Lower Bounds for Oblivious Near-Neighbor Search

Kasper Green Larsen* Tal Malkin† Omri Weinstein‡ Kevin Yeo§

Abstract

We prove an \( \Omega(d \log n / (\log \log n)^2) \) lower bound on the dynamic cell-probe complexity of statistically oblivious approximate-near-neighbor search (ANN) over the \( d \)-dimensional Hamming cube. For the natural setting of \( d = \Theta (\log n) \), our result implies an \( \tilde{\Omega} (\log^2 n) \) lower bound, which is a quadratic improvement over the highest (non-oblivious) cell-probe lower bound for ANN. This is the first super-logarithmic unconditional lower bound for ANN against general (non black-box) data structures. We also show that any oblivious static data structure for decomposable search problems (like ANN) can be obliviously dynamized with \( O(\log n) \) overhead in update and query time, strengthening a classic result of Bentley and Saxe (Algorithmica, 1980).

*\texttt{larsen@cs.au.dk}. Aarhus University. Work supported by a Villum Young Investigator Grant and an AUFF Starting Grant.
†\texttt{tal@cs.columbia.edu}. Columbia University. Work supported in part by the Leona M. & Harry B. Helmsley Charitable Trust.
‡\texttt{omri@cs.columbia.edu}. Columbia University. Work supported by NSF CAREER Award CCF-1844887.
§\texttt{kwlyeo@google.com}. Google LLC.
1 Introduction

The nearest-neighbor search problem asks to preprocess a dataset \( P \) of \( n \) input points in some \( d \)-dimensional metric space, say \( \mathbb{R}^d \), so that for any query point \( q \) in the space, the data structure can quickly retrieve the closest point in \( P \) to \( q \) (with respect to the underlying distance metric). The \( r \)-near-neighbor problem is a relaxation of the nearest-neighbor problem, which requires, more modestly, to return any point in the dataset within distance \( r \) of the query point \( q \) (if any exists). The distance parameter \( r \) is typically referred to as the radius. Efficient algorithms (either offline or online) for both the nearest-neighbor and \( r \)-near-neighbor problems are only known for low-dimensional spaces [Cla88, Mei93], as the only known general solutions for these problems are the naive ones: either a brute-force search requiring \( O(dn) \) time (say, on a word-RAM), or precomputing the answers which requires prohibitive space exponential in \( d \). This phenomenon is commonly referred to as the “curse of dimensionality” in high-dimensional optimization. This obstacle is quite problematic as nearest-neighbor search primitives are the backbone of a wide variety of industrial applications as well as algorithm design, ranging from machine learning [SDI06] and computer vision [HS12], to computational geometry [CDH02], spatial databases [Tya18] and signal processing [MPL00] as some examples.

To circumvent the “curse of dimensionality”, a further relaxation of the near(est)-neighbor problem was introduced, resorting to approximate solutions, which is the focal point of this paper. In the \( (c,r) \)-approximate-near-neighbor problem (ANN), the data structure needs to return any point in \( P \) that is distance at most \( cr \) from the query point \( q \), assuming that there exists at least one data point in \( P \) that is within distance at most \( r \) from the query. If all points in the data set \( P \) are distance greater than \( cr \) from the query point \( q \), no point will be reported. In other words, \( (c,r) \)-ANN essentially asks to distinguish the two extreme cases where there exists a point in \( P \) which is at most \( r \)-close to the query point \( q \), or all points in \( P \) are at \( \geq cr \)-far from \( q \). Perhaps surprisingly, the geometric “gap” in this promise version of the problem turns out to be crucial, and indeed evades the “curse of dimensionality”. A long and influential line of work in geometric algorithms based on locality sensitive hashing (or, LSH, for short) techniques [IM98, Pan06] show that the search time for this promise problem (under various \( \ell_p \) norms) can be dramatically reduced from \( \sim n \) to \( n^\delta \) (for a small constant \( \delta \) depending on \( r \) and \( c \)) at the cost of a mild space overhead of \( n^{1+\epsilon} \) or even \( n \text{polylog} n \) in the static setting. Interestingly, these upper bounds extend to the more challenging and realistic dynamic setting where points in the dataset arrive online, yielding a dynamic data structure with \( \text{polylog} n \) update time and \( n^\delta \) query time [Pan06]. For a more detailed exposition of the state-of-the-art on ANN, we refer the reader to the following surveys [AI06, And09].

On the lower bound side, progress has been much slower. While there has been a considerable amount of work on the limits of ANN in black-box models of computation with “no-coding” assumptions (e.g., [BV02, KL05]), the highest unconditional lower bound to date is the \( \Omega(d/\lg(sw/nd)) \) query time lower bound for any static data structure by Wang and Yin [WY14] as well as Yin [Yin16], extending previous results of [PT06, ACP08, PTW08, PTW10], where \( s \) denotes the data structure’s storage in cells and \( w \) is the word size in bits. This is also the highest cell-probe lower bound to date in the dynamic setting – the aforementioned bound implies that any (randomized) dynamic data structure for ANN with fast (\( \text{polylog} n \)) update time must have \( \tilde{\Omega}(d) \) query time. This is in contrast to typical data structure problems, where online lower bounds are known to be higher than their static counterparts. While this bound is exponentially far from the aforementioned upper bounds, a recurring theme in complexity theory is that information-theoretic lower bounds are significantly more challenging compared to black-box bounds, and hence lower. It
is widely believed that the logarithmic lower bound is far from tight, especially in the fully dynamic setting. Indeed, Panigrahy et al. [PTW10] conjecture that the dynamic cell-probe complexity of ANN should be *polynomial*, but could only prove this for LSH-type data structures (a.k.a. “low contention”) where no single cell is probed too often. There are also *conditional* (“black-box”) lower bounds asserting that polynomial $\Omega(n^\epsilon)$ operational time is indeed necessary for the *offline* version of ANN, under the Strong Exponential-Time Hypothesis ([ARW17, Wil18, Rub18]).

**Privacy-Preserving Near-Neighbor Search.** Due to the increasing size of today’s datasets, an orthogonal line of research has been studied for *privacy-preserving near-neighbor search*. In this scenario, the dataset of points have been outsourced by a client to a third-party server such as a cloud storage provider. The client would like to be able to perform near(est)-neighbor search queries over the outsourced set of data points. However, the storage of potentially sensitive data onto an untrusted third-party brings many privacy concerns. This leads to the natural problem of whether a client is able to outsource a data set of points to an untrusted server while maintaining the ability to perform private near(est)-neighbor queries over the data set efficiently.

One aspect of privacy is protecting the content of the outsourced data set. This problem can be addressed by encryption where the client holds the secret key. However, the use of encryption does not protect information leaked by observing the patterns of access to server memory. Towards that end, the client may wish to implement *oblivious access* where the patterns of access to server memory is independent of both the content of the data set as well as the queries performed by the client. In order to focus on the latter problem, we assume the server’s view only contains the patterns of access to server memory. Informally, $\delta$-statistical obliviousness implies that for any two operation sequences of equal length $O_1$ and $O_2$, it must be that $|V_D(O_1) - V_D(O_2)| \leq \delta$ where $V_D(O)$ is the distribution of access patterns to server memory by the data structure $D$ executing $O$. This can later be combined with standard computational assumptions and cryptographic encryption or information-theoretic encryption via one-time padding (if the client can either hold or securely store a random pad) to ensure privacy of the data set contents.

To address the problem of protecting access patterns, the *oblivious RAM* (ORAM) primitive was introduced by Goldreich and Ostrovsky [GO96]. ORAM considers the scenario where the server holds an array and the client wishes to either retrieve or update various elements in the array while guaranteeing oblivious access. ORAMs are very powerful as they provide a simple transformation from any data structure into an oblivious data structure. By executing every access to server memory of any non-oblivious data structure using an ORAM, the access pattern of the resulting data structure ends up being oblivious. Due to the importance of ORAM, there has been a long line of work constructing ORAMs. For example, we refer the reader to some examples: [PR10, DMN11, GM11, GMOT12, KLO12, SVDS+13, CLP14, GHL+14, BCP16, CT16, GLOS15]. Recently, this wave of research led to the $O(\lg n \cdot \lg \lg n)$ ORAM construction by Patel et al. [PPRY18], and, finally, an $O(\lg n)$ ORAM by Asharov et al. [AKL+18]. Therefore, we can build an oblivious data structure with an additional logarithmic overhead compared to the best non-oblivious data structure.

There has also been significant work on the lower bound of ORAMs. Goldreich and Ostrovsky [GO96] present an $\Omega(\lg n)$ for ORAMs in the restricted setting of “balls-and-bins” model (i.e. a “non-coding” assumption) and statistical security. Larsen and Nielsen [LN18] extended the $\Omega(\lg n)$ lower bound to the cell-probe model and computational security matching the aforementioned upper bounds. Additionally, works by Boyle and Naor [BN16] as well as Weiss and Wichs [WW18] show that any non-trivial lower bounds for either offline or online, read-only ORAMs would imply
huge breakthroughs in lower bounds for sorting circuits and/or locally decodable codes.

Going back to the problem of privacy-preserving near-neighbor search, many works in the past decade [KS07, MCA07, GKK+08, WCKM09, PBP10, YLX13, ESJ14, LSP15, WHL16] attempt to circumvent the additional efficiency overhead incurred by ORAM. Instead of ensuring oblivious access where the access patterns are independent of the data set and queries, the access patterns of many constructions from previous works end up leaking non-trivial amounts of information. For example, the access patterns in the constructions by Wang et al. [WHL16] leak the identity of the point reported by queries. In more detail, as their work considers the k-nearest-neighbor problem, their algorithms leak the identity of the k encrypted points that are closest to the query point. Recent work by Kornaropoulos et al. [KPT18] has shown that this non-trivial leakage can be abused to accurately retrieve almost all private data. As a result, the requirement of oblivious access is integral to ensure privacy for the near-neighbor problem. Therefore, several works consider variants of near-neighbor search with oblivious access such as [EFG+09, SSW09, BBC+10, EHKM11, SFR18, AHLR18, CCD+19] to name a few.

An intriguing question is whether the extra $\Theta(\log n)$ overhead for oblivious data structures over their non-oblivious counterparts is really necessary. For the problem of RAMs, it has been shown that the $\Theta(\log n)$ overhead is both necessary and sufficient [LN18, PY18]. Jacob et al. [JLN19] also show that the $\Theta(\log n)$ overhead is necessary and sufficient for many fundamental data structures such as stacks and queues, but quite surprisingly, Jafargholi et al. [JLS19] very recently showed that (comparison-based) priority queues can be made oblivious with no overhead at all. We consider this question for the ANN problem. In particular, is it possible to prove a logarithmically larger lower bound for the oblivious ANN problem as opposed to the best known non-oblivious ANN lower bound? We answer in the affirmative in this work.

1.1 Our Contributions

Our main result is a stronger cell-probe lower bound for the oblivious ANN problem, which is $\Omega(\log n)$ higher than the best known cell-probe lower bound for the non-oblivious ANN problem.

**Theorem 1.1** (Informal). Let $\mathcal{D}$ be any dynamic, statistically oblivious data structure that solves $(c, r)$-ANN$_{d, \ell_1}$ over the $d$-dimensional Hamming cube, on an online sequence of $n$ insertions and queries, in the oblivious cell-probe model with word size $w$ and client storage of $m = o(n)$ bits. Then for some constant $c > 1$ and $r = \Theta(d)$, $\mathcal{D}$ must have worst case per-operation running time

$$
\Omega\left( \frac{d \cdot \log(n/m)}{\log(w \log n)^2} \right).
$$

In the natural setting of $m \leq n^{1-\rho}$ and $w = \Theta(\log n)$, the operational time is at least $\Omega(d \log n/(\log \log n)^2)$.

To the best of our knowledge, this is the first time that a lower bound of $\omega(d)$ has been successfully proved for ANN in the cell-probe model. This is also the first oblivious cell-probe lower bound exceeding $\omega(\log n)$. Previous works on oblivious cell-probe lower bounds have focused on data structures with $O(\sqrt{\log n})$ or smaller complexity for their non-oblivious counterparts (such as RAMs [LN18, PY18] as well as stacks, queues, deques, priority queues and search trees [JLN19]) and peaked at $\Omega(\log n)$. On the technical side, we remark that our work is the first to apply the technique of Larsen [Lar12a] of combining the chronogram [FS89] with cell sampling [PTW10] to prove a lower bound on privacy-preserving data structures. So far, these techniques could not be leveraged to prove higher bounds in the oblivious cell-probe model.
To complement our main result, we present a variant of the reduction by Bentley and Saxe [BS80], who showed that dynamic data structures can be built in a black-box fashion from their static counterparts, for the special class of decomposable problems (which include many natural variants of near-neighbors search, range searching and any class of linear queries). We show that any oblivious static data structure solving a decomposable problem can be transformed into an oblivious dynamic data structure with only an additional logarithmic overhead.

**Theorem 1.2** (Informal). If there exists an oblivious static data structure for a decomposable problem $P$ of $n$ items with storage of $S_{st}(n)$ cells, preprocessing of $P_{st}(n)$ cell-probes and amortized $Q_{st}(n)$ cell probes for queries, then there exists an oblivious dynamic data structure for $P$ using $S_{dy}(n) = O(S_{st}(n))$ cells of storage, preprocessing of $P_{dy}(n) = P_{st}(n)$ cell probes, amortized $Q_{dy}(n) = O(\lg n \cdot Q_{st}(n) + \lg n \cdot P_{st}(n)/n)$ cell probes for each query/update operation.

The above theorem states that the largest separation between oblivious cell-probe lower bounds for static and dynamic structures solving decomposable problems can be at most logarithmic. One can view the chronogram technique as creating a dynamic data structure lower bound by boosting a static data structure lower bound (via the cell sampling method) by an $\tilde{\Omega}(\lg n)$ factor. Therefore, the chronogram can be viewed as optimal for decomposable problems even in the oblivious model.

### 1.2 Technical Overview

The high-level approach behind the proof of Theorem 1.1 is to exploit obliviousness in a new (and subtle) way in order to compose a variation of the static cell-sampling lower bound for ANN in [PTW10] together with the chronogram method [FS89]. While this template was the technical approach of several previous dynamic data structure lower bounds for queries with “error-correcting codes” (ECC) properties (such as polynomial evaluation [Lar12a], range counting [Lar12b] and online matrix-multiplication [CGL15]), this program is doomed to fail for ANN for two fundamental reasons. The first reason is that the chronogram method requires the underlying data structure problem to have an “ECC-like” property, namely, that any local modification of the database changes the answer to (say) half of the queries (in other words, a random query is sensitive to even a single update in the data set). In contrast, ANN queries are sensitive only to updates in an exponentially-small volumed ball around the query point. This already impedes the application of the chronogram method. The second, more subtle and technically challenging problem, is the fact that in the ANN problem, only a tiny fraction ($1/poly(n)$) of queries actually reveal information about the underlying data set – these are queries which reside close to the data set and hence may report an input point (we call these “yes” queries). As explained below, this feature of ANN turns out to be a significant barrier in carrying over the static cell-sampling argument to the dynamic setting (as opposed to cell-sampling lower bounds for “k-wise independent” queries), and overcoming this problem is the heart of the paper. Surpassing this obstacle also entailed us to construct an alternative information-theoretic proof of [PTW10]’s static lower bound for the standard (non-oblivious) ANN problem, which is key for scaling it to the dynamic setting (and, as a bonus, also improves the parameters of the lower bound in [PTW10]).

In order to overcome the aforementioned two challenges, we use obliviousness in two different ways. The first one, which is more standard (in light of recent works [LN18, PY18]), overcomes the first problem, mentioned above, of insensitivity of near-neighbor queries to the chronogram construction. Recall that the chronogram method partitions a sequence of $\Theta(n)$ random update operations into $\Theta(\lg n)$ geometrically decreasing intervals (“epochs”), where the hope is to show
that a random query is *simultaneously* sensitive to (essentially) all epochs. As discussed above, \(\text{ANN}\) lacks this property, and it is not hard to see that if, for example, updates are drawn uniformly and independently at random, then any query will only be sensitive to the first \(O(1)\) epochs with overwhelming probability (due to the geometric decay of epochs, which is essential, as it reduces a dynamic lower bound to that of solving logarithmically many *independent* static problems, one per epoch). We circumvent this issue by using a simple geometric partitioning trick of the hypercube together with the (computational) indistinguishability constraint of ORAMs. This argument is key to the proof, as it reverses the quantifiers: it implies that for oblivious data structures, it is enough to show that for each epoch, there is some distribution on \(\text{ANN}\) queries that must read \(\Omega(d)\) cells from the epoch (as opposed to a *single* distribution which is sensitive to *all* epochs). Indeed, assuming this (much) weaker condition, if the data structure does not probe \(\Omega(d)\) cells from *every* epoch, an adversary (even when computationally bounded) can learn information about the query’s location (in particular, which partition the query belongs to), contradicting obliviousness.

The second way in which we exploit obliviousness is much more subtle and illuminates the difficulty in carrying out cell-sampling static lower bounds in dynamic settings for data structure problems (like \(\text{ANN}\)) where only \(o(1)\)-fraction of the queries reveal useful information. Before diving into the dynamic case, we briefly explain our modifications of the static lower bound which enables a higher lower bound in the dynamic setting. At a high level, the cell-sampling argument of [PTW10] shows that for very efficient, static data structures, there exists a small number of *memory cells* \(T\) of the data structure that are the only cells probed by many queries. These queries are referred to as *resolved queries* and denoted by \(Q(T)\). The main idea of cell sampling is to show that the queries in \(Q(T)\) reveal more bits of information about the underlying data set (denoted by \(X\)) than the number of bits that can be stored in the sampled cells \(T\), which would lead to a contradiction. However, in the \(\text{ANN}\) setting, showing that the queries in \(Q(T)\) reveal enough information about the underlying data set \(X\) is highly nontrivial – One way to prove this statement is to show that the resolved queries are essentially *independent* of the underlying data set, i.e., \(Q(T) \perp X\). If this were true, then a standard *metric expansion* argument shows that the *neighborhood* of distance \(r\) surrounding all resolved queries \(Q(T)\), covers at least half of the boolean hypercube. As a result, all points landing in the neighborhood of \(Q(T)\) will be reported by at least one query in \(Q(T)\). If the points in the data set are generated *uniformly and independently* distributed conditioned on \(Q(T)\), it can be shown that a constant fraction of data set points in \(X\) will fall into neighborhood of \(Q(T)\) except with negligible probability. Hence, a constant fraction of points in \(X\) will be recovered by using only the contents of sampled cells \(T\). Alas, for *adaptive* data structures, the resolved queries could depend heavily on the content of the cells, and this *correlates* \(Q(T)\) and the database \(X\). In the work of [PTW10], the authors handle this correlation using a careful, adaptive cell-sampling argument combined with a union-bound over all possible memory states of the data structure, which effectively breaks the dependence between resolved queries \(Q(T)\) and the data set \(X\). We present an alternative method of proving independence using information theoretic arguments. Intuitively, even though \(Q(T)\) and \(X\) are indeed correlated random variables in the general adaptive setting, we argue that this correlation cannot be too large: the set of resolved queries \(Q(T)\) are completely determined by the addresses and contents of the sampled cells \(T\), as one can determine whether \(q \in Q(T)\) by executing \(q\) and checking if \(q\) ever probes a cell outside of \(T\). Since \(T\) is a small set of cells, the data set \(X\) and the set of resolved queries \(Q(T)\) have low mutual information by a data processing inequality. We formalize this intuition by constructing an impossible “geometric packing” compression argument of the data set \(X\) using only the sampled cells \(T\). These ideas also
allow us to use one-round cell sampling [Lar12b] as opposed to multiple-round cell-sampling, which slightly improves the lower bound shown in [PTW10].

Moving back to the dynamic setting, our new arguments still break down due to the fact that memory cells may be overwritten at different points in time. The typical method for proving dynamic lower bounds [Lar12a, Lar12b] composes the cell sampling technique and chronogram method. A random update sequence \( U \) is partitioned into geometrically-decreasing sized epochs. For epoch \( i \), we denote \( C_i(U) \) as all cells that were last overwritten by updates in epoch \( i \), \( U_i \). Next, the cell sampling technique is applied to each \( C_i(U) \) to find a small subset of sampled cells \( T_i \subseteq C_i(U) \) such that for almost all queries, the only cells probed in \( C_i(U) \) appear in \( T_i \). We denote these resolved queries by \( Q_i(T_i) \). Once again, we need to show that the answers of resolved queries \( Q_i(T_i) \) reveal a lot of information about points inserted in \( U_i \). Unfortunately, our previous approach fails as it is impossible to determine \( Q_i(T_i) \) using only the sampled cells \( T_i \). Note, if a query probes a cell outside of \( T_i \), one cannot determine whether the cell belongs to \( C_i(U) \) or not. Therefore, one needs to know the addresses of cells in \( C_i(U) \), denoted by \( C_i^{addr}(U) \), to determine \( Q_i(T_i) \). Unfortunately, the number of bits needed to express \( C_i^{addr}(U) \) may be very large and contain significant information about \( U_i \). So, we can no longer argue that the set of resolved queries \( Q_i(T_i) \) is determined by a low-entropy random variable as in the static case.

This is where statistical obliviousness comes to the rescue. The main observation is that the addresses of cells last overwritten by updates in epoch \( i \), \( C_i^{addr}(U) \), cannot reveal too much information about the updates in \( U_i \) for any sufficiently statistically oblivious data structure. We prove this using a certain “reverse Pinsker inequality” which allows us to conclude that the mutual information \( I(U_i; C_i^{addr}(U)) = o(|U_i|) \) bits for any \( O(1/\lg^2 n) \)-statistically oblivious data structure. We note this inequality may be of independent interest to other oblivious lower bounds. Now, we can see that the address sequence \( C_i^{addr}(U) \), together with the small set of sampled cells \( T_i \subseteq C_i(U) \) from the \( i \)-th epoch, completely determine the resolved query set \( Q_i(T_i) \). Therefore, a data processing argument once again asserts that the large resolved query set \( Q_i(T_i) \) is almost independent of the updates \( U_i \). By a packing argument (similar to the static case), we can show that a constant fraction of the points in \( U_i \) fall into the neighborhood around the resolved queries \( Q_i(T_i) \) and each of these points will be returned by at least one resolved query. As a result, the answers of resolved queries reveal more information about \( U_i \) than the number of bits that can be stored in the sampled cells \( T_i \) providing our desired contradiction. We conclude that at least \( \tilde{\Omega}(d) \) cells must be probed from each epoch. Combined with our first application of obliviousness, we show that \( \tilde{\Omega}(d \lg n) \) cells must be probed from all epochs.

1.3 Related Work

The cell-probe model was introduced by Yao [Yao81] as the most abstract (and compelling) model for proving lower bounds on the operational time of data structures, as it is agnostic to implementation or hardware details, and hence captures any imaginable data structure. The chronogram technique of Fredman and Saks [FS89] was the first to prove \( \Omega(\lg n/\lg \lg n) \) dynamic cell-probe lower bounds. Pătraşcu and Demaine [PD06] later introduced the information transfer technique which was able to prove \( \Omega(\lg n) \) lower bounds. Larsen [Lar12a] was able to combine the chronogram with the cell-sampling technique of static data structures [PTW10] to prove an \( \Omega((\lg n/\lg \lg n)^2) \) for range searching problems, which remains the highest cell-probe lower bound to date for any dynamic search problem. Recently, Larsen et al. [LWY18] exhibited a new technique for proving \( \tilde{\Omega}(\lg^{1.5} n) \) cell-probe lower bounds on decision data structure problems, circumventing the need for
Oblivious cell-probe lower bounds. The seminal work of Larsen and Nielsen [LN18] presented the first cell-probe lower bound for oblivious data structures, in which they proved a (tight) $\Omega(\lg n)$ lower bound for ORAMs. Jacob et al. [JLN19] show $\Omega(\lg n)$ cell-probe lower bounds for oblivious stacks, queues, deques, priority queues and search trees. Both [LN18, JLN19] adapt the information transfer technique of Pătraşcu and Demaine [PD06]. Persiano and Yeo [PY18] show an $\Omega(\lg n)$ lower bound for differentially private RAMs which have weaker security notions than ORAMs using the chronogram technique originally introduced by Fredman and Saks [FS89] with modifications by Pătraşcu [Pat08]. Another line of work has investigated the hardness of lower bounds for other variants of ORAMs. Boyle and Naor [BN16] show that lower bounds for offline ORAMs (where all operations are given in batch before execution) imply lower bounds for sorting circuits. Weiss and Wichs [WW18] show that lower bounds for online, read-only ORAMs imply lower bounds for either sorting circuits and/or locally decodable codes.

Near-neighbor lower bounds. There have been many previous works on lower bounds for non-oblivious near(est)-neighbors problems. The following series of lower bound results considered deterministic algorithms in polynomial space [BOR99, BR02, CCGL03, Liu04]. Chakrabarti and Regev [CR04] present tight lower bounds for the approximate-nearest-neighbor problem for possibly randomized algorithms that use polynomial space. Several later works consider various lower bounds for near(est)-neighbors with different space requirements, the ability to use randomness and different metric spaces [CR04, PT06, AIP06, ACP08, PTW08]. As mentioned before, the highest cell-probe lower bound for dynamic ANN is the static $\Omega(d/\lg(sw/dn))$ lower bound of Wang and Yin [WY14]. In fact, all the above works prove lower bounds on static near-neighbor search where the data set is fixed and no points may be added.

2 Preliminaries

We present a formal definition of the oblivious cell-probe model as well as the ANN problem.

2.1 Oblivious Cell Probe Model

We will prove our lower bounds in the oblivious cell-probe model which was introduced by Larsen and Nielsen [LN18] and is an extension of the original cell-probe introduced by Yao [Yao81]. The oblivious cell-probe model consists of two parties: the client and the server. The client outsources the storage of data to the adversarial server which is considered to be honest-but-curious (also referred to as semi-honest). In addition, the client wishes to perform some set of operations over the outsourced data in an oblivious manner. Obliviousness refers to the the client’s wishes to hide the operations performed on the data from the adversarial server that views the sequence of cells probed in the server’s memory. Note the adversary’s view does not contain the contents of server memory as a way to separate the security of accessing data and securing the contents of data. We now describe the oblivious cell-probe model in detail.

In the oblivious cell-probe model, the server’s memory consists of cells with $w$ bits. Each cell is given a unique address from the set of integers $[K]$. It is assumed that all cell addresses can fit into a single word which means that $w \geq \lceil \lg_2 K \rceil$. The client’s memory consists of $m$ bits. Additionally,
there exists an arbitrarily long, finite length binary string $R$ which contains all the randomness that will be used by the data structure. For cryptographic purposes, $R$ may also be used as a random oracle. The binary string $R$ is chosen uniformly at random before the data structure starts processing any operations. As a result, $R$ is independent of any operations of the data structure.

A data structure in the oblivious cell-probe model performs operations that only involve either a cell probe to server memory or accessing bits on client memory. During a cell probe in server memory, the data structure is able to read or overwrite the contents of the probed cell. The cost of any operation is measured by the number of cells that are probed on the server’s memory. The accesses to bits in client memory are considered free for the data structure. Any access to bits in the random string $R$ are also free. We denote the expected query cost to be the maximum over all sequences of operations $O$ and query $q$ of the expected number of cell probes performed when answering query $q$ over the random string $R$ after processing the all operations in $O$. We denote the worst case update cost as the maximum over all sequences of operations $O$, update $u$ and random string $R$ of the number of cells probed when processing update $u$ after processing all operations in $O$.

We now move onto the privacy requirements of data structures in the oblivious cell-probe model. The random variable $V_D(O)$ as the adversary’s view of the data structure $D$ processing a sequence of operations where randomness is over the choice of the random string $R$. The adversary’s view, $V_D(O)$, will contain the addresses of cells that are probed by $D$ when processing $O$. Finally, we assume that $D$ must process a sequence of operations in an online manner. That is, $D$ must finish executing one operation before receiving the next operation. Furthermore, the adversary is aware when execution of one operation finishes and the execution of another operation begins. As a result, for any sequence $O = (op_1, \ldots, op_n)$, we can decompose the adversary’s view as $V_D(O) = (V_D(op_1), \ldots, V_D(op_n))$. Unlike the previous model, we will assume statistical security instead of computational security. We now present a formal definition of the security of an oblivious cell-probe data structure.

**Definition 2.1.** A cell-probe data structure $D$ is $\delta$-statistically oblivious if for any two equal length sequences $O_1$ and $O_2$ consisting of valid operations, then the statistical distances of $V_D(O_1)$ and $V_D(O_2)$ satisfy

$$|V_D(O_1) - V_D(O_2)| \leq \delta.$$ 

Throughout the rest of our work, we will consider $\delta$-statistical obliviousness with $\delta \leq 1/\lg^2 n$. Note that the above definition is a much weaker definition than previous definitions of statistical obliviousness in cryptography as the distinguishing probability need be at most $1/\lg^2 n$ as opposed to being a negligible function of $n$. However, as we are proving a lower bound, a weaker notion of obliviousness results in strong lower bounds.

We now briefly describe the implications of cell-probe lower bounds in the client-server setting. The majority of previous ORAM works considered the server to be passive storage, which means that the server does not perform any computation beyond retrieving and overwriting the contents of cell at the request of the client. In this case, a cell-probe lower bound implies a bandwidth lower bound in the client-server setting for any oblivious data structure. On the other hand, if we consider the case when the server can perform arbitrary computation, any cell-probe lower bound implies a lower bound on server computation.
2.2 Approximate-Near-Neighbor (ANN) Problem

We now formally define the \((c, r)\)-approximate-near-neighbor problem over the \(d\)-dimensional boolean hypercube using the \(\ell_1\) distance as the measure. In our work, we focus on the online version which allows insertion of points into the dataset. Let \(U := \{0, 1\}^d\) denote the set of all points in the space. If the insert operation is called with the same point \(p \in U\) twice, then the second insert operation is ignored. We now formally describe the problem.

**Definition 2.2** (Online, Dynamic \((c, r)\)-ANN\(_d,\ell_1\)). The dynamic \((c, r)\)-approximate-near-neighbor problem over the \(d\)-dimensional boolean hypercube endowed with the \(\ell_1\) distance measure asks to design a data structure that maintains an online dataset \(S \subseteq U\) under an online sequence of \(n\) operations of the following two types:

1. **insert**\((p)\), \(p \in U\): Insert the point \(p\) if it does not already exist in \(S\);
2. **query**\((q)\), \(q \in U\): If there exists a unique \(p \in S\) such that \(\ell_1(p, q) \leq r\), then report any any \(p' \in S\) such that \(\ell_1(p', q) \leq cr\). If all points \(p \in S\) are such that \(\ell_1(p, q) > r\), then the output should be \(\bot\).

2.3 Decomposable Problems

We now define decomposable problems. From a high level, a problem is decomposable if the problem may be solved on partitions of any data set and the results can be combined to give the result over the entire data set. Many natural problems are decomposable such as many variants of near-neighbors search, range counting and interval stabbing.

**Definition 2.3.** A problem \(\mathcal{P}\) is decomposable if for any two disjoint data sets \(D_1\) and \(D_2\) and any query \(q\), there exists a function \(f\) that can be computed in \(O(1)\) time such that

\[
\mathcal{P}(q, D_1 \cup D_2) = f(\mathcal{P}(q, D_1), \mathcal{P}(q, D_2)).
\]

3 Oblivious, Dynamic Lower Bound

In this section, we prove a logarithmically larger lower bound for the dynamic variant of the ANN problem compared to the previous, highest lower bound for non-oblivious ANN by Wang and Yin [WY14]. We consider the \((c, r)\)-ANN\(_{d,\ell_1}\) problem over a \(d\)-dimensional boolean hypercube with respect to the \(\ell_1\) norm where \(d = \Omega(\log n)\) and \(\Theta(n)\) will be number of points inserted into the data set. We denote \(t_u\) as the worst case time for any insert operation and \(t_q\) as the expected time for any query operation. For our oblivious lower bound, we consider the two party scenario where a client stores \(m\) bits that are free to access while the server holds the cells consisting of the data structure’s storage. We prove the following lower bound:

**Theorem 3.1.** Let \(\mathcal{D}\) be an randomized, dynamic, oblivious cell-probe data structure for \((c, r)\)-ANN\(_{d,\ell_1}\) over a \(d\)-dimensional boolean hypercube where \(d = \Omega(\log n)\) under the \(\ell_1\) norm. Let \(w\) denote the cell size in bits, \(S\) denote the number of cells stored by the server for the data structure and \(m\) denote the client storage in bits. If \(m = o(n)\), then there exists parameters of constant \(c \geq 1\) and \(r = \Theta(d)\) and a sequence of \(\Theta(n)\) operations such that

\[
t_q = \Omega\left(\frac{d \cdot \log(n/m)}{(\log(t_u w))^2}\right).
\]
To prove Theorem 3.1, we proceed with a “geometric variation” of the chronogram argument in [Lar12a] where our operation sequence consists of \( \Theta(n) \) independent but not identically drawn random insert operations, which we will describe later. This random sequence of updates is followed by a single query operation. The insert operations are partitioned into epochs whose sizes decrease exponentially by a parameter \( \beta \geq 2 \) which will be defined later. All epochs will contain at least \( \max\{\sqrt{n}, m^2\} \) insert operations. Each epoch will be indexed by an non-negative integer that increases in reverse chronological time. Epoch 0 will consist of the last \( \max\{\sqrt{n}, m^2\} \) insert operations before the query is performed, epoch 1 will consist of the last \( \beta \cdot \max\{\sqrt{n}, m^2\} \) insert operations before epoch 1 and so forth. Therefore, there will be \( k := \Theta(\lg(n/m)) \) epochs. For all epochs \( i \) where \( 0 \leq i < k \) will consist of exactly \( n_i := \beta^i \cdot \max\{\sqrt{n}, m^2\} \) insert operations.

For notation, the sequence of \( n \) insert operations are denoted by the random variable \( U \). We denote the sequence of insert operations in any epoch indexed by \( i \) using the random variable \( U_i \). Therefore, we can write \( U = (U_{k−1}, \ldots, U_0) \).

**Hard distribution.** Given the partitioning of the \( \Theta(n) \) insert operations into geometrically decaying epochs, we now define the hard distribution for our lower bound. In order to (later) exploit the obliviousness of the data structure, our hard distribution shall have a “direct sum” structure, which is simple to design using the geometry of the ANN problem. Conceptually, the hard distribution will split the \( d \)-dimensional boolean hypercube into disjoint subcubes where each subcube is uniquely assigned to one of the epochs. To this end, every epoch \( i \in \{0, \ldots, k−1\} \) will be assigned a \( d' \)-dimensional boolean subcube where \( d' := \Theta(d) \) will be determined later. Each of the insert operations of any epoch \( i \) will be generated independently by picking a point from epoch \( i \)'s \( d' \)-dimensional boolean hypercube uniformly at random.

We now show how we split up the original \( d \)-dimensional boolean hypercube into \( k \) \( d' \)-dimensional boolean subcubes that are disjoint. We choose the parameter \( d > d' \) where \( d' \) will be specified later. We assign each of the \( k \) epochs a unique prefix of \( d − d' \) bits denoted by \( p_0, \ldots, p_{k−1} \in \{0, 1\}^{d−d'} \) where \( p_i \) is the prefix for epoch \( i \). We will pick the prefixes in such a way that for any \( i \neq j \in [k] \), \( \ell_1(p_i, p_j) > d' \). To see that such a choice of prefixes exist, we consider the following probabilistic method where we pick the \( k \) prefixes of \( d − d' \) bits uniformly at random. For any two \( i \neq j \in [k] \) and sufficiently large \( d = \Omega(\lg(n)) \), we know that

\[
\Pr[\ell_1(p_i, p_j) < 0.49(d − d')] \leq 1/n^3.
\]

By a Union bound over all \( n^2 \) possible pairs, we get that there must exist some choice of \( k \) prefixes such that pairwise prefixes have \( \ell_1 \) distance at least \( d' \) as long as \( d \geq 4d' \). The \( d' \)-dimensional subcube for epoch \( i \) is constructed as all points in the original \( d \)-dimensional subcube restricted to the case that the \( d − d' \) coordinates match the prefix \( p_i \). We note that our choice of subcubes has the important property that two points from different subcubes will be distance at least \( d' \) from each other as their prefixes already have \( \ell_1 \) distance of at least \( d' \).

Before continuing, we describe why this choice of hard distribution is compatible with oblivious data structures. Intuitively, our choice of hard distribution is very revealing for the choice of update points. An adversary is aware that updates from epoch \( i \) will be completely contained in the subcube assigned to epoch \( i \). Furthermore, as all subcubes are pairwise disjoint, two update points from different epochs cannot be from the same subcube. We will exploit this fact in combination with the oblivious guarantees to prove lower bounds on the operational cost of the final query. If, on average, a query point does not probe many cells that were last overwritten in some epoch \( i \), then the
adversary can simply rule out that the query point was chosen from the disjoint subcube assigned to epoch \( i \). This knowledge learned by the adversary can be used to contradict the obliviousness property. As a result, we can show that an oblivious data structure must query many cells last written from all epochs to hide the identity of the query point even if the query needs no information from some epochs.

Formally, we define the distribution of updates in epoch \( i \in \{0, \ldots, k-1\} \), \( U_i \), as the product of \( n_i \) identical distributions, \( \mu_i \). The distribution \( \mu_i \) deterministically appends the prefix \( p_i \) uniquely assigned to epoch \( i \) and picks the remaining \( d' \) coordinates uniformly at random. We denote this \( d' \)-dimensional subcube using \( P_i \). The entire distribution of updates over all epochs, \( U \), can be viewed as the product of distribution \( U = U_{k-1} \times \ldots \times U_0 \). Our hard query distribution \( q \) will simply be to query any fixed point that lies outside each of the subcubes \( P_0, \ldots, P_{k-1} \).

We show that the probability that any two points inserted during \( U_i \) are too close is low.

**Lemma 3.2.** Let \( U_i \) be the set of update points inserted in epoch \( i \) according to the hard distribution. For sufficiently large \( d' = \Omega(\log n) \), there cannot exist any query \( q \) such that \( \ell_1(u, q) \leq 0.24d' \) and \( \ell_1(v, q) \leq 0.24d' \) for any two different points \( u \) and \( v \) chosen by \( U_i \) except with probability at most \( 1/n \).

**Proof.** Note both \( u \) and \( v \) are chosen uniformly at random from a \( d' \)-dimensional boolean hypercube. As a result, we know that \( E[\ell_1(u, v)] = 0.5d' \). We apply Chernoff Bounds over the coordinates of \( u \) and \( v \) to get that \( \Pr[\ell_1(u, v) \geq 0.49d'] \leq 1/n^3 \) for sufficiently large \( d' = \Theta(d) = \Omega(\log n) \). Next, we apply a Union Bound over all \((\frac{n}{2}) \leq n^2\) pairs of points in \( X \). As a result, the probability of the existence of two points \( u \) and \( v \) whose distance is at most \( 0.49d' \) is at most \( 1/n \).

Suppose there exists a query \( q \) such that \( \ell_1(u, q) \leq 0.24d' \) and \( \ell_1(v, q) \leq 0.24d' \). By the triangle inequality, we know that \( \ell_1(u, v) \leq 0.48d' < 0.49d' \). This only occurs with probability at most \( 1/n \). \( \square \)

Additionally, we also want that queries cover large portions of the boolean hypercube cube such that they must report a point if it lands in these large subspaces of the boolean hypercube. We quantify this by considering the *neighborhood* of subsets of queries over the boolean hypercube. For any query \( q \), we consider its neighborhood to be all points in the boolean hypercube that are distance at most \( r \) from \( q \). For subsets of queries within an epoch’s assigned subcube denoted by \( Q \subseteq \{0, 1\}^{d'} \), we consider the neighborhood of \( Q \) to be any points within distance \( r \) of any query \( q \in Q \). We denote the neighborhood of \( Q \) by \( \Gamma_r(Q) \). We will use the following standard isoperimetric inequality describing the size of neighborhoods over any \( d' \)-dimensional boolean hypercube which follows directly from Harper’s theorem [FF81].

**Lemma 3.3.** Let \( H \) be all the vertices of a \( d \)-dimensional boolean hypercube. Let \( V \) be a subset of vertices in \( H \) such that \( |V| \leq 1/(2a^{2d}) \cdot |H| \) and let \( \Gamma_{ed}(V) \) be the set of all vertices that are distance at most \( ed \) from any of the vertices in \( V \). Then, there exists some constant \( a > 1 \) such that

\[
|\Gamma_{ed}(V)| \geq a^{2d} \cdot |V|.
\]

For convenience, we denote \( \Phi := \Phi(r) := a^{2d} \) as the expansion over each of the \( d' \)-dimensional boolean hypercubes for distances of \( r := \epsilon \cdot d' \) where \( 0 < \epsilon < 1 \) is a constant. The above lemmata will end up being important later when we prove our lower bounds.
Choosing parameters. We now choose the parameters for our problem. First, we want to ensure that if a query $q$ may report any point, that point will be unique with high probability. We can ensure this property by picking $c r \leq 0.24 d'$ and applying Lemma 3.2. To ensure large expansion within each epoch’s subcube, we will set $r = \Theta(d')$. As an example, we can choose parameters such as $r = 0.01 d'$ and $1 \leq c \leq 24$ to get our desired properties.

3.1 Overview of Our Proof

Before we begin formally proving our lower bound, we present a high level overview showing the steps of our approach. Our techniques will follow the techniques first outlined by Larsen [Lar12a], which combine the chronogram introduced by Fredman and Saks [FS89] and the cell sampling method introduced by Panigrahy et al. [PTW10]. We fix $t_u$ to be the worst case update time and our goal is to prove a lower bound on the expected query time $t_q$.

For the sequence of $\Theta(n)$ randomly chosen insert operations $U$, we denote $C(U)$ as the random variable of the set of all the cells stored by the data structure after processing all insert operations of $U$. We partition the cells of $C(U)$ into $k$ groups depending on the most recent operation that updated the contents of the cell. In particular, we denote $C_i(U)$ as the random variable describing the set of cells in $C(U)$ whose contents were last updated by an insert operation performed during epoch $i$. For any query point $q \in Q$, we denote $t_i(U, q)$ as the random variable denoting the number of cells that are probed by the query algorithm on input $q$ that belong to the set $C_i(U)$. For any set of queries $Q' \subseteq Q$, we denote the random variable $t_i(U, q)$ as the total number of cells probed from $C_i(U)$ when processing a query operation where the input $q$ is chosen uniformly at random from $Q'$.

The first step of our proof will be to focus on individual epochs.

Lemma 3.4. Fix the random string $R$. If $\beta = (w t_u)^2$, then for all epochs $i \in \{0, \ldots, k - 1\}$,

$$\Pr \left[ t_i(U, q_i) = \Omega \left( \frac{d'}{\lg(t_u w)} \right) \right] \geq 1/2$$

where $q_i$ is chosen uniformly at random from $P_i$.

The proof of this lemma will use the cell sampling technique introduced by Panigrahy et al. [PTW10] for static (non-oblivious) ANN lower bounds. Their main idea is to, first, assume the existence of an extremely efficient static data structure that probes a small number of cells in expectation. Next, they show the existence of a small subset of cells that resolve a very large subset of possible queries where a query is resolved by a subset of cells if the query does not probe any cells outside of the subset. Afterwards, they show that the answers of the resolved queries reveal more bits of information about the input than the maximal amount of information that can be stored about input in the subset of sampled cells. This results in a contradiction showing there cannot exist such an efficient static data structure that was original assumed.

However, to show that a lot of information is revealed by resolved queries, the work of [PTW10] used several complex combinatorial techniques. These complex techniques end up being hard to scale for the dynamic setting. To prove our dynamic lower bound, we first present new ideas that simplify the static (non-oblivious) ANN proof using information theoretic arguments. At a high level, we show that the set of resolved queries are a deterministic function of the sampled cells which contain very little information about the inputs. This suffices to prove that the resolved query set
and inputs are almost independent. Since the input points are chosen uniformly at random, it turns out that resolved queries will return a large number of input points with constant probability, which would allow us to forego the complex techniques that appear in [PTW10].

Unfortunately, it turns out significantly larger problems appear when moving to the dynamic setting even when using our simplifications. Towards a contradiction, assume that there exists an efficient data structure that probes \( o(d'/\lg(t_uw)) \) cells from the set \( C_i(U) \) in expectation. We can apply the cell sampling technique to find a small subset \( T_i \subset C_i(U) \) that resolves a large number of queries. In this case, a query \( q \) is resolved by \( T_i \) if all cells that are probed by \( q \) in the set \( C_i(U) \) all belong to \( T_i \). Note, we do not project any restrictions on the cells probed by \( q \) outside the set \( C_i(U) \). Once again, denote the set of queries resolved by \( T_i \) using \( Q_i(T_i) \). Using our information theoretic ideas, we want to show that \( Q_i(T_i) \) cannot be computed using only the little information stored in the set of sampled cells \( T_i \) and the client storage \( M(U) \) as well as the random string \( R \). In the dynamic case, the set of queries \( Q_i(T_i) \) cannot be computed using only \( T_i, M(U) \) and \( R \). As an example, consider a query \( q \in P_i \). During the execution of \( q \), consider the first time a probe is performed outside the set \( T_i \). There is no way to determine whether the probed cell exists in \( C_i(U) \) or not using only the information in \( T_i, M(U) \) and \( R \). As a result, it is impossible to accurately compute the set of resolved queries \( Q_i(T_i) \).

To get around this, we can attempt to also use \( C_i(U) \) to compute \( Q_i(T_i) \). However, the set \( C_i(U) \) is very large and may potentially contain significantly more information about \( U_i \) compared to the set of sampled cells \( T_i \) and client storage \( M(U) \). As a result, we would not be able to prove our contradiction. Instead, it turns out that computing \( Q_i(T_i) \) only requires knowledge of the addresses of \( C_i(U) \). By the guarantees of statistical obliviousness, we know that the addresses of \( C_i(U) \) may not reveal too much information about the underlying update operations \( U \). As a result, we can show that even though \( C_i(U) \) is expressed using many bits, that most of the bits cannot contain information about \( U_i \).

One more issue that arises is that the above lemma is similar yet crucially different than those used in lower bounds for non-oblivious data structures. In the standard application of the chronogram technique, the analogue of this lemma typically asserts that a single random query \( q_i \) must be simultaneously “sensitive” to most epochs. That is, \( q_i \) forces a large number of probes from cells in \( C_i(U) \) for many (essentially all) epochs \( i \) simultaneously. Instead, our lemma says that for all epochs, there exists a special query distribution \( q_i \) drawn uniformly at random from \( P_i \) built specially for that epoch \( i \) that forces many probes to cells in \( C_i(U) \). It turns out that this weaker lemma suffices for oblivious data structures. We are able to use the fact that obliviousness must hide the input query point from any adversary. The main idea is that the adversary knows there exists some query point from the set \( P_i \) that must probe \( \Omega(d'/\lg(t_uw)) \) cells from \( C_i(U) \) to correctly answer the query. If the adversary views a query that probes significantly less cells from \( C_i(U) \), it can effectively deduce that the query does not come from the query set \( P_i \) otherwise the answer of the query could not be correct. This observation by the adversary would contradict obliviousness. As a result, we can essentially boost Lemma 3.4 into the stronger variant below.

**Lemma 3.5.** If \( \beta = (wt_u)^2 \), there exists a fixed query \( q \) such that

\[
E[t_i(U,q)] = \Omega \left( \frac{d'}{\lg(t_uw)} \right)
\]

for all epochs \( i \in \{0, \ldots, k-1\} \).
The above lemma resembles the form of lemmata typically used in non-oblivious data structure lower bounds. We now show Lemma 3.5 suffices to complete the lower bound by proving Theorem 3.1.

Proof of Theorem 3.1. Note that sets of cells $C_0(U), \ldots, C_{k-1}(U)$ are all disjoint and the random variable $t_i(U, q)$ only counts the number of cells that are probed from $C_i(U)$. Therefore, the total number of cells probed by $t(U, q) = t_0(U, q) + \ldots + t_{k-1}(U, q)$. There are $k = \Theta(lg_\beta(n/m))$ epochs. Using linearity of expectation and Lemma 3.5, it can be shown that

$$E[t(U, q)] = \Omega\left(\frac{d' \lg(n/m)}{\lg(t_u w)}\right)^2.$$

The proof is completed by noting that $d = \Theta(d')$.

3.2 Bounding Cell Probes to Individual Epochs

Towards a contradiction, assume an extremely efficient data structure with $t_i(U, q_i) = o\left(d'/\lg(t_u w)\right)$ where $q_i$ is drawn uniformly at random from $P_i$. We apply the cell sampling technique such that a small subset of cells $T_i \subseteq C_i(U)$ resolves a large number of queries $Q(T_i) \subseteq P_i$.

3.2.1 Cell Sampling

**Lemma 3.6.** Fix the random string $R$. Suppose that $t_i(U, q_i) = o\left(\frac{\lg \Phi}{\lg(t_u w)}\right) = o\left(\frac{d'}{\lg(t_u w)}\right)$ where $q_i$ is drawn uniformly at random from $P_i$. Then, there exists a subset of cells $T_i \subseteq C_i(U)$ with the following properties:

- $|T_i| = \frac{n}{100w}$;
- Let $Q_i(T_i)$ be all queries resolved by $T_i$ and probe at most $2t_i(U, q_i)$ cells in $C_i(U)$. Recall a query $q \in Q_i(T_i)$ is resolved by $T_i$ if every cell in $C_i(U)$ that is probed when executing $q$ must exist in the subset $T_i$. Then, $|Q_i(T_i)| \geq 2^{d' - 1}/\Phi$.

**Proof.** For convenience, denote $t_i = t_i(U, q_i) = o\left(\frac{\lg \Phi}{\lg(t_u w)}\right) = o\left(\frac{d'}{\lg(t_u w)}\right)$. Since we fixed the random string $R$, the randomness of the data structure is strictly over the choice of updates from the hard distribution $U$ and the random query $q_i$. By Markov’s inequality, there exists a subset of queries $Q_i \subseteq P_i$ such that each $q \in Q_i$ probes at most $2t_i$ cells in $C_i(U)$ and $Q_i$ contains at least $|P_i|/2 = 2^{d' - 1}$ queries.

Consider the following random experiment where a subset $T_i \subseteq C_i(U)$ is chosen uniformly at random from all subsets with exactly $n/(100w)$ cells. Pick any query $q \in Q_i$ probing at most $2t_i$ cells in $C_i(U)$. We will analyze the probability that $q$ is resolved by $T_i$ over the random choice of


\[ T_i, \]

\[
\frac{|C_i(\mathbf{U})| - 2t_i}{n_i/(100w) - 2t_i} \geq \frac{n_i/(100w) \cdot (n_i/(100w) - 1) \cdots (n_i/(100w) - 2t_i + 1)}{|C_i(\mathbf{U})| \cdot (|C_i(\mathbf{U})| - 1) \cdots (|C_i(\mathbf{U})| - 2t_i + 1)} \\
\geq \frac{n_i/(100w) - 2t_i}{|C_i(\mathbf{U})|}^{2t_i} \\
\geq \frac{n_i}{200|C_i(\mathbf{U})|w}^{2t_i} \\
\geq \left(\frac{1}{200t_u^2}\right)^{2t_i} \\
\geq \Phi^{-1}. \]

The second last inequality uses the fact that \(|C_i(\mathbf{U})| \leq n_i t_u\) while the last inequality uses the fact that \(t_i = o(\log \Phi / \log(t_u w))\). By linearity of expectation, we know that

\[ \mathbb{E}[|Q_i(T_i)|] \geq |Q_i| \cdot \Phi^{-1} = 2^{d'_{i}}/\Phi. \]

As a result, there exists a subset \(T_i \subset C_i(\mathbf{U})\) satisfying all the required properties. \(\square\)

### 3.2.2 Information from Resolved Queries

Next, we will show that the resolved queries \(Q_i(T_i)\) will report a large number of points that are inserted by \(\mathbf{U}_i\). Recall that a point in \(\mathbf{U}_i\) is reported by a query in \(Q_i(T_i)\) if and only if it belongs to the neighborhood of \(Q_i(T_i)\) denoted by \(\Gamma_r(Q_i(T_i)) \subseteq P_i\). Note, we only consider expansion within the subcube \(P_i\). For convenience, we fix \(\mathbf{U}_{-i}\), which consists of all updates outside of epoch \(i\).

Towards a contradiction, we will suppose that most points inserted by \(\mathbf{U}_i\) land outside of \(\Gamma_r(Q_i(T_i))\) and present an impossible compression of \(\mathbf{U}_i\). Formally, we construct a one-way encoding protocol from an encoder (Alice) to a decoder (Bob). Alice receives as input \(\mathbf{U}\) and the random string \(\mathbf{R}\). Bob will receive the addresses of cells in \(C_i(\mathbf{U})\) denoted by \(C_i^{addr}(\mathbf{U})\) and the random string \(\mathbf{R}\). The goal of Alice is to encode the \(n_i\) points inserted in \(\mathbf{U}_i\). By Shannon’s source coding theorem, the expected length of Alice’s encoding must be at least \(H(\mathbf{U}_i \mid C_i^{addr}(\mathbf{U}), \mathbf{R})\), which we now analyze. In particular, we present an argument that the entropy of \(\mathbf{U}_i\) remains high even conditioned on \(C_i^{addr}(\mathbf{U})\) due to statistical obliviousness guarantees. However, statistical obliviousness provides guarantees using statistical distance which is not directly compatible with our information theoretic arguments. To do this, we present the following lemma upper bounds the contributions of the positive terms to the Kullback-Leibler divergence between two distributions, in terms of their statistical distance. We note a similar lemma previously appeared in [BRWY13].

**Lemma 3.7** (Reverse Pinsker). Let \(p(a, b)\) and \(q(a, b)\) be two distributions over \(A \times B\) in the same probability space, and let \(S = \{(a, b): \lg \frac{p(a, b)}{q(a, b)} > 1\}\). Then, \(p(S) < 2|p(a, b) - q(a, b)|\).

**Proof.** Let \(\epsilon = |p(a, b) - q(a, b)| := 2 \max_T \{p(T) - q(T)\} \geq 2(p(S) - q(S))\). Rearranging sides, we have:

\[ 15 \]
We can rewrite the following distributions:
\[ p(S) \leq \epsilon/2 + q(S) \]
\[ < \epsilon/2 + (1/2) \sum_{(a,b) \in S} q(b) \cdot p(a|b) \]
\[ \leq \epsilon/2 + (1/2) \sum_{(a,b) \in S} p(b) \cdot p(a|b) + (1/2) \sum_{(a,b) \in S} |q(b) - p(b)| \cdot p(a|b) \]
\[ \leq \epsilon/2 + p(S)/2 + (1/2) \sum_{b} |q(b) - p(b)| \]
\[ \leq \epsilon + p(S)/2 \]
where the second inequality follows from the fact for any \((a, b) \in S\), \(q(a|b) \geq p(a|b)/2\) by the choice of \(S\).

\[ \square \]

This lemma directly implies that \(D_{KL}(p(a, b) || q(a, b)) \leq 2|p - q|_{1} \cdot \max_{a,b} \log(p(a|b)/(q(a|b))) + 1\), since the total contribution of terms outside \(S\) is at most \(\sum_{(a,b)} p(a|b) \leq 1\). Using the above, we show that the entropy of \(U_{i}\) conditioned on Bob’s input remains large.

**Lemma 3.8.** Consider any \(U\) where all of \(U_{1}, \ldots, U_{i-1}, U_{i+1}, \ldots, U_{k-1}\) are fixed. That is, all update operations outside of epoch \(i\) are fixed, and denote this fixed value by \(U_{-i} = u_{-i}\). Then,

\[ H(U_{i} | C_{i}^{addr}(U), R, u_{-i}) = n_{i} \cdot (d' - o(1)) \cdot (1 - O) \]

**Proof.** We analyze the mutual information between \(U_{i}\) and \(V_{D}(U)\). Denote by \(P\) and \(Q\) the following distributions: \(P \sim (U_{i} | V_{D}(U), u_{-i})\) and \(Q \sim (U_{i} | u_{-i})\). By definition,

\[ I(P; Q) = E_{v \sim V_{D}(U)} [D_{KL} (P(U_{i} | v, u_{-i}) \parallel Q(U_{i} | u_{-i}))] \]
\[ \leq 2 \cdot E_{v} \left[ \parallel P - Q \parallel_{1} \right] \cdot \max_{u_{i,k_{-i}}, v} \log \left( \frac{P(u_{i} | u_{-i}, v)}{Q(u_{i})} \right) + 1 \]
\[ = O \left( \frac{n_{i} \cdot d'}{\lg^{2} n} \right) \]

where the first inequality is by Lemma 3.7, and the second is by the statistical-indistinguishability premise that \(\parallel P - Q \parallel_{1} \leq 1/\lg^{2} n\), and the fact that \(U_{i}\) picks points uniformly at random and independent of \(U_{-i}\). Hence the ratio between \(P\) and \(Q\) never exceeds \(2^{n_{i}d'}\).

Now, recall that \(R\) is independent of \(U_{i}\) and that \(U_{i}\) is generated independent of \(U_{-i}\). Therefore,

\[ H(U_{i} | C_{i}^{addr}(U), R, u_{-i}) = H(U_{i} | C_{i}^{addr}(U)) \]

We can rewrite

\[ H(U_{i} | C_{i}^{addr}(U)) = H(U_{i}) - I(U_{i}; C_{i}^{addr}(U)) \geq (n_{i} \cdot d') \left( 1 - O \left( \frac{1}{\lg^{2} n} \right) \right) \geq n_{i} \cdot (d' - o(1)). \]

The second inequality uses the fact that \(C_{i}^{addr}(U)\) appears in \(V_{D}(U)\). So, \(I(P; Q) = I(U_{i}; V_{D}(U)) \geq I(U_{i}; C_{i}^{addr}(U))\). The last inequality uses the fact that \(d' = \Omega(\lg n)\). \(\square\)
Going back to the original encoding protocol, we know that Alice’s expected encoding size must be at least $H(U_i | C_i^{addr}(U), R) = n_i \cdot (d' - o(1))$. We will utilize the fact that most points inserted by $U_i$ land outside of $\Gamma_r(Q_i(T_i))$ to present an impossible encoding scheme.

**Lemma 3.9.** Fix $U_{-i}$, that is all update operations outside of epoch $i$. With probability at least $1/2$ over the choice of $U_i$, at least $n_i/8$ points in $U_i$ exist in the neighborhood of the set of resolved queries, $\Gamma_r(Q_i(T_i))$. That is,

$$\Pr[|\Gamma_r(Q_i(T_i)) \cap U_i| \geq n_i/8] \geq 1/2.$$  

**Proof.** Towards a contradiction, suppose that the number of points in $U_i$ that land in $\Gamma_r(Q_i(T_i))$ is at least $n_i/8$ with probability at most $1/2$. Let $u_{-i}$ be the realization of $U_{-i}$. We construct an impossible one-way communication protocol for encoding $U_i$ which will contradict Shannon’s source coding theorem.

**Alice’s Encoding.** Alice receives as input $U_i$, $u_{-i}$ and $R$.

1. Using $u_{-i}$, $U_i$ and $R$, execute all operations to compute the cell sets $C_{k-1}(U), \ldots, C_0(U)$. Afterwards, Alice finds the supposed $T_i$ of Lemma 3.6. To do this, Alice can iterate through all subsets of $C_i(U)$ containing exactly $n_i/(100w)$ cells. Alice can also compute query sets $Q_i(T_i)$ and $\Gamma_r(Q_i(T_i))$. Finally, Alice computes $F$ denoting the number of points of $U_i$ in $\Gamma_r(Q_i(T_i))$.

2. If there are more than $n_i/8$ points of $U_i$ in $\Gamma_r(Q_i(T_i))$, $F \geq n_i/8$, then Alice’s encoding starts with a 0-bit. Alice encodes $U_i$ in the trivial manner using $n_i \cdot d'$ bits.

3. Otherwise, suppose that less than $n_i/8$ points of $U_i$ land in $\Gamma_r(Q_i(T_i))$. That is, $F < n_i/8$. In this case, Alice encodes the contents and addresses of $T_i$ using $2w \cdot |T_i| = n/50$ bits. Next, Alice encodes the set of cells last updated by operations after epoch $i$. That is, the addresses and contents of cells in $C_{i-1}(U), \ldots, C_0(U)$. The total number of cells in these are $n_i/\beta + n_i/\beta^2 + \ldots = \Theta(n_i/\beta)$ as $\beta \geq 2$. Alice also encodes the client storage after executing all updates, $M(u_{-i}, U_i)$ using $m = o(n)$ bits. Alice encodes $F$ using $\log n_i$ bits and the indices of $U_i$ whose points land in $\Gamma_r(Q_i(T_i))$ using $\log \binom{n}{P}$ bits. Each of these $F$ points are encoded trivially using $d'$ bits each. The remaining $n_i - F$ points that land outside of $\Gamma_r(Q_i(T_i))$ are encoded using $\log(|P| - |\Gamma_r(Q_i(T_i))|)$ bits.

**Bob’s Decoding.** Bob receives as input $u_{-i}$, $C_i^{addr}(U)$, $R$ and Alice’s encoding.

1. If Alice’s encoding starts with a 0-bit, then Bob decodes $U_i$ using the next $n_i \cdot d'$ bits in the trivial manner.

2. Otherwise, Bob executes all updates prior to epoch $i$ using $u_{-i}$ and $R$. Bob decodes the addresses and contents of $T_i \subset C_i(U)$ as well as the addresses and contents of $C_{i-1}(U), \ldots, C_0(U)$. At this point, Bob has the contents and addresses of all cell sets $C_{k-1}(U), \ldots, C_{i+1}(U)$, $C_{i-1}(U), \ldots, C_0(U)$. Additionally, Bob has the addresses of $C_i(U)$, $C_i^{addr}(U)$, but not the contents. Using the next $m$ bits, Bob decodes the client storage $M(U)$ after executing all updates. Bob attempts to execute each possible query in $P_i$ to compute $Q_i(T_i)$. Note, Bob executes each query using $R$ and $M(U_i)$ until the query attempts to probe a cell with an
address in \( C_i^{addr}(U) \setminus T_i^{addr} \), probes more than \( 2t_q \) cells or finishes executing. As long as a query does not probe a cell in \( C_i(U) \setminus T_i \), Bob is able to accurately simulate the query. As a result, Bob accurately computes \( Q_i(T_i) \) as well as \( \Gamma_r(Q_i(T_i)) \). Next, Bob decodes \( F \) as well as the \( F \) indices of \( U_i \) of points that in \( \Gamma_r(Q_i(T_i)) \). For each of these \( F \) points, Bob uses the next \( d' \) bits to decode them in the trivial manner. For the remaining \( n_i - F \) points, Bob decodes the point using the next \( \lg(|P_i| - |\Gamma_r(Q_i(T_i))|) \).

**Analysis.** We start with the case of Alice’s encoding is prepended with a 0-bit. For this scenario, Alice’s encoding is always \( 1 + n_i \cdot d' \) bits. Alice’s encoding starts with a 0-bit only in the case that there are more than \( n_i/8 \) points of \( U_i \) that land in \( \Gamma_r(Q_i(T_i)) \) which happens with probability at most \( 1/2 \) by our assumption towards a contradiction.

When Alice’s encoding starts with a 1-bit, Alice’s encoding size in bits is at most
\[
1 + 2w(|T_i| + |C_{i-1}(U)| + \ldots + |C_0(U)|) + n_i \log n_i + \log \left( \frac{n_i}{F} \right) + Fd' + (n_i - F) \log(|P_i| - |\Gamma_r(Q_i(T_i))|).
\]

By our choice of \( \beta = (t_u w)^2 \), we know that \( |C_{i-1}(U)| + \ldots + |C_0(U)| = \Theta(n_i/\beta) \). By Lemma 3.3, we know that \( |\Gamma_r(Q_i(T_i))| \geq |Q_i(T_i)| \cdot \Phi \geq 2^{d' - 1} \) as long as \( |Q_i(T_i)|/\Phi \leq 2^{d' - 1} \). If \( |Q_i(T_i)| \) is too large, we can pick any arbitrary subset of size \( 2^{d' - 1}/\Phi \) and consider the neighborhood of the subset. As a result, we know that \( \log(|P_i| - |\Gamma_r(Q_i(T_i))|) \leq \log(2^{d'} - 2^{d' - 1}) = d' - 1 \). Also, we note that \( n_i \geq m^2 \), so \( m = o(n_i) \). Note that the encoding is maximized when \( F = n_i/8 \):
\[
n_i d' - \frac{7n_i}{8} + \frac{n_i}{50} + o(n_i) < n_i d' - \frac{n_i}{2} + o(n_i).
\]

Denote \( p \leq 1/2 \) to be the probability that Alice’s encoding starts with a 0-bit. Putting together the two cases, we get:
\[
p(1 + n_i d') + (1 - p) \left( n_i d' - \frac{n_i}{2} + o(n_i) \right) < n_i d' - \frac{n_i}{4} + o(n) < n_i (d' - o(1)) = H(U_i \mid C_i^{addr}(U), R, u_{-i})
\]
since the encoding is maximized when \( p = 1/2 \). As a result, our encoding is impossible as it contradicts Shannon’s source coding theorem.

### 3.2.3 Proof of Lemma 3.4

Lemma 3.9 shows that at least \( n_i/8 \) points of \( U_i \) will land in the set \( \Gamma_r(Q_i(T_i)) \) with high constant probability. By Lemma 3.2, all of these \( n_i/8 \) points will be reported by at least one query \( Q_i(T_i) \) with high probability. We now show that the entropy contained in these \( n_i/8 \) points is larger than the number of bits that may be stored in the contents of the cells in \( T_i \) to prove Lemma 3.4.

**Proof of Lemma 3.4.** Towards a contradiction, suppose that \( t_i(U, q_i) = o(d'/\lg(t_u w)) \). Our assumption directly implies that \( \Pr[t_i(U, q_i) = \Omega(d'/\lg(t_u w))] < 1/2 \). For convenience, fix all updates outside of epoch \( i \) as \( U_{-i} = u_{-i} \). We present an impossible one-way communication protocol between an encoder (Alice) and a decoder (Bob). Alice will attempt to encode \( U_i \) efficiently. Both Alice and Bob will receive \( R \). In addition, Bob will receive the addresses of cells in \( C_i(U) \) denoted by \( C_i^{addr}(U) \). Alice’s expected encoding size must at least \( H(U_i \mid C_i^{addr}(U), R, u_{-i}) \). By Lemma 3.8, we know that
\[
H(U_i \mid C_i^{addr}(U), R, u_{-i}) = n_i \cdot (d' - o(1)).
\]
Alice’s Encoding. As input, Alice receives $U_i$, $u_{-i}$ and $R$.

1. Using $u_{-i}$, $U_i$ and $R$, execute all operations to compute the cell sets $C_{k-1}(U), \ldots, C_0(U)$. Afterwards, Alice finds the supposed $T_i$ of Lemma 3.6. To do this, Alice can iterate through all subsets of $C_i(U)$ containing exactly $n_i(100w)$ cells. Alice can also compute query sets $Q_i(T_i)$ and $\Gamma_r(Q_i(T_i))$. Finally, Alice computes $F$ denoting the number of points of $U_i$ in $\Gamma_r(Q_i(T_i))$.

2. If there are less than $n_i/8$ points in $\Gamma_r(Q_i(T_i))$ corresponding to $F < n_i/8$ or there exists two points in $U_i$ within distance at most $0.49d'$, Alice’s encoding will start with a 0-bit. Alice will encode $U_i$ in the trivial manner using $n_i \cdot d'$ bits.

3. Otherwise, Alice’s encoding starts with a 1-bit. Alice encodes the addresses and contents of $T_i$ using $2w \cdot |T_i| = n/50$ bits. Next, Alice encodes the addresses and contents of all cells overwritten by an update operation after epoch $i$. That is, the cells in $C_{i-1}(U), \ldots, C_0(U)$. Alice also encodes the client storage after executing all update operations, $M(U)$, using $m$ bits. Using $n_i$ bits, Alice encodes whether each of the points in $U_i$ belong to $\Gamma_r(Q_i(T_i))$ or not. For all $n - F$ points outside of $\Gamma_r(Q_i(T_i))$, Alice encodes them using $d'$ bits each in the trivial manner. Afterwards, Alice executes the queries in $Q_i(T_i)$ in some fixed order (such as lexicographically increasing order). Each time a new point in $U_i$ is reported, Alice encodes the index of the point in $U_i$ using $\lg n_i$ bits completing the encoding.

Bob’s Decoding. Bob receives as input $u_{-i}$, $R$ and Alice’s encoding.

1. If Alice’s message starts with a 0-bit, then Bob decodes $U_i$ in the trivial manner using the next $n_i d'$ bits.

2. If Alice’s encoding starts with a 1-bit, Bob executes all updates prior to epoch $i$ using $u_{-i}$ and $R$. Bob decodes the addresses and contents of $T_i \subset C_i(U)$ as well as the addresses and contents of $C_{i-1}(U), \ldots, C_0(U)$. At this point, Bob has the contents and addresses of all cell sets $C_{k-1}(U), \ldots, C_{i+1}(U), C_{i-1}(U), \ldots, C_0(U)$. Additionally, Bob has the addresses of $C_i(U)$, $C_i^{addr}(U)$, but not the contents. Using the next $m$ bits, Bob decodes the client storage $M(u_{-i}, U_i)$ after executing all updates. Bob attempts to execute each possible query in $P_i$ to compute $Q_i(T_i)$. Note, Bob executes each query using $R$ and $M(U_i)$ until the query attempts to probe a cell with an address in $C_i^{addr}(U) \setminus T_i^{addr}$, probes more than $2t_q$ cells or finishes executing. As long as a query does not probe a cell in $C_i(U) \setminus T_i$, Bob is able to accurately simulate the query. As a result, Bob accurately computes $Q_i(T_i)$ as well as $\Gamma_r(Q_i(T_i))$. Using the next $n_i$ bits, Bob decodes whether each point in $U_i$ belongs to $\Gamma_r(Q_i(T_i))$. Bob decodes all $n_i - F$ points outside of $\Gamma_r(Q_i(T_i))$ using the next $(n_i - F) \cdot d'$ bits in the trivial manner. To decode the $F$ points in $\Gamma_r(Q_i(T_i))$, Bob will execute the queries in $Q_i(T_i)$ in the same fixed order as Alice. Each time a new point is reported, Bob uses the next $\lg n_i$ bits to decode the point’s index in $U_i$ completing the decoding procedure. As Alice’s encoding starts with a 1-bit only when no two points in $U_i$ are within distance $0.49d'$, we know that all points in $\Gamma_r(Q_i(T_i))$ will be reported by at least one query in $Q_i(T_i)$.

Analysis. We now analyze the expected length of Alice’s encoding. If Alice’s encoding starts with a 0-bit, we know that Alice’s encoding is exactly $1 + n_i d'$ bits. Alice’s encoding starts with a 0-bit
only when less than \( n_i/8 \) points land in \( \Gamma_r(Q_i(T_i)) \) or there exists two points in \( U_i \) within distance at most \( 0.49d' \). This occurs with probability at most \( 1/2 + 1/n \) by Lemma 3.9 and Lemma 3.2.

On the other hand, consider the case when Alice’s encoding starts with a 1-bit. In this case, Alice’s encoding length is

\[
1 + 2w(|T_i| + |C_{i-1}(U)| + \ldots + |C_0(U)|) + m + n_i + (n_i - F)d' + F \log n_i.
\]

Note, that \( |C_{i-1}(U)| + \ldots + |C_0(U)| = \Theta(n_i/\beta) \) by our choice of \( \beta = (t_u w)^2 \). We chose epochs such that \( n_i \geq m^2 \), so \( m = o(n_i) \). For sufficiently large \( d' > \log n_i \), we know that Alice’s encoding is maximized when \( F = n_i/8 \):

\[
\frac{7n_i d'}{8} + \frac{n_i \log n_i}{8} + o(n_i \log n_i).
\]

Denote \( p \leq 1/2 + 1/n \) as the probability that Alice’s encoding starts with a 0-bit. Then, Alice’s encoding length in expectation is

\[
p(1 + n_i d') + (1 - p) \left( \frac{7n_i d'}{8} + \frac{n_i \log n_i}{8} + o(n_i \log n_i) \right)
\]

which is maximized when \( p = 1/2 + 1/n \). So, Alice’s expected encoding length is at most

\[
\frac{15n_i d'}{16} + \frac{n_i}{16} \log n_i + o(n_i \log n_i) < n_i(d' - o(1)) = H(U_i \mid C_i^{\text{addr}}(U), R, u_{-i}).
\]

This contradicts Shannon’s source coding theorem completing the proof.

\[\square\]

### 3.3 Bounding Cell Probes to All Epochs

In this section, we complete the proof of Lemma 3.5 using Lemma 3.4. Our proof for Lemma 3.5 will apply obliviousness to Lemma 3.4. The main idea is that any adversary with the ability can view the number of probes to cells in the sets \( C_{k-1}(U), \ldots, C_1(U) \). Lemma 3.4 states that the expected running time for queries chosen uniformly at random from \( P_i \) requires probing \( \Omega(d'/\log(t_u w)) \) cells from \( C_i(U) \) with high constant probability. If queries from outside of the \( P_i \) were to probe significantly less cells from \( C_i(U) \), the adversary can distinguish queries that lie in \( P_i \) as opposed to those outside which would contradict the obliviousness of the data structure. We now formalize these ideas to prove Lemma 3.5.

#### 3.3.1 Proof of Lemma 3.5

**Proof of Lemma 3.5.** By Lemma 3.4, we know that if we pick a query point \( q_i \) uniformly at random from a subset of \( P_i \), then \( \Pr[t_i(U, q_i) \geq \gamma(d'/\log(t_u w))] \geq 1/2 \) for some constant \( \gamma \) and for all epochs \( i \in \{0, \ldots, k - 1\} \). We now consider two sequences of operations which both start with \( U \). For the first sequence, the query \( q_i \) is chosen uniformly at random from \( P_i \). For the second sequence, the query is chosen as any query point outside of \( P_i \) (that is \( q \notin P_i \)).

We note that an adversary can compute the sets of cells \( C_{k-1}(U), \ldots, C_0(U) \) by simply executing the update operations in \( U \) and keep tracking of the last time the contents of a cell were updated. Furthermore, for any query point \( q \), an adversary can compute \( t_i(U, q) \) by simply counting the number of probes performed to cells in the set \( C_i(U) \).

Suppose that \( t_i(U, q) < (\gamma/4) \cdot (d'/\log(t_u w)) \) which implies that \( \Pr[t_i(U, q) \geq \gamma(d'/\log(t_u w))] \leq 1/4 \) by Markov’s inequality. The adversary can now apply the following distinguisher to differentiate
the two sequences where the final query is $q_i$ chosen uniformly at random from $Q_i$ or $q \notin Q_i$. The adversary computes the number of probes to cells in the set $C_i(U)$. If the number of probes is less than $\gamma \cdot (d'/\lg(t_uw))$, then the adversary outputs 0. Otherwise, the adversary outputs 1. As a result, the adversary distinguishes the two sequences with probability at least $1/4$ contradicting obliviousness. If we pick the query point $q$ such that $q \notin P_i$ for all epochs $i \in \{k-1, \ldots, 0\}$, we apply the above result simultaneously to all epochs.

Note each epoch must contain at least $\min\{\sqrt{n}, m^2\}$. Furthermore, epochs grow geometrically by a $\beta = (t_uw)^2$ factor and there are $\Theta(n)$ update operations in total. So, there are $k = \Theta((\lg(n/m))/\lg(t_uw))$ epochs completing the proof.

4 Oblivious Dynamization

Let $P$ be a decomposable problem and suppose that we have an oblivious static data structure that solves $P$ that holds $n$ items which requires storage of $S(n)$ cells, preprocessing of at most $P(n)$ cell probes before queries and answers queries in amortized $Q(n)$ cell probes. The static data structure has two functions: preprocess$^s_t$ and query$^s_t$. The preprocessing function, preprocess$^s_t$ takes as input an encryption key $K^{enc}$ and a set of items encrypted under $K^{enc}$. The output of the preprocessing function is the data structure’s memory as well as a query key $K^{st}$. The query algorithm takes as input the query key $K^{st}$ as well as the queried argument and outputs the result as well as the possibly updated static data structure’s memory. We assume that both the preprocessing and queries are performed obliviously. That is, the adversary’s view of the preprocessing and queries are independent of the underlying items and sequence of operations. Using this oblivious static data structure in a blackbox manner, we will construct an oblivious dynamic data structure which support updating the underlying data.

Theorem 4.1. If there exists an oblivious static data structure for a decomposable problem $P$ of $n$ items with storage of $S^s_t(n)$ cells, preprocessing of $P^s_t(n)$ cell probes and amortized $Q^s_t(n)$ cell probes for queries, then there exists an oblivious dynamic data structure for $P$ using $S^{dy}(n) = O(\sum_{i=1}^{\lg n} S^{st}(2^i))$ cells of storage and amortized $Q^{dy}(n) = O(\sum_{i=1}^{\lg n} Q^{st}(2^i) + \sum_{i=1}^{\lg n} \frac{P^{st}(2^i)}{2})$ cell probes for each query/insert operation.

Proof. We assume that the oblivious dynamic data structure is initially empty and assume that the number of operations, $n$, is a power of two for convenience. When $n$ is not a power of two, one can replace all $\lg n$ with $\lceil \lg n \rceil$ to get the correct bounds. We construct our dynamic data structure by initializing $\lg n$ levels of geometrically increasing levels. Level $i$ will be initialized using an oblivious static data structure with a data set of $2^i$ items. We will denote level $i$ as $L_i$. To satisfy the requirements of obliviousness, we must hide from the adversary whether we are performing a query or insertion operation. To do this, we simply perform both for each operation. In particular, we will perform a query first before performing the insertion operation where exactly one of the query or the insertion will be a fake operation. Fake insertions will insert a ⊥ and not affect future operations while fake queries will perform an arbitrary query and ignore the result. We now formally present our preprocess$^{dy}$, query$^{dy}$ and insert$^{dy}$ algorithms.

$$(K^{enc}, K^{st}_1, \ldots, K^{st}_{\lg n}), (L_1, \ldots, L_{\lg n}, S_1, \ldots, S_{\lg n}) \leftarrow \text{preprocess}^{dy}().$$

Our preprocessing algorithm simply initializes all $\lg n$ levels to be empty and generate an encryption key.
1. Generate encryption key $\mathcal{K}^{\text{enc}}$.

2. For each $i = 1, \ldots, \lg n$:
   
   (a) Set $L_i \leftarrow \bot$.
   (b) Set $\mathcal{K}^{\text{st}}_i \leftarrow \bot$.
   (c) Set $S_i \leftarrow \emptyset$.

$r, (L_1, \ldots, L_{\lg n}) \leftarrow \text{query}^{\text{dy}}((q, \mathcal{K}^{\text{st}}_1, \ldots, \mathcal{K}^{\text{st}}_{\lg n}), (L_1, \ldots, L_{\lg n}))$. Our query algorithm receives as input a query $q$.

1. Set $r \leftarrow \bot$.

2. For each $i = 1, \ldots, \lg n$:

   (a) If $L_i \neq \bot$, then execute $r_i \leftarrow \text{query}^{\text{st}}(\mathcal{K}^{\text{st}}_i, q, L_i)$.

3. Return $r$.

$(\mathcal{K}^{\text{st}}_1, \ldots, \mathcal{K}^{\text{st}}_{\lg n}), (L_1, \ldots, L_{\lg n}, S_1, \ldots, S_{\lg n}) \leftarrow \text{insert}^{\text{st}}(\mathcal{K}^{\text{enc}}, x)$. Our insertion algorithm receives as input the item that should be inserted $x$.

1. Find minimum $k$ such that $L_i = \bot$.

2. Set $S_k \leftarrow \text{Enc}(\mathcal{K}^{\text{enc}}, \{x\} \cup S_1 \cup \ldots \cup S_{k-1})$.

3. Set $(L_k, \mathcal{K}^{\text{st}}_k) \leftarrow \text{preprocess}^{\text{st}}(\mathcal{K}^{\text{enc}}, S_k)$ where $\mathcal{K}^{\text{st}}_k$ is the privacy key used to query the oblivious static data structure.

4. For each $i = 1, \ldots, k-1$:

   (a) Set $L_i \leftarrow \bot$.
   (b) Set $\mathcal{K}^{\text{st}}_i \leftarrow \bot$.
   (c) Set $S_i \leftarrow \emptyset$.

We now analyze the costs for our dynamic data structure. Note that the query algorithm requires performing at most $\lg n$ queries to static data structures of size at most $n$ resulting in $\sum_{i=1}^{\lg n} Q^{\text{st}}(2^i) = O(\lg n \cdot Q^{\text{st}}(n))$ cell probes. For the insert algorithm, we perform an amortized analysis over $n$ queries. Level $i$ is reconstructed every $n/2^i$ operations with $P^{\text{st}}(2^i)$ cell probes. As a result, the total cost over all $n$ queries is $\sum_{i=1}^{\lg n} P^{\text{st}}(2^i) \cdot n/2^i$ or $\sum_{i=1}^{\lg n} P^{\text{st}}(2^i)/2^i$ amortized over all $n$ queries. The total storage is always at most $\sum_{i=1}^{\lg n} S^{\text{st}}(2^i)$.

Finally, we analyze the obliviousness of our data structure. We note that the schedule of constructing static data structures as well as querying is completely deterministic and independent of the stored data, input arguments to operations as well as previous updates. As we assume the queries to each static data structure and the preprocessing to construct the static data structure are oblivious, our dynamic data structure also provides obliviousness. \qed
References


