wieder. Lower bounds on near neighbor search via metric expansion. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 805–814. IEEE, 2010.

[RR18] Sivaramakrishnan Natarajan Ramamoorthy and Anup Rao. Lower bounds on non-adaptive data structures maintaining sets of numbers, from sunflowers. In *33rd Computational Complexity Conference*, 2018.

[RSS10] Jaikumar Radhakrishnan, Smit Shah, and Saswata Shannigrahi. Data structures for storing small sets in the bitprobe model. In *European Symposium on Algorithms*, pages 159–170. Springer, 2010.

[Sen03] Pranab Sen. Lower bounds for predecessor searching in the cell probe model. In *18th IEEE Annual Conference on Computational Complexity, 2003. Proceedings.*, pages 73–83. IEEE, 2003.

[TY79] Robert Endre Tarjan and Andrew Chi-Chih Yao. Storing a sparse table. *Communications of the ACM*, 22(11):606–611, 1979.

[vEB77] Peter van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information processing letters*, 6(3):80–82, 1977.

[Vio19] Emanuele Viola. Lower bounds for data structures with space close to maximum imply circuit lower bounds. *Theory of Computing*, 15(1):1–9, 2019.

[Wil83] Dan E Willard. Log-logarithmic worst-case range queries are possible in space θ (n). *Information Processing Letters*, 17(2):81–84, 1983.

[WY16] Omri Weinstein and Huacheng Yu. Amortized dynamic cell-probe lower bounds from four-party communication. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 305–314. IEEE, 2016.

[Yao81] Andrew Chi-Chih Yao. Should tables be sorted? *Journal of the ACM (JACM)*, 28(3):615–628, 1981.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il