# CS369E: Communication Complexity (for Algorithm Designers) Lecture #6: Data Structure Lower Bounds[*]

Tim Roughgarden[†]

February 12 & 19, 2015

## 1 Preamble

Next we discuss how to use communication complexity to prove lower bounds on the performance — meaning space, query time, and approximation — of data structures. Our case study will be the high-dimensional approximate nearest neighbor problem.

There is a large literature on data structure lower bounds. There are several different ways to use communication complexity to prove such lower bounds, and we'll unfortunately only have time to discuss one of them. For example, we discuss only a static data structure problem — where the data structure can only be queried, not modified — and lower bounds for dynamic data structures tend to use somewhat different techniques. See [8, 10] for some starting points for further reading.

We focus on the approximate nearest neighbor problem for a few reasons: it is obviously a fundamental problem, that gets solved all the time (in data mining, for example); there are some non-trivial upper bounds; for certain parameter ranges, we have matching lower bounds; and the techniques used to prove these lower bounds are representative of work in the area — asymmetric communication complexity and reductions from the "Lopsided Disjointness" problem.

## 2 The Approximate Nearest Neighbor Problem

In the *nearest neighbor problem*, the input is a set $S$ of $n$ points that lie in a metric space $(X, \ell)$. Most commonly, the metric space is Euclidean space ($\mathcal{R}^d$ with the $\ell_2$ norm). In these lectures, we'll focus on the Hamming cube, where $X = \{0,1\}^d$ and $\ell$ is Hamming distance.

[†] Department of Computer Science, Stanford University, 474 Gates Building, 353 Serra Mall, Stanford, CA 94305. Email: `tim@cs.stanford.edu`.

On the upper bound side, the high-level ideas (related to “locality sensitive hashing (LSH)”) we use are also relevant for Euclidean space and other natural metric spaces — we’ll get a glimpse of this at the very end of the lecture. On the lower bound side, you won’t be surprised to hear that the Hamming cube is the easiest metric space to connect directly to the standard communication complexity model. Throughout the lecture, you’ll want to think of $d$ as pretty big — say $d = \sqrt{n}$.

Returning to the general nearest neighbor problem, the goal is to build a data structure $D$ (as a function of the point set $S \subseteq X$) to prepare for all possible nearest neighbor queries. Such a query is a point $q \in X$, and the responsibility of the algorithm is to use $D$ to return the point $p^* \in S$ that minimizes $\ell(p, q)$ over $p \in S$. One extreme solution is to build no data structure at all, and given $q$ to compute $p^*$ by brute force. Assuming that computing $\ell(p, q)$ takes $O(d)$ time, this query algorithm runs in time $O(dn)$. The other extreme is to pre-compute the answer to every possible query $q$, and store the results in a look-up table. For the Hamming cube, this solution uses $\Theta(d2^d)$ space, and the query time is that of one look-up in this table. The exact nearest neighbor problem is believed to suffer from the “curse of dimensionality,” meaning that a non-trivial query time (sublinear in $n$, say) requires a data structure with space exponential in $d$.

There have been lots of exciting positive results for the $(1 + \epsilon)$-*approximate* version of the nearest neighbor problem, where the query algorithm is only required to return a point $p$ with $\ell(p, q) \leq (1 + \epsilon)\ell(p^*, q)$, where $p^*$ is the (exact) nearest neighbor of $q$. This is the problem we discuss in these lectures. You’ll want to think of $\epsilon$ as a not-too-small constant, perhaps 1 or 2. For many of the motivating applications of the nearest-neighbor problem — for example, the problem of detecting near-duplicate documents (e.g., to filter search results) — such approximate solutions are still practically relevant.

# 3 An Upper Bound: Biased Random Inner Products

In this section we present a non-trivial data structure for the $(1 + \epsilon)$-approximate nearest neighbor problem in the Hamming cube. The rough idea is to hash the Hamming cube and then precompute the answer for each of the hash table’s buckets — the trick is to make sure that nearby points are likely to hash to the same bucket. Section 4 proves a sense in which this data structure is the best possible: no data structure for the problem with equally fast query time uses significantly less space.

## 3.1 The Key Idea (via a Public-Coin Protocol)

For the time being, we restrict attention to the decision version of the $(1+\epsilon)$-nearest neighbor problem. Here, the data structure construction depends both on the point set $S \subseteq \{0, 1\}^d$ and on a given parameter $L \in \{0, 1, 2, \ldots, d\}$. Given a query $\mathbf{q}$, the algorithm only has to decide correctly between the following cases:

1. There exists a point $p \in S$ with $\ell(p, q) \leq L$.

2. $\ell(p, q) \geq (1 + \epsilon)L$ for every point $p \in S$.

If neither of these two cases applies, then the algorithm is off the hook (either answer is regarded as correct). We'll see in Section 3.3 how, using this solution as an ingredient, we can build a data structure for the original version of the nearest neighbor problem.

Recall that *upper bounds* on communication complexity are always suspect — by design, the computational model is extremely powerful so that lower bounds are as impressive as possible. There are cases, however, where designing a good communication protocol reveals the key idea for a solution that is realizable in a reasonable computational model. Next is the biggest example of this that we'll see in the course.

In the special case where $S$ contains only one point, the decision version of the $(1 + \epsilon)$-approximate nearest neighbor problem resembles two other problems that we've studied before in other contexts, one easy and one hard.

1. Equality. Recall that when Alice and Bob just want to decide whether their inputs are the same or different — equivalently, deciding between Hamming distance 0 and Hamming distance at least 1 — there is an unreasonably effective (public-coin) randomized communication protocol for the problem. Alice and Bob interpret the first $2n$ public coins as two random $n$-bits strings $\mathbf{r}_1, \mathbf{r}_2$. Alice sends the inner product modulo 2 of her input $\mathbf{x}$ with $\mathbf{r}_1$ and $\mathbf{r}_2$ (2 bits) to Bob. Bob accepts if and only if the two inner products modulo 2 of his input $\mathbf{y}$ with $\mathbf{r}_1, \mathbf{r}_2$ match those of Alice. This protocol never rejects inputs with $\mathbf{x} = \mathbf{y}$, and accepts inputs with $\mathbf{x} \neq \mathbf{y}$ with probability 1/4.

2. Gap Hamming. Recall in this problem Alice and Bob want to decide between the cases where the Hamming distance between their inputs is at most $\frac{n}{2} - \sqrt{n}$, or at least $\frac{n}{2} + \sqrt{n}$. We proved in Lecture #3 that this problem is hard for 1-way communication protocols (via a clever reduction from Index); it is also hard for general communication protocols [2, 11, 12]. Note however that the gap between the two cases is very small, corresponding to $\epsilon \approx \frac{2}{\sqrt{n}}$. In the decision version of the $(1 + \epsilon)$-approximate nearest neighbor problem, we're assuming a constant-factor gap in Hamming distance between the two cases, so there's hope that the problem is easier.

Consider now the communication problem where Alice and Bob want to decide if the Hamming distance $\ell_H(\mathbf{x}, \mathbf{y})$ between their inputs $\mathbf{x}, \mathbf{y} \in \{0, 1\}^d$ is at most $L$ or at least $(1 + \epsilon)L$. We call this the $\epsilon$-Gap Hamming problem. We analyze the following protocol; we'll see shortly how to go from this protocol to a data structure.

1. Alice and Bob use the public random coins to choose $s$ random strings $R = \{\mathbf{r}_1, \ldots, \mathbf{r}_s\} \in \{0, 1\}^d$, where $d = \Theta(\epsilon^{-2})$. The strings are *not* uniformly random: each entry is chosen independently, with probability $1/2L$ of being equal to 1.

2. Alice sends the $s$ inner products (modulo 2) $\langle \mathbf{x}, \mathbf{r}_1 \rangle, \ldots, \langle \mathbf{x}, \mathbf{r}_s \rangle$ of her input and the random strings to Bob — a "hash value" $h_R(\mathbf{x}) \in \{0, 1\}^s$.

3. Bob accepts if and only if the Hamming distance between the corresponding hash value $h_R(\mathbf{y})$ of his input — the $s$ inner products (modulo 2) of $\mathbf{y}$ with the random strings in $R$ — differs from $h_R(\mathbf{x})$ in only a "small" (TBD) number of coordinates.

Intuitively, the goal is to modify our randomized communication protocol for Equality so that it continues to accept strings that are close to being equal. A natural way to do this is to bias the coefficient vectors significantly more toward 0 than before. For example, if $\mathbf{x}, \mathbf{y}$ differ in only a single bit, then choosing $\mathbf{r}$ uniformly at random results in $\langle \mathbf{x}, \mathbf{r} \rangle \not\equiv \langle \mathbf{y}, \mathbf{r} \rangle \bmod 2$ with probability $1/2$ (if and only if $\mathbf{r}$ has a 1 in the coordinate where $\mathbf{x}, \mathbf{y}$ differ). With probability $1/2L$ of a 1 in each coordinate of $\mathbf{r}$, the probability that $\langle \mathbf{x}, \mathbf{r} \rangle \not\equiv \langle \mathbf{y}, \mathbf{r} \rangle \bmod 2$ is only $1/2L$. Unlike our Equality protocol, this protocol for $\epsilon$-Gap Hamming has two-sided error.

For the analysis, it's useful to think of each random choice of a coordinate $r_{ji}$ as occurring in two stages: in the first stage, the coordinate is deemed *relevant* with probability $1/L$ and *irrelevant* otherwise. In stage 2, $r_{ji}$ is set to 0 if the coordinate is irrelevant, and chosen uniformly from $\{0, 1\}$ if the coordinate is relevant. We can therefore think of the protocol as: (i) first choosing a subset of relevant coordinates; (ii) running the old Equality protocol on these coordinates only. With this perspective, we see that if $\ell_H(\mathbf{x}, \mathbf{y}) = \Delta$, then

$$\mathbf{Pr}_{\mathbf{r}_j}[\langle \mathbf{r}_j, \mathbf{x} \rangle \not\equiv \langle \mathbf{r}_j, \mathbf{y} \rangle \bmod 2] = \frac{1}{2}\left(\left(1 - \left(1 - \frac{1}{L}\right)^{\Delta}\right)\right) \tag{1}$$

for every $\mathbf{r}_j \in R$. In (1), the quantity inside the outer parentheses is exactly the probability that at least one of the $\Delta$ coordinates on which $\mathbf{x}, \mathbf{y}$ differ is deemed relevant. This is a necessary condition for the event $\langle \mathbf{r}_j, \mathbf{x} \rangle \not\equiv \langle \mathbf{r}_j, \mathbf{y} \rangle \bmod 2$ and, in this case, the conditional probability of this event is exactly $\frac{1}{2}$ (as in the old Equality protocol).

The probability in (1) is an increasing function of $\Delta$, as one would expect. Let $t$ denote the probability in (1) when $\Delta = L$. We're interested in how much bigger this probability is when $\Delta$ is at least $(1 + \epsilon)L$. We can take the difference between these two cases and bound it below using the fact that $1 - x \in [e^{-2x}, e^{-x}]$ for $x \in [0, 1]$:

$$\tfrac{1}{2}\left[\underbrace{\left(1 - \frac{1}{L}\right)^{L}}_{\geq e^{-2}}\left(1 - \underbrace{\left(1 - \frac{1}{L}\right)^{\epsilon L}}_{\leq e^{-\epsilon}}\right)\right] \geq \frac{1}{2\epsilon^2}\left(1 - e^{-\epsilon}\right) := h(\epsilon).$$

Note that $h(\epsilon)$ is a constant, depending on $\epsilon$ only. Thus, with $s$ random strings, if $\ell_H(\mathbf{x}, \mathbf{y}) \leq \Delta$ then we expect $ts$ of the random inner products (modulo 2) to be different; if $\ell_H(\mathbf{x}, \mathbf{y}) \geq (1 + \epsilon)\Delta$, then we expect at least $(t + h(\epsilon))s$ of them to be different. A routine application of Chernoff bounds implies the following.

**Corollary 3.1** *Define the hash function $h_R$ as in the communication protocol above. If $s = \Omega(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$, then with probability at least $1 - \delta$ over the choice of $R$:*

*(i) If $\ell_H(\mathbf{x},\mathbf{y}) \le L$, then $\ell_H(h(\mathbf{x}),h(\mathbf{y})) \le (t + \frac{1}{2}h(\epsilon)))s$.*

*(ii) If $\ell_H(\mathbf{x},\mathbf{y}) \ge (1+\epsilon)L$, then $\ell_H(h(\mathbf{x}),h(\mathbf{y})) > (t + \frac{1}{2}h(\epsilon))s$.*

We complete the communication protocol above by defining "small" in the third step as $(t+\frac{1}{2}h(\epsilon))s$. We conclude that the $\epsilon$-Gap Hamming problem can be solved by a public-coin randomized protocol with two-sided error and communication cost $\Theta(\epsilon^{-2})$.

## 3.2 The Data Structure (Decision Version)

We now show how to translate the communication protocol above into a data structure for the $(1+\epsilon)$-nearest neighbor problem. For the moment, we continue to restrict to the decision version of the problem, for an a priori known value of $L \in \{0,1,2,\ldots,d\}$. All we do is precompute the answers for all possible hash values of a query (an "inverted index").

Given a point set $P$ of $n$ points in $\{0,1\}^d$, we choose a set $R$ of $s = \Theta(\epsilon^{-2}\log n)$ random strings $\mathbf{r}_1,\ldots,\mathbf{r}_s$ according to the distribution of the previous section (with a "1" chosen with probability $1/2L$). We again define the hash function $h_R : \{0,1\}^d \to \{0,1\}^s$ by setting the $j$th coordinate of $h_R(\mathbf{x})$ to $\langle \mathbf{r}_j, \mathbf{x}\rangle \bmod 2$. We construct a table with $2^s = n^{\Theta(\epsilon^{-2})}$ buckets, indexed by $s$-bit strings.[1] Then, for each point $\mathbf{p} \in P$, we insert $\mathbf{p}$ into every bucket $\mathbf{b} \in \{0,1\}^s$ for which $\ell_H(h_R(\mathbf{p}),\mathbf{b}) \le (t + \frac{1}{2}h(\epsilon))s$, where $t$ is defined as in the previous section (as the probability in (1) with $\Delta = L$). This preprocessing requires $n^{\Theta(\epsilon^{-2})}$ time.

With this data structure in hand, answering a query $\mathbf{q} \in \{0,1\}^d$ is easy: just compute the hash value $h_R(\mathbf{q})$ and return an arbitrary point of the corresponding bucket, or "none" if this bucket is empty.

For the analysis, think of an adversary who chooses a query point $\mathbf{q} \in \{0,1\}^d$, and then we subsequently flip our coins and build the above data structure. (This is the most common way to analyze hashing, with the query independent of the random coin flips used to choose the hash function.) Choosing the hidden constant in the definition of $s$ appropriately and applying Corollary 3.1 with $\delta = \frac{1}{n^2}$, we find that, for every point $\mathbf{p} \in P$, with probability at least $1 - \frac{1}{n^2}$, $\mathbf{p}$ is in $h(\mathbf{q})$ (if $\ell_H(\mathbf{p},\mathbf{q}) \le L$) or is not in $h(\mathbf{q})$ (if $\ell_h(\mathbf{p},\mathbf{q}) \ge (1+\epsilon)L$). Taking a Union Bound over the $n$ points of $P$, we find that the data structure correctly answers the query $\mathbf{q}$ with probability at least $1 - \frac{1}{n}$.

Before describing the full data structure, let's take stock of what we've accomplished thus far. We've shown that, for every constant $\epsilon > 0$, there is a data structure for the decision version of the $(1+\epsilon)$-nearest neighbor problem that uses space $n^{O(\epsilon^{-2})}$, answers a query with a single random access to the data structure, and for every query is correct with high probability. Later in this lecture, we show a matching lower bound: every (possibly randomized) data structure with equally good search performance for the decision version of the $(1+\epsilon)$-nearest neighbor problem has space $n^{\Omega(\epsilon^{-2})}$. Thus, smaller space can only be achieved by increasing the query time (and there are ways to do this, see e.g. [7]).

[1] Note the frightening dependence of the space on $\frac{1}{\epsilon}$. This is why we suggested thinking of $\epsilon$ as a not-too-small constant.

## 3.3 The Data Structure (Full Version)

The data structure of the previous section is an unsatisfactory solution to the $(1+\epsilon)$-nearest neighbor problem in two respects:

1. In the real problem, there is no a priori known value of $L$. Intuitively, one would like to take $L$ equal to the actual nearest-neighbor distance of a query point $\mathbf{q}$, a quantity that is different for different $\mathbf{q}$'s.

2. Even for the decision version, the data structure can answer some queries incorrectly. Since the data structure only guarantees correctness with probability at least $1-\frac{1}{n}$ for each query $\mathbf{q}$, it might be wrong on as many as $2^d/n$ different queries. Thus, an adversary that knows the data structure's coin flips can exhibit a query that the data structure gets wrong.

The first fix is straightforward: just build $\approx d$ copies of the data structure of Section 3.2, one for each relevant value of $L$.[2] Given a query $\mathbf{q}$, the data structure now uses binary search over $L$ to compute a $(1+\epsilon)$-nearest neighbor of $\mathbf{q}$; see the Exercises for details. Answering a query thus requires $O(\log d)$ lookups to the data structure. This also necessitates blowing up the number $s$ of random strings used in each data structure by a $\Theta(\log\log d)$ factor — this reduces the failure probability of a given lookup by a $\log d$ factor, enabling a Union Bound over $\log d$ times as many lookups as before.[3]

A draconian approach to the second problem is to again replicate the data structure above $\Theta(d)$ times. Each query $\mathbf{q}$ is asked in all $\Theta(d)$ copies, and majority vote is used to determine the final answer. Since each copy is correct on each of the $2^d$ possible queries with probability at least $1-\frac{1}{n} > \frac{2}{3}$, the majority vote is wrong on a given query with probability at most inverse exponential in $d$. Taking a Union Bound over the $2^d$ possible queries shows that, with high probability over the coin flips used to construct the data structure, the data structure answers every query correctly. Put differently, for almost all outcomes of the coin flips, not even an adversary that knows the coin flip outcomes can produce a query on which the data structure is incorrect. This solution blows up both the space used and the query time by a factor of $\Theta(d)$.

An alternative approach is to keep $\Theta(d)$ copies of the data structure as above but, given a query $\mathbf{q}$, to answer the query using one of the $\Theta(d)$ copies chosen uniformly at random. With this solution, the space gets blown up by a factor of $\Theta(d)$ but the query time is unaffected. The correctness guarantee is now slightly weaker. With high probability over the coin flips used to construct the data structure (in the preprocessing), the data structure satisfies: for every query $\mathbf{q} \in \{0,1\}^d$, with probability at least $1-\Theta(\frac{1}{n})$ over the coins flipped at query time, the data structure answers the query correctly (why?). Equivalently, think of an adversary who is privy to the outcomes of the coins used to construct the data structure, but

[2] For $L \geq d/(1+\epsilon)$, the data structure can just return an arbitrary point of $P$. For $L=0$, when the data structure of Section 3.2 is not well defined, a standard data structure for set membership, such as a perfect hash table [4], can be used.

[3] With some cleverness, this $\log\log d$ factor can be avoided – see the Exercises.

not those used to answer queries. For most outcomes of the coins used in the preprocessing phase, no matter what query the adversary suggests, the data structure answers the query correctly with high probability.

To put everything together, the data structure for a fixed $L$ (from Section 3.2) requires $n^{\Theta(\epsilon^{-2})}$ space, the first fix blows up the space by a factor of $\Theta(d \log d)$, and the second fix blows up the space by another factor of $d$. For the query time, with the alternative implementation of the second fix, answering a query involves $O(\log d)$ lookups into the data structure. Each lookup involves computing a hash value, which in turn involves computing inner products (modulo 2) with $s = \Theta(\epsilon^{-2} \log n \log\log d)$ random strings. Each such inner product requires $O(d)$ time.

Thus, the final scorecard for the data structure is:

- Space: $O(d^2 \log d) \cdot n^{\Theta(\epsilon^{-2})}$.
- Query time: $O(\epsilon^{-2} d \log n \log\log d)$.

For example, for the suggested parameter values of $d = n^c$ for a constant $c \in (0,1)$ and $\epsilon$ a not-too-small constant, we obtain a query time significantly better than the brute-force (exact) solution (which is $\Theta(dn)$), while using only a polynomial amount of space.

# 4 Lower Bounds via Asymmetric Communication Complexity

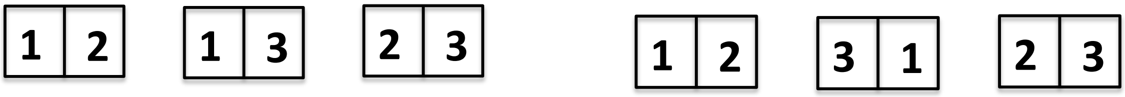
We now turn our attention from upper bounds for the $(1+\epsilon)$-nearest neighbor problem to lower bounds. We do this in three steps. In Section 4.1, we introduce a model for proving lower bounds on time-space trade-offs in data structures — the *cell probe model*. In Section 4.2, we explain how to deduce lower bounds in the cell probe model from a certain type of communication complexity lower bound. Section 4.3 applies this machinery to the $(1+\epsilon)$-approximate nearest neighbor problem, and proves a sense in which the data structure of Section 3 is optimal: every data structure for the decision version of the problem that uses $O(1)$ lookups per query has space $n^{\Omega(\epsilon^{-2})}$. Thus, no polynomial-sized data structure can be both super-fast and super-accurate for this problem.

## 4.1 The Cell Probe Model

### 4.1.1 Motivation

The most widely used model for proving data structure lower bounds is the *cell probe model*, introduced by Yao — two years after he developed the foundations of communication complexity [13] — in the paper "Should Tables Be Sorted?" [14].[4] The point of this model is to prove lower bounds for data structures that allow random access. To make the model as

[4] Actually, what is now called the cell probe model is a bit stronger than the model proposed in [14].



(a) Sorted Array

(b) Unsorted Array

Figure 1: For $n = 3$ and $k = 2$, it is suboptimal to store the array elements in sorted order.

powerful as possible, and hence lower bounds for it as strong as possible (cf., our communication models), a random access to a data structure counts as 1 step in this model, no matter how big the data structure is.

### 4.1.2 Some History

To explain the title of [14], suppose your job is to store $k$ elements from a totally ordered universe of $n$ elements ($\{1, 2, \ldots, n\}$, say) in an array of length $k$. The goal is to minimize the worst-case number of array accesses necessary to check whether or not a given element is in the array, over all subsets of $k$ elements that might be stored and over the $n$ possible queries.

To see that this problem is non-trivial, suppose $n = 3$ and $k = 2$. One strategy is to store the pair of elements in sorted order, leading to the three possible arrays in Figure 1(a). This yields a worst-case query time of 2 array accesses. To see this, suppose we want to check whether or not 2 is in the array. If we initially query the first array element and find a "1," or if we initially query the second array element and find a "3," then we can't be sure whether or not 2 is in the array.

Suppose, on the other hand, our storage strategy is as shown in Figure 1(b). Whichever array entry is queried first, the answer uniquely determines the other array entry. Thus, storing the table in a non-sorted fashion is necessary to achieve the optimal worst-case query time of 1. On the other hand, if $k = 2$ and $n = 4$, storing the elements in sorted order (and using binary search to answer queries) is optimal![5]

### 4.1.3 Formal Model

In the cell problem model, the goal is to encode a "database" $D$ in order to answer a set $Q$ of queries. The query set $Q$ is known up front; the encoding scheme must work (in the sense below) for all possible databases.

For example, in the $(1 + \epsilon)$-approximate nearest neighbor problem, the database corresponds to the point set $P \subseteq \{0, 1\}^d$, while the possible queries $Q$ correspond to the elements

[5] Yao [14] also uses Ramsey theory to prove that, provided the universe size is a sufficiently (really, really) large function of the array size, then binary search on a sorted array is optimal. This result assumes that no auxiliary storage is allowed, so solutions like perfect hashing [4] are ineligible. If the universe size is not too much larger than the array size, then there are better solutions [3, 5, 6], even when there is no auxiliary storage.

of $\{0,1\}^d$. Another canonical example is the set membership problem: here, $D$ is a subset of a universe $U$, and each query $q \in Q$ asks "is $i \in D$?" for some element $i \in U$.

A parameter of the cell probe model is the *word size* $w$; more on this shortly. Given this parameter, the design space consists of the ways to encode databases $D$ as $s$ *cells* of $w$ bits each. We can view such an encoding as an abstract data structure representing the database, and we view the number $s$ of cells as the *space* used by the encoding. To be a valid encoding, it must be possible to correctly answer every query $q \in Q$ for the database $D$ by reading enough bits of the encoding. A query-answering algorithm accesses the encoding by specifying the name of a cell; in return, the algorithm is given the contents of that cell. Thus every access to the encoding yields $w$ bits of information. The *query time* of an algorithm (with respect to an encoding scheme) is the maximum, over databases $D$ (of a given size) and queries $q \in Q$, number of accesses to the encoding used to answer a query.

For example, in the original array example from [14] mentioned above, the word size $w$ is $\lceil \log_2 n \rceil$ — just enough to specify the name of an element. The goal in [14] was, for databases consisting of $k$ elements of the universe, to understand when the minimum-possible query time is $\lceil \log_2 k \rceil$ under the constraint that the space is $k$.[6]

Most research on the cell-probe model seeks time-space trade-offs with respect to a fixed value for the word size $w$. Most commonly, the word size is taken large enough so that a single element of the database can be stored in a single cell, and ideally not too much larger than this. For nearest-neighbor-type problems involving $n$-point sets in the $d$-dimensional hypercube, this guideline suggests taking $w$ to be polynomial in $\max\{d, \log_2 n\}$.

For this choice of $w$, the data structure in Section 3.2 that solves the decision version of the $(1+\epsilon)$-approximate nearest neighbor problem yields a (randomized) cell-probe encoding of point sets with space $n^{\Theta(\epsilon^{-2})}$ and query time 1. Cells of this encoding correspond to all possible $s = \Theta(\epsilon^{-2} \log n)$-bit hash values $h_R(\mathbf{q})$ of a query $\mathbf{q} \in \{0,1\}^d$, and the contents of a cell name an arbitrary point $\mathbf{p} \in P$ with hash value $h_R(\mathbf{p})$ sufficiently close (in Hamming distance in $\{0,1\}^s$) to that of the cell's name (or "NULL" if no such $\mathbf{p}$ exists). The rest of this lecture proves a matching lower bound in the cell-probe model: constant query time can only be achieved by encodings (and hence data structures) that use $n^{\Omega(\epsilon^{-2})}$ space.

### 4.1.4 From Data Structures to Communication Protocols

Our goal is to derive data structure lower bounds in the cell-probe model from communication complexity lower bounds. Thus, we need to extract low-communication protocols from good data structures. Similar to our approach last lecture, we begin with a contrived communication problem to forge an initial connection. Later we'll see how to prove lower bounds for the contrived problem via reductions from other communication problems that we already understand well.

Fix an instantiation of the cell probe model – i.e., a set of possible databases and possible queries. For simplicity, we assume that all queries are Boolean. In the corresponding Query-Database problem, Alice gets a query $q$ and Bob gets a database $D$. (Note that in all

[6] The model in [14] was a bit more restrictive — cells were required to contain names of elements in the database, rather than arbitrary $\lceil \log_2 n \rceil$-bit strings.

natural examples, Bob's input is *much* bigger than Alice's.) The communication problem is to compute the answer to $q$ on the database $D$.

We made up the Query-Database problem so that the following lemma holds.

**Lemma 4.1** *Consider a set of databases and queries so that there is a cell-probe encoding with word size $w$, space $s$, and query time $t$. Then, there is a communication protocol for the corresponding* Query-Database *problem with communication at most*

$$\underbrace{t \log_2 s}_{\textit{bits sent by Alice}} + \underbrace{tw}_{\textit{bits sent by Bob}} .$$

The proof is the obvious simulation: Alice simulates the query-answering algorithm, sending at most $\log_2 s$ bits to Bob specify each cell requested by the algorithm, and Bob sends $w$ bits back to Alice to describe the contents of each requested cell. By assumption, they only need to go back-and-forth at most $t$ times to identify the answer to Alice's query $q$.

Lemma 4.1 reduces the task of proving data structure lower bounds to proving lower bounds on the communication cost of protocols for the Query-Database problem.[7]

## 4.2 Asymmetric Communication Complexity

Almost all data structure lower bounds derived from communication complexity use *asymmetric* communication complexity. This is just a variant of the standard two-party model where we keep track of the communication by Alice and by Bob separately. The most common motivation for doing this is when the two inputs have very different sizes, like in the protocol used to prove Lemma 4.1 above.

### 4.2.1 Case Study: Index

To get a feel for asymmetric communication complexity and lower bound techniques for it, let's revisit an old friend, the Index problem. In addition to the application we saw earlier in the course, Index arises naturally as the Query-Database problem corresponding to the membership problem in data structures.

Recall that an input of Index gives Alice an index $i \in \{1, 2, \ldots, n\}$, specified using $\approx \log_2 n$ bits, and Bob a subset $S \subseteq \{1, 2, \ldots, n\}$, or equivalently an $n$-bit vector.[8] In Lecture #2 we proved that the communication complexity of Index is $\Omega(n)$ for one-way randomized protocols with two-sided error — Bob must send almost his entire input to Alice

[7] The two-party communication model seems strictly stronger than the data structure design problem that it captures — in a communication protocol, Bob can remember which queries Alice asked about previously, while a (static) data structure cannot. An interesting open research direction is to find communication models and problems that more tightly capture data structure design problems, thereby implying strong lower bounds.

[8] We've reversed the roles of the players relative to the standard description we gave in Lectures #1–2. This reverse version is the one corresponding to the Query-Database problem induced by the membership problem.

for Alice to have a good chance of computing her desired index. This lower bound clearly does not apply to general communication protocols, since Alice can just send her $\log_2 n$-bit input to Bob. It is also easy to prove a matching lower bound on deterministic and nondeterministic protocols (e.g., by a fooling set argument).

We might expect a more refined lower bound to hold: to solve INDEX, not only do the players have to send at least $\log_2 n$ bits total, but more specifically *Alice* has to send at least $\log_2 n$ bits to Bob. Well not quite: Bob could always send his entire input to Alice, using $n$ bits of communication while freeing Alice to use essentially no communication. Revising our ambition, we could hope to prove that in every INDEX protocol, *either* (i) Alice has to communicate most of her input; or (ii) Bob has to communicate most of his input. The next result states that this is indeed the case.

**Theorem 4.2 ([9])** *For every $\delta > 0$, there exists a constant $N = N(\delta)$ such that, for every $n \geq N$ and every randomized communication protocol with two-sided error that solves* INDEX *with $n$-bit inputs, either:*

*(i) in the worst case (over inputs and protocol randomness), Alice communicates at least $\delta \log_2 n$ bits; or*

*(ii) in the worst case (over inputs and protocol randomness), Bob communicates at least $n^{1-2\delta}$ bits.*[9]

Loosely speaking, Theorem 4.2 states that the only way Alice can get away with sending $o(\log n)$ bits of communication is if Bob sends at least $n^{1-o(1)}$ bits of communication.

For simplicity, we'll prove Theorem 4.2 only for deterministic protocols. The lower bound for randomized protocols with two-sided error is very similar, just a little messier (see [9]).

Conceptually, the proof of Theorem 4.2 has the same flavor as many of our previous lower bounds, and is based on covering-type arguments. The primary twist is that rather than keeping track only of the size of monochromatic rectangles, we keep track of both the height and width of such rectangles. For example, we've seen in the past that low-communication protocols imply the existence of a large monochromatic rectangle — if the players haven't had the opportunity to speak much, then an outside observer hasn't had the chance to eliminate many inputs as legitimate possibilities. The next lemma proves an analog of this, with the height and width of the monochromatic rectangle parameterized by the communication used by Alice and Bob, respectively.

**Lemma 4.3 (Richness Lemma [9])** *Let $f : X \times Y \to \{0, 1\}$ be a Boolean function with corresponding $X \times Y$ 0-1 matrix $M(f)$. Assume that:*

*(1) $M(f)$ has at least $v$ columns that each have at least $u$ 1-inputs.*[10]

*(2) There is a deterministic protocol that computes $f$ in which Alice and Bob always send at most $a$ and $b$ bits, respectively.*[11]

[9] From the proof, it will be evident that $n^{1-2\delta}$ can be replaced by $n^{1-c\delta}$ for any constant $c > 1$.

[10] Such a matrix is sometimes called $(u, v)$-*rich*.

[11] This is sometimes called an $[a, b]$-*protocol*.

*Then, $M(f)$ has a 1-rectangle $A \times B$ with $|A| \geq \frac{u}{2^a}$ and $|B| \geq \frac{v}{2^{a+b}}$.*

The proof of Lemma 4.3 is a variation on the classic argument that a protocol computing a function $f$ induces a partition of the matrix $M(f)$ into monochromatic rectangles. Let's recall the inductive argument. Let $\mathbf{z}$ be a transcript-so-far of the protocol, and assume by induction that the inputs $(\mathbf{x}, \mathbf{y})$ that lead to $\mathbf{z}$ form a rectangle $A \times B$. Assume that Alice speaks next (the other case is symmetric). Partition $A$ into $A_0, A_1$, with $A_\eta$ the inputs $\mathbf{x} \in A$ such Alice sends the bit $\eta$ next. (As always, this bit depends only on her input $\mathbf{x}$ and the transcript-so-far $\mathbf{z}$.) After Alice speaks, the inputs consistent with the resulting transcript are either $A_0 \times B$ or $A_1 \times B$ — either way, a rectangle. All inputs that generate the same final transcript $\mathbf{z}$ form a monochromatic rectangle — since the protocol's output is constant across these inputs and it computes the function $f$, $f$ is also constant across these inputs.

Now let's refine this argument to keep track of the dimensions of the monochromatic rectangle, as a function of the number of times that each of Alice and Bob speak.

*Proof of Lemma 4.3:* We proceed by induction on the number of steps of the protocol. Suppose the protocol has generated the transcript-so-far $\mathbf{z}$ and that $A \times B$ is the rectangle of inputs consistent with this transcript. Suppose that at least $c$ of the columns of $B$ have at least $d$ 1-inputs in rows of $A$ (possibly with different rows for different columns).

For the first case, suppose that Bob speaks next. Partition $A \times B$ into $A \times B_0$ and $A \times B_1$, where $B_\eta$ are the inputs $\mathbf{y} \in B$ such that (given the transcript-so-far $\mathbf{z}$) Bob sends the bit $\eta$. At least one of the sets $B_0, B_1$ contains at least $c/2$ columns that each contain at least $d$ 1-inputs in the rows of $A$ (Figure 2(a)).

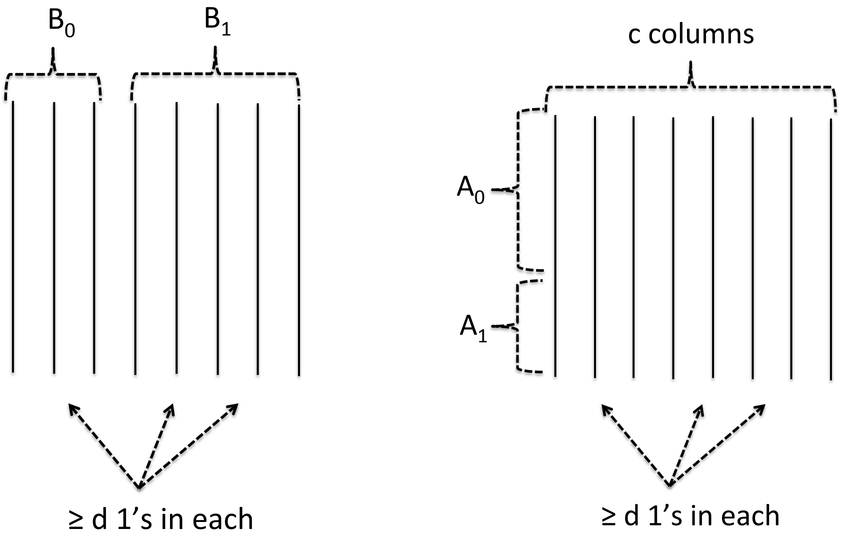
For the second case, suppose that Alice speaks. Partition $A \times B$ into $A_0 \times B$ and $A_1 \times B$. It is not possible that both (i) $A_0 \times B$ has strictly less that $c/2$ columns with $d/2$ or more 1-inputs in the rows of $A_0$ and (ii) $A_1 \times B$ has strictly less that $c/2$ columns with $d/2$ or more 1-inputs in the rows of $A_1$. For if both (i) and (ii) held, then $A \times B$ would have less than $c$ columns with $d$ or more 1-inputs in the rows of $A$, a contradiction (Figure 2(b)).

By induction, we conclude that there is a 1-input $(\mathbf{x}, \mathbf{y})$ such that, at each point of the protocol's execution on $(\mathbf{x}, \mathbf{y})$ (with Alice and Bob having sent $\alpha$ and $\beta$ bits so-far, respectively), the current rectangle $A \times B$ of inputs consistent with the protocol's execution has at least $v/2^{\alpha+\beta}$ columns (in $A$) that each contain at least $u/2^\alpha$ 1-inputs (among the rows of $B$). Since the protocol terminates with a monochromatic rectangle of $M(f)$ and with $\alpha \leq a$, $\beta \leq b$, the proof is complete. ■

Lemma 4.3 and some simple calculations prove Theorem 4.2.

*Proof of Theorem 4.2:* We first observe that the matrix $M(Index)$ has a set of $\binom{n}{n/2}$ columns that each contain $n/2$ 1-inputs: choose the columns (i.e., inputs for Bob) that correspond to subsets $S \subseteq \{1, 2, \ldots, n\}$ of size $n/2$ and, for such a column $S$, consider the rows (i.e., indices for Alice) that correspond to the elements of $S$.

Now suppose for contradiction that there is a protocol that solves INDEX in which Alice always sends at most $a = \delta \log_2 n$ bits and Bob always sends at most $b = n^{1-2\delta}$ bits. Invoking



(a) When Bob Speaks (b) When Alice Speaks

Figure 2: Proof of the Richness Lemma (Lemma 4.3). When Bob speaks, at least one of the corresponding subrectangles has at least $c/2$ columns that each contain at least $d$ 1-inputs. When Alice speaks, at least one of the corresponding subrectangles has at least $c/2$ columns that each contain at least $d/2$ 1-inputs.

Lemma 4.3 proves that the matrix $M(Index)$ has a 1-rectangle of size at least

$$\frac{n}{2}\cdot\underbrace{\frac{1}{2^a}}_{=n^{-\delta}}\times\underbrace{\binom{n}{n/2}}_{\approx 2^n/\sqrt{n}}\cdot\underbrace{\frac{1}{2^{a+b}}}_{=n^{-\delta}\cdot 2^{-n^{1-2\delta}}} = \tfrac{1}{2}n^{1-\delta}\times c_2 2^{n-n^{1-2\delta}},$$

where $c_2 > 0$ is a constant independent of $n$. (We're using here that $n \geq N(\delta)$ is sufficiently large.)

On the other hand, how many columns can there be in a 1-rectangle with $\frac{1}{2}n^{1-\delta}$ rows? If these rows correspond to the set $S \subseteq \{1, 2, \ldots, n\}$ of indices, then every column of the 1-rectangle must correspond to a superset of $S$. There are

$$2^{n-|S|} = 2^{n-\frac{1}{2}n^{1-\delta}}$$

of these. But

$$c_2 2^{n-n^{1-2\delta}} > 2^{n-\frac{1}{2}n^{1-\delta}},$$

for sufficiently large $n$, providing the desired contradiction. ■

Does the asymmetric communication complexity lower bound in Theorem 4.2 have any interesting implications? By Lemma 4.1, a data structure that supports membership queries with query time $t$, space $s$, and word size $w$ induces a communication protocol for Index in which Alice sends at most $t \log_2 s$ bits and Bob sends at most $tw$ bits. For example, suppose $t = \Theta(1)$ and $w$ at most poly-logarithmic in $n$. Since Bob only sends $tw$ bits in the induced protocol for Index, he certainly does not send $n^{1-2\delta}$ bits.[12] Thus, Theorem 4.2 implies that Alice must send at least $\delta \log_2 n$ bits in the protocol. This implies that

$$t \log_2 s \geq \delta \log_2 n$$

and hence $s \geq n^{\delta/t}$. The good news is that this is a polynomial lower bound for every constant $t$. The bad news is that even for $t = 1$, this argument will never prove a super-linear lower bound. We don't expect to prove a super-linear lower bound in the particular case of the membership problem, since there is a data structure for this problem with constant query time and linear space (e.g., perfect hashing [4]). For the $(1+\epsilon)$-approximate nearest neighbor problem, on the other hand, we want to prove a lower bound of the form $n^{\Omega(\epsilon^{-2})}$. To obtain such a super-linear lower bound, we need to reduce from a communication problem harder than Index— or rather, a communication problem in which Alice's input is bigger than $\log_2 n$ and in which she still reveals almost her entire input in every communication protocol induced by a constant-query data structure.

### 4.2.2 $(k, \ell)$-Disjointness

A natural idea for modifying Index so that Alice's input is bigger is to give Alice *multiple* indices; Bob's input remains an $n$-bit vector. The new question is whether or not for *at least one* of Alice's indices, Bob's input is a 1.[13] This problem is essentially equivalent — up to the details of how Alice's input is encoded — to Disjointness.

This section considers the special case of Disjointness where the sizes of the sets given to Alice and Bob are restricted. If we follow the line of argument in the proof of Theorem 4.2, the best-case scenario is a space lower bound of $2^{\Omega(a)}$, where $a$ is the length of Alice's input; see also the proofs of Corollary 4.7 and Theorem 4.8 at the end of the lecture. This is why the Index problem (where Alice's set is a singleton and $a = \log_2 n$) cannot lead — at least via Lemma 4.1 — to super-linear data structure lower bounds. The minimum $a$ necessary for the desired space lower bound of $n^{\Omega(\epsilon^{-2})}$ is $\epsilon^{-2} \log_2 n$. This motivates considering instances of Disjointness in which Alice receives a set of size $\epsilon^{-2}$. Formally, we define $(k, \ell)$-Disjointness as the communication problem in which Alice's input is a set of size $k$ (from a universe $U$) and Bob's input is a set of size $\ell$ (also from $U$), and the goal is to determine whether or not the sets are disjoint (a 1-input) or not (a 0-input).

We next extend the proof of Theorem 4.2 to show the following.

**Theorem 4.4 ([1, 9])** *For every $\epsilon, \delta > 0$ and every sufficiently large $n \geq N(\epsilon, \delta)$, in every communication protocol that solves $(\frac{1}{\epsilon^2}, n)$-*Disjointness *with a universe of size $2n$, either:*

[12] Here $\delta \in (0,1)$ is a constant and $n \geq N(\delta)$ is sufficiently large. Using the version of Theorem 4.2 with "$n^{1-2\delta}$" replaced by "$n^{1-c\delta}$" for an arbitrary constant $c > 1$, we can take $\delta$ arbitrarily close to 1.

[13] This is reminiscent of a "direct sum," where Alice and Bob are given multiple instances of a communication problem and have to solve all of them. Direct sums are a fundamental part of communication complexity, but we won't have time to discuss them.

*(i) Alice sends at least $\frac{\delta}{\epsilon^2}\log_2 n$ bits; or*

*(ii) Bob sends at least $n^{1-2\delta}$ bits.*

As with Theorem 4.2, we'll prove Theorem 4.4 for the special case of deterministic protocols. The theorem also holds for randomized protocols with two-sided error [1], and we'll use this stronger result in Theorem 4.8 below. (Our upper bound in Section 3.2 is randomized, so we really want a randomized lower bound.) The proof for randomized protocols argues along the lines of the $\Omega(\sqrt{n})$ lower bound for the standard version of DISJOINTNESS, and is not as hard as the stronger $\Omega(n)$ lower bound (recall the discussion in Lecture #4).

*Proof of Theorem 4.4:* Let $M$ denote the 0-1 matrix corresponding to the $(\frac{1}{\epsilon^2}, n)$-DISJOINTNESS function. Ranging over all subsets of $U$ of size $n$, and for a given such set $S$, over all subsets of $U \setminus S$ of size $\frac{1}{\epsilon^2}$, we see that $M$ has at least $\binom{2n}{n}$ columns that each have at least $\binom{n}{\epsilon^{-2}}$ 1-inputs.

Assume for contraction that there is a communication protocol for $(\frac{1}{\epsilon^2}, n)$-DISJOINTNESS such that neither (i) nor (ii) holds. By the Richness Lemma (Lemma 4.3), there exists a 1-rectangle $A \times B$ where

$$|A| = \binom{n}{\epsilon^{-2}} \cdot 2^{-\delta\frac{\log n}{\epsilon^2}} \geq (\epsilon^2 n)^{\frac{1}{\epsilon^2}} \cdot n^{-\frac{\delta}{\epsilon^2}} = \epsilon^{\frac{2}{\epsilon^2}} n^{\frac{1}{\epsilon^2}(1-\delta)} \tag{2}$$

and

$$|B| = \underbrace{\binom{2n}{n}}_{\approx 2^{2n}/\sqrt{2n}} \cdot 2^{-\delta\frac{\log n}{\epsilon^2}} \cdot 2^{-n^{1-2\delta}} \geq 2^{2n-n^{1-3\delta/2}}, \tag{3}$$

where in (3) we are using that $n$ is sufficiently large.

Since $A \times B$ is a rectangle, $S$ and $T$ are disjoint for every choice of $S \in A$ and $T \in B$. This implies that $\cup_{S \in A} S$ and $\cup_{T \in B} T$ are disjoint sets. Letting

$$s = |\cup_{S \in A} S|,$$

we have

$$|A| \leq \binom{s}{\epsilon^{-2}} \leq s^{1/\epsilon^2}. \tag{4}$$

Combining (2) and (4) implies that

$$s \geq \epsilon^2 n^{1-\delta}.$$

Since every subset $T \in B$ avoids the $s$ elements in $\cup_{S \in A} S$,

$$|B| \leq 2^{2n-s} \leq 2^{2n-\epsilon^2 n^{1-\delta}}. \tag{5}$$

Inequalities (3) and (5) furnish the desired contradiction. ■

The upshot is that, for the goal of proving a communication lower bound of $\Omega(\epsilon^{-2}\log n)$ (for Alice, in a QUERY-DATABASE problem) and a consequent data structure space lower bound of $n^{\Omega(\epsilon^{-2})}$, $(\frac{1}{\epsilon^2}, n)$-DISJOINTNESS is a promising candidate to reduce from.

## 4.3 A Lower Bound for the $(1+\epsilon)$-Approximate Nearest Neighbor Problem

The final major step is to show that $(\frac{1}{\epsilon^2}, n)$-DISJOINTNESS, which is hard by Theorem 4.4, reduces to the QUERY-DATABASE problem for the decision version of the $(1+\epsilon)$-nearest neighbor problem.

### 4.3.1 A Simpler Lower Bound of $n^{\Omega(\epsilon^{-1})}$

We begin with a simpler reduction that leads to a suboptimal but still interesting space lower bound of $n^{\Omega(\epsilon^{-1})}$. In this reduction, we'll reduce from $(\frac{1}{\epsilon}, n)$-DISJOINTNESS rather than $(\frac{1}{\epsilon^2}, n)$-DISJOINTNESS. Alice is given a $\frac{1}{\epsilon}$-set $S$ (from a universe $U$ of size $2n$), which we need to map to a nearest-neighbor query. Bob is given an $n$-set $T \subseteq U$, which we need to map to a point set.

Our first idea is to map the input to $(\frac{1}{\epsilon}, n)$-DISJOINTNESS to a nearest-neighbor query in the $2n$-dimensional hypercube $\{0,1\}^{2n}$. Alice performs the obvious mapping of her input, from the set $S$ to a query point $\mathbf{q}$ that is the characteristic vector of $S$ (which lies in $\{0,1\}^{2n}$). Bob maps his input $T$ to the point set $P = \{\mathbf{e}_i : i \in T\}$, where $\mathbf{e}_i$ denotes the characteristic vector of the singleton $\{i\}$ (i.e., the $i$th standard basis vector).

If the sets $S$ and $T$ are disjoint, then the corresponding query $\mathbf{q}$ has Hamming distance $\frac{1}{\epsilon} + 1$ from every point in the corresponding point set $P$. If $S$ and $T$ are not disjoint, then there exists a point $\mathbf{e}_i \in P$ such that the Hamming distance between $\mathbf{q}$ and $\mathbf{e}_i$ is $\frac{1}{\epsilon} - 1$. Thus, the $(\frac{1}{\epsilon}, n)$-DISJOINTNESS problem reduces to the $(1+\epsilon)$-approximate nearest neighbor problem in the $2n$-dimensional Hamming cube, where $2n$ is also the size of the universe from which the point set is drawn.

We're not done, because extremely high-dimensional nearest neighbor problems are not very interesting. The convention in nearest neighbor problems is to assume that the word size $w$ — recall Section 4.1 — is at least the dimension. When $d$ is at least the size of the universe from which points are drawn, an entire point set can be described using a single word! This means that our reduction so far cannot possibly yield an interesting lower bound in the cell probe model. We fix this issue by applying dimension reduction — just as in our upper bound in Section 3.2 — to the instances produced by the above reduction.

Precisely, we can use the following embedding lemma.

**Lemma 4.5 (Embedding Lemma #1)** *There exists a randomized function $f$ from $\{0,1\}^{2n}$ to $\{0,1\}^d$ with $d = \Theta(\frac{1}{\epsilon^2}\log n)$ and a constant $\alpha > 0$ such that, for every set $P \subseteq \{0,1\}^{2n}$ of $n$ points and query $\mathbf{q} \in \{0,1\}^{2n}$ produced by the reduction above, with probability at least $1 - \frac{1}{n}$:*

*(1) if the nearest-neighbor distance between $\mathbf{q}$ and $P$ is $\frac{1}{\epsilon} - 1$, then the nearest-neighbor distance between $f(\mathbf{q})$ and $f(P)$ is at most $\alpha$;*

*(2) if the nearest-neighbor distance between $\mathbf{q}$ and $P$ is $\frac{1}{\epsilon} + 1$, then the nearest-neighbor distance between $f(\mathbf{q})$ and $f(P)$ is at least $\alpha(1 + h(\epsilon))$, where $h(\epsilon) > 0$ is a constant depending on $\epsilon$ only.*

Lemma 4.5 is an almost immediate consequence of Corollary 3.1 — the map $f$ just takes $d = \Theta(\epsilon^{-2} \log n)$ random inner products with $2n$-bit vectors, where the probability of a "1" is roughly $\epsilon/2$. We used this idea in Section 3.2 for a data structure — here we're using it for a lower bound!

Composing our initial reduction with Lemma 4.5 yields the following.

**Corollary 4.6** *Every randomized asymmetric communication lower bound for* $(\frac{1}{\epsilon}, n)$-Disjointness *carries over to the* Query-Database *problem for the* $(1+\epsilon)$*-approximate nearest neighbor problem in* $d = \Omega(\epsilon^{-2} \log n)$ *dimensions.*

*Proof:* To recap our ideas, the reduction works as follows. Given inputs to $(\frac{1}{\epsilon}, n)$-Disjointness, Alice interprets her input as a query and Bob interprets his input as a point set (both in $\{0,1\}^{2n}$) as described at the beginning of the section. They use shared randomness to choose the function $f$ of Lemma 4.5 and use it to map their inputs to $\{0,1\}^d$ with $d = \Theta(\epsilon^{-2} \log n)$. They run the assumed protocol for the Query-Database problem for the $(1+\epsilon)$-approximate nearest neighbor problem in $d$ dimensions. Provided the hidden constant in the definition of $d$ is sufficiently large, correctness (with high probability) is guaranteed by Lemma 4.5. (Think of Lemma 4.5 as being invoked with a parameter $\epsilon'$ satisfying $h(\epsilon') = \epsilon$.) The amount of communication used by the $(\frac{1}{\epsilon}, n)$-Disjointness protocol is identical to that of the Query-Database protocol.[14] ■

Following the arguments of Section 4.2.1 translates our asymmetric communication complexity lower bound (via Lemma 4.1) to a data structure space lower bound.

**Corollary 4.7** *Every data structure for the decision version of the* $(1+\epsilon)$*-approximate nearest neighbors problem with query time* $t = \Theta(1)$ *and word size* $w = O(n^{1-\delta})$ *for constant* $\delta > 0$ *uses space* $s = n^{\Omega(\epsilon^{-1})}$.

*Proof:* Since $tw = O(n^{1-\delta})$, in the induced communication protocol for the Query-Database problem (and hence $(\frac{1}{\epsilon}, n)$-Disjointness, via Corollary 4.6), Bob sends a sublinear number of bits. Theorem 4.4 then implies that Alice sends at least $\Omega(\epsilon^{-1} \log n)$ bits, and so (by Lemma 4.1) we have $t \log_2 s = \Omega(\epsilon^{-1} \log n)$. Since $t = O(1)$, this implies that $s = n^{\Omega(\epsilon^{-1})}$. ■

### 4.3.2 The $n^{\Omega(\epsilon^{-2})}$ Lower Bound

The culmination of this lecture is the following.

**Theorem 4.8** *Every data structure for the decision version of the* $(1+\epsilon)$*-approximate nearest neighbors problem with query time* $t = \Theta(1)$ *and word size* $w = O(n^{1-\delta})$ *for constant* $\delta > 0$ *uses space* $s = n^{\Omega(\epsilon^{-2})}$.

[14] The $(\frac{1}{\epsilon}, n)$-Disjointness protocol is randomized with two-sided error even if the Query-Database protocol is deterministic. This highlights our need for Theorem 4.4 in its full generality.

The proof is a refinement of the embedding arguments we used to prove Corollary 4.7. In that proof, the reduction structure was

$$S, T \subseteq U \mapsto \{0,1\}^{2n} \mapsto \{0,1\}^{d},$$

with inputs $S, T$ of $(\frac{1}{\epsilon}, n)$-Disjointness mapped to the $2n$-dimensional Hamming cube and then to the $d$-dimensional Hamming cube, with $d = \Theta(\epsilon^{-2} \log n)$.

The new plan is

$$S, T \subseteq U \mapsto (\mathcal{R}^{2n}, \ell_2) \mapsto (\mathcal{R}^{D}, \ell_1) \mapsto \{0,1\}^{D'} \mapsto \{0,1\}^{d},$$

where $d = \Theta(\epsilon^{-2} \log n)$ as before, and $D, D'$ can be very large. Thus we map inputs $S, T$ of $(\frac{1}{\epsilon^2}, n)$-Disjointness to $2n$-dimensional Euclidean space (with the $\ell_2$ norm), which we then map (preserving distances) to high-dimensional space with the $\ell_1$ norm, then to the high-dimensional Hamming cube, and finally to the $\Theta(\epsilon^{-2} \log n)$-dimensional Hamming cube as before (via Lemma 4.5). The key insight is that switching the initial embedding from the high-dimensional hypercube to high-dimensional Euclidean space achieves a nearest neighbor gap of $1 \pm \epsilon$ even when Alice begins with a $\frac{1}{\epsilon^2}$-set; the rest of the argument uses standard (if non-trivial) techniques to eventually get back to a hypercube of reasonable dimension.

To add detail to the important first step, consider inputs $S, T$ to $(\frac{1}{\epsilon^2}, n)$-Disjointness. Alice maps her set $S$ to a query vector $\mathbf{q}$ that is $\epsilon$ times the characteristic vector of $S$, which we interpret as a point in $2n$-dimensional Euclidean space. Bob maps his input $T$ to the point set $P = \{\mathbf{e}_i : i \in T\}$, again in $2n$-dimensional Euclidean space, where $\mathbf{e}_i$ denotes the $i$th standard basis vector.

First, suppose that $S$ and $T$ are disjoint. Then, the $\ell_2$ distance between Alice's query $\mathbf{q}$ and each point $\mathbf{e}_i \in P$ is

$$\sqrt{1 + \tfrac{1}{\epsilon^2} \cdot \epsilon^2} = \sqrt{2}.$$

If $S$ and $T$ are not disjoint, then there exists a point $\mathbf{e}_i \in P$ such that the $\ell_2$ distance between $\mathbf{q}$ and $\mathbf{e}_i$ is:

$$\sqrt{(1-\epsilon)^2 + \left(\tfrac{1}{\epsilon^2} - 1\right)\epsilon^2} = \sqrt{2 - 2\epsilon} \leq \sqrt{2}\left(1 - \tfrac{\epsilon}{2}\right).$$

Thus, as promised, switching to the $\ell_2$ norm — and tweaking Alice's query – allows us to get a $1 \pm \Theta(\epsilon)$ gap in nearest-neighbor distance between the "yes" and "no" instances of $(\frac{1}{\epsilon^2}, n)$-Disjointness. This immediately yields (via Theorem 4.4, following the proof of Corollary 4.7) lower bounds for the $(1+\epsilon)$-approximate nearest neighbor problem in high-dimensional Euclidean space in the cell-probe model. We can extend these lower bounds to the hypercube through the following embedding lemma.

**Lemma 4.9 (Embedding Lemma #2)** *For every $\delta > 0$ there exists a randomized function $f$ from $\mathcal{R}^{2n}$ to $\{0,1\}^D$ (with possibly large $D = D(\delta)$) such that, for every set $P \subseteq \{0,1\}^{2n}$ of $n$ points and query $\mathbf{q} \in \{0,1\}^{2n}$ produced by the reduction above, with probability at least $1 - \frac{1}{n}$,*

$$\ell_H(f(\mathbf{p}), f(\mathbf{q})) \in (1 \pm \delta) \cdot \ell_2(\mathbf{p}, \mathbf{q})$$

*for every $\mathbf{p} \in P$.*

Thus Lemma 4.9 says that one can re-represent a query $\mathbf{q}$ and a set $P$ of $n$ points in $\mathcal{R}^{2n}$ in a high-dimensional hypercube so that the nearest-neighbor distance — $\ell_2$ distance in the domain, Hamming distance in the range — is approximately preserved, with the approximation factor tending to 1 as the number $D$ of dimensions tends to infinity. The Exercises outline the proof, which combines two standard facts from the theory of metric embeddings:

1. "$L_2$ embeds isometrically into $L_1$." For every $\delta > 0$ and dimension $D$ there exists a randomized function $f$ from $\mathcal{R}^D$ to $\mathcal{R}^{D'}$, where $D'$ can depend on $D$ and $\delta$, such that, for every set $P \subseteq \mathcal{R}^D$ of $n$ points, with probability at least $1 - \frac{1}{n}$,

$$\|f(\mathbf{p}) - f(\mathbf{p}')\|_1 \in (1 \pm \delta) \cdot \|\mathbf{p} - \mathbf{p}'\|_2$$

for all $\mathbf{p}, \mathbf{p}' \in P$.

2. "$L_1$ embeds isometrically into the (scaled) Hamming cube." For every $\delta > 0$, there exists constants $M = M(\delta)$ and $D'' = D''(D', \delta)$ and a function $g : \mathcal{R}^{D'} \to \{0,1\}^{D''}$ such that, for every set $P \subseteq \mathcal{R}^{D'}$,

$$d_H(g(\mathbf{p}, \mathbf{p}')) = M \cdot \|\mathbf{p} - \mathbf{p}'\|_1 \pm \delta$$

for every $\mathbf{p}, \mathbf{p}' \in P$.

With Lemma 4.9 in hand, we can prove Theorem 4.8 by following the argument in Corollary 4.7.

*Proof of Theorem 4.8:* By Lemmas 4.5 and 4.9, Alice and Bob can use a communication protocol that solves the Query-Database problem for the decision version of $(1+\epsilon)$-nearest neighbors to solve the $(\frac{1}{\epsilon^2}, n)$-Disjointness problem, with no additional communication (only shared randomness, to pick the random functions in Lemmas 4.5 and 4.9).[15] Thus, the (randomized) asymmetric communication lower bound for the latter problem applies also to the former problem.

Since $tw = O(n^{1-\delta})$, in the induced communication protocol for the Query-Database problem (and hence $(\frac{1}{\epsilon^2}, n)$-Disjointness), Bob sends a sublinear number of bits. Theorem 4.4 then implies that Alice sends at least $\Omega(\epsilon^{-2} \log_2 n)$ bits, and so (by Lemma 4.1) we have $t \log_2 s = \Omega(\epsilon^{-2} \log_2 n)$. Since $t = O(1)$, this implies that $s = n^{\Omega(\epsilon^{-2})}$. ■

# References

[1] A. Andoni, P. Indyk, and M. Pătraşcu. On the optimality of the dimensionality reduction method. In *Proceedings of the 47th Symposium on Foundations of Computer Science (FOCS)*, pages 449–458, 2006.

[15] Strictly speaking, we're using a generalization of Lemma 4.5 (with the same proof) where the query and point set can lie in a hypercube of arbitrarily large dimension, not just $2n$.

[2] A. Chakrabarti and O. Regev. An optimal lower bound on the communication complexity of gap-hamming-distance. *SIAM Journal on Computing*, 41(5):1299–1317, 2012.

[3] A. Fiat and M. Naor. Implicit O(1) probe search. *SIAM Journal on Computing*, 22(1):1–10, 1993.

[4] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with 0(1) worst case access time. *Journal of the ACM*, 31(3):538–544, 1984.

[5] A. Gabizon and A. Hassidim. Derandomizing algorithms on product distributions and other applications of order-based extraction. In *Proceedings of Innovations in Computer Science (ICS)*, pages 397–405, 2010.

[6] A. Gabizon and R. Shaltiel. Increasing the output length of zero-error dispersers. *Random Structures & Algorithms*, 40(1):74–104, 2012.

[7] P. Indyk. Nearest-neighbor searching in high dimensions. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 39, pages 877–892. CRC Press, 2004. Second Edition.

[8] P. B. Miltersen. Cell probe complexity — a survey. Invited paper at Workshop on Advances in Data Structures, 1999.

[9] P. B. Miltersen, N. Nisan, S. Safra, and A. Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences*, 57(1):37–49, 1998.

[10] M. Pătraşcu. *Lower Bound Techniques for Data Structures.* PhD thesis, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 2008.

[11] A. A. Sherstov. The communication complexity of gap hamming distance. *Theory of Computing*, 8(8):197–208, 2012.

[12] T. Vidick. A concentration inequality for the overlap of a vector on a large set, with application to the communication complexity of the gap-hamming-distance problem. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:51, 2011.

[13] A. C.-C. Yao. Some complexity questions related to distributive computing (preliminary report). In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing (STOC)*, pages 209–213, 1979.

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