# CS369E: Communication Complexity (for Algorithm Designers) Lecture #3: Lower Bounds for Compressive Sensing*

Tim Roughgarden†

January 22, 2015

## 1 An Appetizer: Randomized Communication Complexity of the Equality Function

We begin with an appetizer before starting the lecture proper — an example that demonstrates that randomized one-way communication protocols can sometimes exhibit surprising power.

It won't surprise you that the EQUALITY function — with $f(\mathbf{x}, \mathbf{y}) = 1$ if and only if $\mathbf{x} = \mathbf{y}$ — is a central problem in communication complexity. It's easy to prove, by the Pigeonhole Principle, that its deterministic one-way communication complexity is $n$, where $n$ is the length of the inputs $\mathbf{x}$ and $\mathbf{y}$.[1] What about its randomized communication complexity? Recall from last lecture that by default, our randomized protocols can use public coins[2] and can have two-sided error $\epsilon$, where $\epsilon$ is any constant less than $\frac{1}{2}$.

**Theorem 1.1 ([8])** *The (public-coin) randomized one-way communication complexity of* EQUALITY *is* $O(1)$.

Thus, the randomized communication complexity of a problem can be radically smaller than its deterministic communication complexity. A similar statement follows from our upper and lower bound results for estimating the frequency moments $F_0$ and $F_2$ using small-space streaming algorithms, but Theorem 1.1 illustrates this point in a starker and clearer way.



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

[1] We'll see later that this lower bound applies to general deterministic protocols, not just to one-way protocols.

[2] Recall the public-coin model: when Alice and Bob show up there is already an infinite stream of random bits written on a blackboard, which both of them can see. Using shared randomness does not count toward the communication cost of the protocol.

Theorem 1.1 provides a cautionary tale: sometimes we expect a problem to be hard for a class of protocols and are proved wrong by a clever protocol; other times, clever protocols don't provide non-trivial solutions to a problem but still make proving strong lower bounds technically difficult. Theorem 1.1 also suggests that, if we want to prove strong communication lower bounds for randomized protocols via a reduction, there might not be too many natural problems out there to reduce from.[3]

*Proof of Theorem 1.1:* The protocol is as follows.

1. Alice and Bob interpret the first $2n$ public coins as random strings $\mathbf{r}_1, \mathbf{r}_2 \in \{0,1\}^n$. This requires no communication.

2. Alice sends the two random inner products $\langle \mathbf{x}, \mathbf{r}_1 \rangle$ mod 2 and $\langle \mathbf{x}, \mathbf{r}_2 \rangle$ mod 2 to Bob. This requires two bits of communication.

3. Bob reports "1" if and only if his random inner products match those of Alice: $\langle \mathbf{y}, \mathbf{r}_i \rangle = \langle \mathbf{x}, \mathbf{r}_i \rangle$ mod 2 for $i = 1, 2$. Note that Bob has all of the information needed to perform this computation.

We claim that the error of this protocol is at most 25% on every input. The protocol's error is one-sided: when $\mathbf{x} = \mathbf{y}$ the protocol always accepts, so there are no false negatives. Suppose that $\mathbf{x} \neq \mathbf{y}$. We use the Principle of Deferred Decisions to argue that, for each $i = 1, 2$, the inner products $\langle \mathbf{y}, \mathbf{r}_i \rangle$ and $\langle \mathbf{x}, \mathbf{r}_i \rangle$ are different (mod 2) with probability exactly 50%. To see this, pick an index $i$ where $x_i \neq y_i$ and condition on all of the bits of a random string except for the $i$th one. Let $a$ and $b$ denote the values of the inner products-so-far of $\mathbf{x}$ and $\mathbf{y}$ (modulo 2) with the random string. If the $i$th random bit is a 0, then the final inner products are also $a$ and $b$. If the $i$th random bit is a 1, then one inner product stays the same while the other flips its value (since exactly one of $x_i, y_i$ is a 1). Thus, whether $a = b$ or $a \neq b$, exactly one of the two random bit values (50% probability) results in the final two inner products having different values (modulo 2). The probability that two unequal strings have equal inner products (modulo 2) in two independent experiments is 25%. ■

The proof of Theorem 1.1 gives a 2-bit protocol with (1-sided) error 25%. As usual, executing many parallel copies of the protocol reduces the error to an arbitrarily small constant, with a constant blow-up in the communication complexity.

The protocol used to prove Theorem 1.1 makes crucial use of public coins. We'll see later that the private-coin 1-way randomized communication complexity is $\Theta(\log n)$, which is worse than public-coin protocols but still radically better than deterministic protocols. More generally, next lecture we'll prove Newman's theorem, which states that the private-coin communication complexity of a problem is at most $O(\log n)$ more than its public-coin communication complexity.

[3]Recall our discussion about GAP-HAMMING last lecture: for the problem to be hard, it is important to choose the midpoint $t$ to be $\frac{n}{2}$. With $t$ too close to 0 or $n$, the problem is a special case of EQUALITY and is therefore easy for randomized protocols.

The protocol in the proof of Theorem 1.1 effectively gives each of the two strings $\mathbf{x}, \mathbf{y}$ a 2-bit "sketch" or "fingerprint" such that the property of distinctness is approximately preserved. Clearly, this is the same basic idea as hashing. This is a useful idea in both theory and practice, and we'll use it again shortly.

**Remark 1.2** The computational model studied in communication complexity is potentially very powerful — for example, Alice and Bob have unlimited computational power — and the primary point of the model is to prove lower bounds. Thus, whenever you see an *upper bound* result in communication complexity, like Theorem 1.1, it's worth asking what the point of the result is. In many cases, a positive result is really more of a "negative negative result," intended to prove the tightness of a lower bound rather than offer a practical solution to a real problem. In other cases, the main point is demonstrate separations between different notions of communication complexity or between different problems. For example, Theorem 1.1 shows that the one-way deterministic and randomized communication complexity of a problem can be radically different, even if we insist on one-sided error. It also shows that the randomized communication complexity of Equality is very different than that of the problems we studied last lecture: Disjointness, Index, and Gap-Hamming.

Theorem 1.1 also uses a quite reasonable protocol, which is not far from a practical solution to probabilistic equality-testing. In some applications, the public coins can be replaced by a hash function that is published in advance; in other applications, one party can choose a random hash function that can be specified with a reasonable number of bits and communicate it to other parties.

# 2 Sparse Recovery

## 2.1 The Basic Setup

The field of sparse recovery has been a ridiculously hot area for the past ten years, in applied mathematics, machine learning, and theoretical computer science. We'll study sparse recovery in the standard setup of "compressive sensing" (also called "compressed sensing"). There is an unknown "signal" — i.e., a real-valued vector $\mathbf{x} \in \mathbb{R}^n$ — that we want to learn. The bad news is that we're only allowed to access the signal through "linear measurements;" the good news is that we have the freedom to choose whatever measurements we want. Mathematically, we want to design a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, with $m$ as small as possible, such that we can recover the unknown signal $\mathbf{x}$ from the linear measurements $\mathbf{Ax}$ (whatever $\mathbf{x}$ may be).

As currently stated, this is a boring problem. It is clear that $n$ measurements are sufficient – just take $\mathbf{A} = I$, or any other invertible $n \times n$ matrix. It is also clear that $n$ measurements are necessary: if $m < n$, then there is a entire subspace of dimension $n - m$ of vectors that have image $\mathbf{Ax}$ under $\mathbf{A}$, and we have no way to know which one of them is the actual unknown signal.

The problem becomes interesting when we also assume that the unknown signal $\mathbf{x}$ is

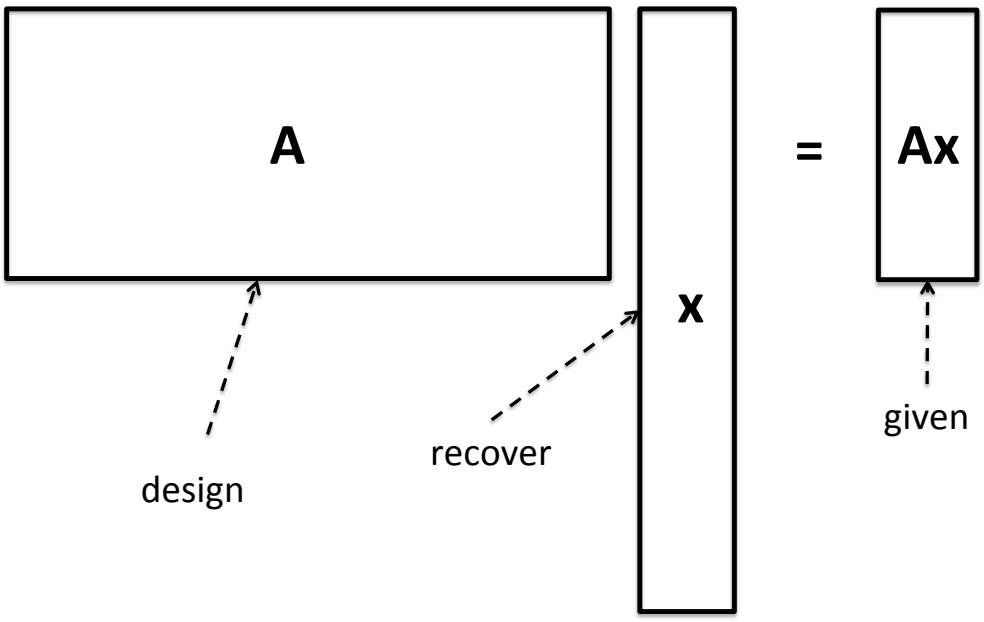


Figure 1: The basic compressive sensing setup. The goal is to design a matrix $\mathbf{A}$ such that an unknown sparse signal $\mathbf{x}$ can be recovered from the linear measurements $\mathbf{Ax}$.

"sparse," in senses we define shortly. The hope is that under the additional promise that $\mathbf{x}$ is sparse, we can get away with much fewer than $n$ measurements (Figure 1).

## 2.2 A Toy Version

To develop intuition for this problem, let's explore a toy version. Suppose you are promised that $\mathbf{x}$ is a 0-1 vector with exactly $k$ 1's (and hence $n-k$ 0's). Throughout this lecture, $k$ is the parameter that measures the sparsity of the unknown signal $\mathbf{x}$ — it could be anything, but you might want to keep $k \approx \sqrt{n}$ in mind as a canonical parameter value. Let $X$ denote the set of all such $k$-sparse 0-1 vectors, and note that $|X| = \binom{n}{k}$.

Here's a solution to the sparse recovery problem under that guarantee that $\mathbf{x} \in X$. Take $m = 3\log_2 |X|$, and choose each of the $m$ rows of the sensing matrix $\mathbf{A}$ independently and uniformly at random from $\{0,1\}^n$. By the fingerprinting argument used in our randomized protocol for Equality (Theorem 1.1), for fixed distinct $\mathbf{x}, \mathbf{x}' \in X$, we have

$$\mathbf{Pr}_{\mathbf{A}}[\mathbf{Ax} = \mathbf{Ax}' \bmod 2] = \frac{1}{2^m} = \frac{1}{|X|^3}.$$

Of course, the probability that $\mathbf{Ax} = \mathbf{Ax}'$ (not modulo 2) is only less. Taking a Union Bound over the at most $|X|^2$ different distinct pairs $\mathbf{x}, x' \in X$, we have

$$\mathbf{Pr}_{\mathbf{A}}[\text{there exists } \mathbf{x} \neq \mathbf{x}' \text{ s.t. } \mathbf{Ax} = \mathbf{Ax}'] \leq \frac{1}{|X|}.$$

Thus, there is a matrix $\mathbf{A}$ that maps all $\mathbf{x} \in X$ to distinct $m$-vectors. (Indeed, a random $\mathbf{A}$ works with high probability.) Thus, given $\mathbf{Ax}$, one can recover $\mathbf{x}$ — if nothing else, one can just compute $\mathbf{Ax}'$ for every $\mathbf{x}' \in X$ until a match is found.[4]

The point is that $m = \Theta(\log |X|)$ measurements are sufficient to recover exactly $k$-sparse 0-1 vectors. Recalling that $|X| = \binom{n}{k}$ and that

$$\left(\frac{n}{k}\right)^k \underbrace{\leq}_{\text{easy}} \binom{n}{k} \underbrace{\leq}_{\text{Stirling's approx.}} \left(\frac{en}{k}\right)^k,$$

we see that $m = \Theta(k \log \frac{n}{k})$ rows suffice.[5]

This exercise shows that, at least for the special case of exactly sparse 0-1 signals, we can indeed achieve recovery with far fewer than $n$ measurements. For example, if $k \approx \sqrt{n}$, we need only $O(\sqrt{n} \log n)$ measurements.

Now that we have a proof of concept that recovering an unknown sparse vector is an interesting problem, we'd like to do better in two senses. First, we want to move beyond the toy version of the problem to the "real" version of the problem. Second, we have to wonder whether even fewer measurements suffice.

## 2.3 Motivating Applications

To motivate the real version of the problem, we mention a couple of canonical applications of compressive sensing. One buzzword you can look up and read more about is the "single-pixel camera." The standard approach to taking pictures is to first take a high-resolution picture in the "standard basis" — e.g., a light intensity for each pixel — and then to compress the picture later (via software). Because real-world images are typically sparse in a suitable basis, they can be compressed a lot. The compressive sensing approach asks, then why not just capture the image directly in a compressed form — in a representation where its sparsity shines through? For example, one can store random linear combinations of light intensities (implemented via suitable mirrors) rather than the light intensities themselves. This idea leads to a reduction in the number of pixels needed to capture an image at a given resolution. Another application of compressive sensing is in MRI. Here, decreasing the number of measurements decreases the time necessary for a scan. Since a patient needs to stay motionless during a scan — in some cases, not even breathing — shorter scan times can be a pretty big deal.

## 2.4 The Real Problem

If the unknown signal $\mathbf{x}$ is an image, say, there's no way it's an exactly sparse 0-1 vector. We need to consider more general unknown signals $\mathbf{x}$ that are real-valued and only "ap-

[4] We won't focus on computational efficiency in this lecture, but positive results in compressive sensing generally also have computationally efficient recovery algorithms. Our lower bounds will hold even for recovery algorithms with unbounded computational power.

[5] If you read through the compressive sensing literature, you'll be plagued by ubiquitous "$k \log \frac{n}{k}$" terms — remember this is just $\approx \log \binom{n}{k}$.

proximately sparse." To measure approximate sparsity, with respect to a choice of $k$, we define the *residual* $\text{res}(\mathbf{x})$ of a vector $\mathbf{x} \in \mathbb{R}^n$ as the contribution to $\mathbf{x}$'s $\ell_1$ norm by its $n-k$ coordinates with smallest magnitudes. Recall that the $\ell_1$ norm is just $\|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i|$. If we imagine sorting the coordinates by $|x_i|$, then the residual of $\mathbf{x}$ is just the sum of the $|x_i|$'s of the final $n-k$ terms. If $\mathbf{x}$ has exactly $k$-sparse, it has at least $n-k$ zeros and hence $\text{res}(\mathbf{x}) = 0$.

The goal is to design a sensing matrix $\mathbf{A}$ with a small number $m$ of rows such that an unknown approximately sparse vector $\mathbf{x}$ can be recovered from $\mathbf{Ax}$. Or rather, given that $\mathbf{x}$ is only approximately sparse, we want to recover a close approximation of $\mathbf{x}$.

The formal guarantee we'll seek for the matrix $\mathbf{A}$ is the following: for every $\mathbf{x} \in \mathbb{R}^n$, we can compute from $\mathbf{Ax}$ a vector $\mathbf{x}'$ such that

$$\|\mathbf{x}' - \mathbf{x}\|_1 \leq c \cdot \text{res}(\mathbf{x}). \tag{1}$$

Here $c$ is a constant, like 2. The guarantee (1) is very compelling. First, if $\mathbf{x}$ is exactly $k$-sparse, then $\text{res}(\mathbf{x}) = 0$ and so (1) demands exact recovery of $\mathbf{x}$. The guarantee is parameterized by how close $\mathbf{x}$ is to being sparse — the recovered vector $\mathbf{x}'$ should lie in a ball (in the $\ell_1$ norm) around $\mathbf{x}$, and the further $\mathbf{x}$ is from being $k$-sparse, the bigger this ball is. Intuitively, the radius of this ball has to depend on something like $\text{res}(\mathbf{x})$. For example, suppose that $\mathbf{x}'$ is exactly $k$-sparse (with $\text{res}(\mathbf{x}) = 0$). The guarantee (1) forces the algorithm to return $\mathbf{x}'$ for every unknown signal $\mathbf{x}$ with $\mathbf{Ax} = \mathbf{Ax}'$. Recall that when $m < n$, there is an $(n-m)$-dimensional subspace of such signals $\mathbf{x}$. In the extreme case where there is such an $\mathbf{x}$ with $\mathbf{x} - \text{res}(\mathbf{x}) = \mathbf{x}'$ – i.e., where $\mathbf{x}$ is $\mathbf{x}'$ with a little noise added to its zero coordinates — the recovery algorithm is forced to return a solution $\mathbf{x}'$ with $\|\mathbf{x}' - \mathbf{x}\|_1 = \text{res}(\mathbf{x})$.

Now that we've defined the real version of the problem, is there an interesting solution? Happily, the real version can be solved as well as the toy version.

**Fact 2.1 ([3, 5])** *With high probability, a random $m \times n$ matrix $\mathbf{A}$ with $\Theta(k \log \frac{n}{k})$ rows admits a recovery algorithm that satisfies the guarantee in* (1) *for every $\mathbf{x} \in \mathbb{R}^n$.*

Fact 2.1 is non-trivial and well outside the scope of this course. The fact is true for several different distributions over matrices, including the case where each matrix entry is an independent standard Gaussian. Also, there are computationally efficient recovery algorithms that achieve the guarantee.[6]

# 3 A Lower Bound for Sparse Recovery

## 3.1 Context

At the end of Section 2.2, when we asked "can we do better?," we meant this in two senses. First, can we extend the positive results for the toy problem to a more general problem?

[6] See e.g. [7, Chapter 4] for an introduction to such positive results about sparse recovery. Lecture #9 of the instructor's CS264 course also gives a brief overview.

Fact 2.1 provides a resounding affirmative answer. Second, can we get away with an even smaller value of $m$ — even fewer measurements? In this section we prove a relatively recent (2011) result of Do Ba et al. [4], who showed that the answer is "no."[7] Amazingly, they proved this fundamental result via a reduction to a lower bound in communication complexity (for INDEX).

**Theorem 3.1 ([4])** *If an $m \times n$ matrix $\mathbf{A}$ admits a recovery algorithm $R$ that, for every $\mathbf{x} \in \mathbb{R}^n$, computes from $\mathbf{Ax}$ a vector $\mathbf{x}'$ that satisfies (1), then $m = \Omega(k \log \frac{n}{k})$.*

The lower bound is information-theoretic, and therefore applies also to recovery algorithms with unlimited computational power.

Note that there is an easy lower bound of $m \geq k$.[8] Theorem 3.1 offers a non-trivial improvement when $k = o(n)$; in this case, we're asking whether or not it's possible to shave off the log factor in Fact 2.1. Given how fundamental the problem is, and how much a non-trivial improvement could potentially matter in practice, this question is well worth asking.

## 3.2 Proof of Theorem 3.1: First Attempt

Recall that we first proved our upper bound of $m = O(k \log \frac{n}{k})$ in the toy setting of Section 2.2, and then stated in Fact 2.1 that it can be extended to the general version of the problem. Let's first try to prove a matching lower bound on $m$ that applies even in the toy setting. Recall that $X$ denotes the set of all 0-1 vectors that have exactly $k$ 1's, and that $|X| = \binom{n}{k}$.

For vectors $\mathbf{x} \in X$, the guarantee (1) demands exact recovery. Thus, the sensing matrix $\mathbf{A}$ that we pick has to satisfy $\mathbf{Ax} \neq \mathbf{Ax}'$ for all distinct $x, x' \in X$. That is, $\mathbf{Ax}$ encodes $\mathbf{x}$ for all $\mathbf{x} \in X$. But $X$ has $\binom{n}{k}$ members, so the worst-case encoding length of $\mathbf{Ax}$ has to be at least $\log_2 \binom{n}{k} = \Theta(k \log \frac{n}{k})$. So are we done?

The issue is that we want a lower bound on the number of rows $m$ of $\mathbf{A}$, not on the worst-case length of $\mathbf{Ax}$ in bits. What is the relationship between these two quantities? Note that, even if $\mathbf{A}$ is a 0-1 matrix, then each entry of $\mathbf{Ax}$ is generally of magnitude $\Theta(k)$, requiring $\Theta(\log k)$ bits to write down. For example, when $k$ is polynomial in $n$ (like our running choice $k = \sqrt{n}$), then $\mathbf{Ax}$ generally requires $\Omega(m \log n)$ bits to describe, even when $\mathbf{A}$ is a 0-1 matrix. Thus our lower bound of $\Theta(k \log \frac{n}{k})$ on the length of $\mathbf{Ax}$ does not yield a lower bound on $m$ better than the trivial lower bound of $k$.[9]

The argument above was doomed to fail. The reason is that, if you only care about recovering exactly $k$-sparse vectors $\mathbf{x}$ — rather than the more general and robust guarantee

[7] Such a lower bound was previously known for various special cases — for particular classes of matrices, for particular families of recovery algorithms, etc.

[8] For example, consider just the subset of vectors $\mathbf{x}$ that are zero in the final $n - k$ coordinates. The guarantee (1) demands exact recovery of all such vectors. Thus, we're back to the boring version of the problem mentioned at the beginning of the lecture, and $\mathbf{A}$ has to have rank at least $k$.

[9] This argument does imply that $m = \Omega(k \log \frac{n}{k})$ is we only we report $\mathbf{Ax}$ modulo 2 (or some other constant), since in this case the length of $\mathbf{Ax}$ is $\Theta(m)$.

in (1) — then $m = 2k$ suffices! One proof is via "Prony's Method," which uses the fact that a $k$-sparse vector $\mathbf{x}$ can be recovered exactly from its first $2k$ Fourier coefficients (see e.g. [7]).[10] Our argument above only invoked the requirement (1) for $k$-sparse vectors $\mathbf{x}$, and such an argument cannot prove a lower bound of the form $m = \Omega(k \log \frac{n}{k})$.

The first take-away from this exercise is that, to prove the lower bound that we want, we need to use the fact that the matrix $\mathbf{A}$ satisfies the guarantee (1) also for non-$k$-sparse vectors $\mathbf{x}$. The second take-away is that we need a smarter argument — a straightforward application of the Pigeonhole Principle is not going to cut it.

## 3.3 Proof of Theorem 3.1

### 3.3.1 A Communication Complexity Perspective

We can interpret the failed proof attempt in Section 3.2 as an attempted reduction from a "promise version" of INDEX. Recall that in this communication problem, Alice has an input $\mathbf{x} \in \{0,1\}^n$, Bob has an index $i \in \{1, 2, \ldots, n\}$, specified using $\approx \log_2 n$ bits, and the goal is to compute $x_i$ using a one-way protocol. We showed last lecture that the deterministic communication complexity of this problem is $n$ (via an easy counting argument) and its randomized communication complexity is $\Omega(n)$ (via a harder counting argument).

The previous proof attempt can be rephrased as follows. Consider a matrix $\mathbf{A}$ that permits exact recovery for all $\mathbf{x} \in X$. This induces a one-way protocol for solving INDEX whenever Alice's input $\mathbf{x}$ lies in $X$ — Alice simply sends $\mathbf{Ax}$ to Bob, Bob recovers $\mathbf{x}$, and Bob can then solve the problem, whatever his index $i$ might be. The communication cost of this protocol is exactly the length of $\mathbf{Ax}$, in bits. The deterministic one-way communication complexity of this promise version of INDEX is $k \log \frac{n}{k}$, by the usual counting argument, so this lower bound applies to the length of $\mathbf{Ax}$.

### 3.3.2 The Plan

How can we push this idea further? We begin by assuming the existence of an $m \times n$ matrix $\mathbf{A}$, a recovery algorithm $R$, and a constant $c \geq 1$ such that, for every $\mathbf{x} \in \mathbb{R}^n$, $R$ computes from $\mathbf{Ax}$ a vector $\mathbf{x}' \in \mathbb{R}^n$ that satisfies

$$\|\mathbf{x}' - \mathbf{x}\|_1 \leq c \cdot \text{res}(\mathbf{x}). \tag{2}$$

Our goal is to show that if $m << k \log \frac{n}{k}$, then we can solve INDEX with sublinear communication.

For simplicity, we assume that the recovery algorithm $R$ is deterministic. The lower bound continues to hold for randomized recovery algorithms that have success probability at least $\frac{2}{3}$, but the proof requires more work; see Section 3.4.

[10] This method uses a sensing matrix $\mathbf{A}$ for which $\mathbf{Ax}$ will generally have $\Omega(m \log n) = \Omega(k \log n)$ bits, so this does not contradict out lower bound on the necessary length of $\mathbf{Ax}$.

### 3.3.3 An Aside on Bit Complexity

We can assume that the sensing matrix $\mathbf{A}$ has reasonable bit complexity. Roughly: (i) we can assume without loss of generality that $\mathbf{A}$ has orthonormal rows (by a change of basis argument); (ii) dropping all but the $O(\log n)$ highest-order bits of every entry has negligible effect on the recovery algorithm $R$. We leave the details to the Exercises.

When every entry of the matrix $\mathbf{A}$ and of a vector $\mathbf{x}$ can be described in $O(\log n)$ bits — equivalently, by scaling, is a polynomially-bounded integer — the same is true of $\mathbf{Ax}$. In this case, $\mathbf{Ax}$ has length $O(m \log n)$.

### 3.3.4 Redefining $X$

It will be useful later to redefine the set $X$. Previously, $X$ was all 0-1 $n$-vectors with exactly $k$ 1's. Now it will be a subset of such vectors, subject to the constraint that

$$d_H(\mathbf{x}, \mathbf{x}') \geq .1k$$

for every distinct pair $\mathbf{x}, \mathbf{x}'$ of vectors in the set. Recall the $d_H(\cdot, \cdot)$ denotes the Hamming distance between two vectors — the number of coordinates in which they differ. Two distinct 0-1 vectors with $k$ 1's each have Hamming distance between 2 and $2k$, so we're restricting to a subset of such vectors that have mostly disjoint supports. The following lemma says that there exist sets of such vectors with size not too much smaller than the number of all 0-1 vectors with $k$ 1's; we leave the proof to the Exercises.

**Lemma 3.2** *Suppose $k \leq .01n$. There is a set $X$ of 0-1 $n$-bits vectors such that each $\mathbf{x} \in X$ has $k$ 1s, each distinct $\mathbf{x}, \mathbf{x}' \in X$ satisfy $d_H(\mathbf{x}, \mathbf{x}') \geq .1k$, and $\log_2 |X| = \Omega(k \log \frac{n}{k})$.*

Lemma 3.2 is reminiscent of the fact that there are large error-correcting codes with large distance. One way to prove Lemma 3.2 is via the probabilistic method — by showing that a suitable randomized experiment yields a set with the desired size with positive probability.

Intuitively, our proof attempt in Section 3.2 used that, because each $\mathbf{x} \in X$ is exactly $k$-sparse, it can be recovered exactly from $\mathbf{Ax}$ and thus there is no way to get confused between distinct elements of $X$ from their images. In the following proof, we'll instead need to recover "noisy" versions of the $\mathbf{x}$'s — recall that our proof cannot rely only on the fact that the matrix $\mathbf{A}$ performs exact recovery of exactly sparse vectors. This means we might get confused between two different 0-1 vectors $\mathbf{x}, \mathbf{x}'$ that have small (but non-zero) Hamming distance. The above redefinition of $X$, which is essentially costless by Lemma 3.2, fixes the issue by restricting to vectors that all look very different from one another.

### 3.3.5 The Reduction

To obtain a lower bound of $m = \Omega(k \log \frac{n}{k})$ rather than $m = \Omega(k)$, we need a more sophisticated reduction than in Section 3.2.[11] The parameters offer some strong clues as to what

[11] We assume from now on that $k = o(n)$, since otherwise there is nothing to prove.

the reduction should look like. If we use a protocol where Alice sends $\mathbf{Ay}$ to Bob, where $\mathbf{A}$ is the assumed sensing matrix with a small number $m$ of rows and $\mathbf{y}$ is some vector of Alice's choosing — perhaps a noisy version of some $\mathbf{x} \in X$ — then the communication cost will be $O(m \log n)$. We want to prove that $m = \Omega(k \log \frac{n}{k}) = \Omega(\log |X|)$ (using Lemma 3.2). This means we need a communication lower bound of $\Omega(\log |X| \log n)$. Since the communication lower bound for INDEX is linear, this suggests considering inputs to INDEX where Alice's input has length $\log |X| \log n$, and Bob is given an index $i \in \{1, 2, \ldots, \log |X| \log n\}$.

To describe the reduction formally, let $\text{enc} : X \to \{0,1\}^{\log_2 |X|}$ be a binary encoding of the vectors of $X$. (We can assume that $|X|$ is a power of 2.) Here is the communication protocol for solving INDEX.

(1) Alice interprets her $(\log |X| \log n)$-bit input as $\log n$ blocks of $\log |X|$ bits each. For each $j = 1, 2, \ldots, \log n$, she interprets the bits of the $j$th block as $\text{enc}(\mathbf{x}_j)$ for some $\mathbf{x}_j \in X$. See Figure 2.

(2) Alice computes a suitable linear combination of the $\mathbf{x}_j$'s:

$$\mathbf{y} = \sum_{i=1}^{\log n} \alpha^j \mathbf{x}_j,$$

with the details provided below. Each entry of $\mathbf{y}$ will be a polynomially-bounded integer.

(3) Alice sends $\mathbf{Ay}$ to Bob. This uses $O(m \log n)$ bits of communication.

(4) Bob uses $\mathbf{Ay}$ and the assumed recovery algorithm $R$ to recover all of $\mathbf{x}_1, \ldots, \mathbf{x}_{\log n}$. (Details TBA.)

(5) Bob identifies the block $j$ of $\log |X|$ bits that contains his given index $i \in \{1, 2, \ldots, \log |X| \log n\}$, and outputs the relevant bit of $\text{enc}(\mathbf{x}_j)$.

If we can implement steps (2) and (4), then we're done: this five-step deterministic protocol would solve INDEX on $\log |X| \log n$-bit inputs using $O(m \log n)$ communication. Since the communication complexity of INDEX is linear, we conclude that $m = \Omega(\log |X|) = \Omega(k \log \frac{n}{k})$.[12]

For step (2), let $\alpha \geq 2$ be a sufficiently large constant, depending on the constant $c$ in the recovery guarantee (2) that the matrix $\mathbf{A}$ and algorithm $R$ satisfy. Then

$$\mathbf{y} = \sum_{i=1}^{\log n} \alpha^j \mathbf{x}_j. \tag{3}$$

[12]Thus even the deterministic communication lower bound for INDEX, which is near-trivial to prove (Lecture #2), has very interesting implications. See Section 3.4 for the stronger implications provided by the randomized communication complexity lower bound.

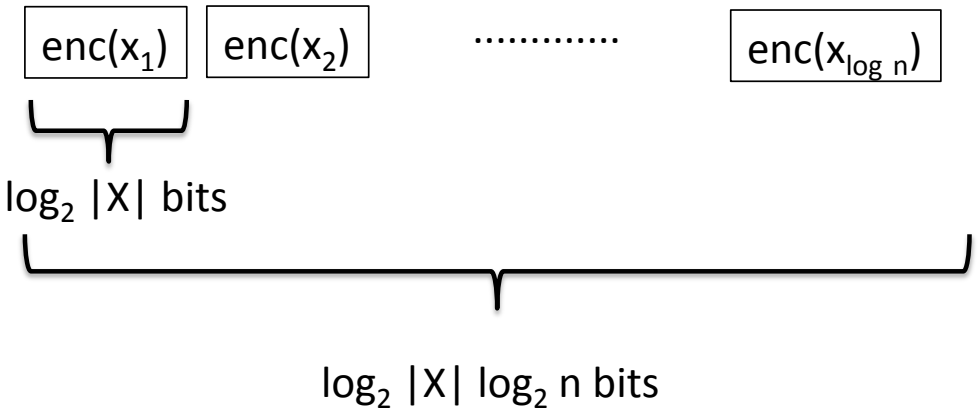


Figure 2: In the reduction, Alice interprets her $\log|X|\log n$-bit input as $\log n$ blocks, with each block encoding some vector $x_j \in X$.

Recall that each $\mathbf{x}_j$ is a 0-1 $n$-vector with exactly $k$ 1's, so $\mathbf{y}$ is just a superposition of $\log n$ scaled copies of such vectors. For example, the $\ell_1$ norm of $\mathbf{y}$ is simply $k\sum_{j=1}^{\log n}\alpha^j$. In particular, since $\alpha$ is a constant, the entries of $\mathbf{y}$ are polynomially bounded non-negative integers, as promised earlier, and $\mathbf{Ay}$ can be described using $O(m\log n)$ bits.

### 3.3.6 Bob's Recovery Algorithm

The interesting part of the protocol and analysis is step (4), where Bob wants to recover the vectors $\mathbf{x}_1,\ldots,\mathbf{x}_{\log n}$ encoded by Alice's input using only knowledge of $\mathbf{Ay}$ and black-box access to the recovery algorithm $R$. To get started, we explain how Bob can recover the last vector $\mathbf{x}_{\log n}$, which suffices to solve INDEX in the lucky case where Bob's index $i$ is one of the last $\log_2|X|$ positions. Intuitively, this is the easiest case, since $\mathbf{x}_{\log n}$ is by far (for large $\alpha$) the largest contributor to the vector $\mathbf{y}$ Alice computes in (3). With $\mathbf{y} = \alpha^{\log n}\mathbf{x}_{\log n} +$ noise, we might hope that the recovery algorithm can extract $\alpha^{\log n}\mathbf{x}_{\log n}$ from $\mathbf{Ay}$.

Bob's first step is the only thing he can do: invoke the recovery algorithm $R$ on the message $\mathbf{Ay}$ from Alice. By assumption, $R$ returns a vector $\hat{\mathbf{y}}$ satisfying

$$\|\hat{\mathbf{y}}-\mathbf{y}\|_1 \le c\cdot \mathrm{res}(\mathbf{y}),$$

where $\mathrm{res}(\mathbf{y})$ is the contribution to $\mathbf{y}$'s $\ell_1$ norm by its smallest $n-k$ coordinates.

Bob then computes $\hat{\mathbf{y}}$'s nearest neighbor $\mathbf{x}^*$ in a scaled version of $X$ under the $\ell_1$ norm — $\mathrm{argmin}_{\mathbf{x}\in X}\|\hat{\mathbf{y}}-\alpha^{\log n}\mathbf{x}\|_1$. Bob can do this computation by brute force.

The key claim is that the computed vector $\mathbf{x}^*$ is indeed $\mathbf{x}^{\log n}$. This follows from a geometric argument, pictured in Figure 3. Briefly: (i) because $\alpha$ is large, $\mathbf{y}$ and $\alpha^{\log n}\mathbf{x}_{\log n}$ are close to each other; (ii) since $\mathbf{y}$ is approximately $k$-sparse (by (i)) and since $R$ satisfies the approximate recovery guarantee in (2), $\hat{\mathbf{y}}$ and $\mathbf{y}$ are close to each other and hence $\alpha^{\log n}\mathbf{x}_{\log n}$ and $\hat{\mathbf{y}}$ are close; and (iii) since distinct vectors in $X$ have large Hamming distance, for every

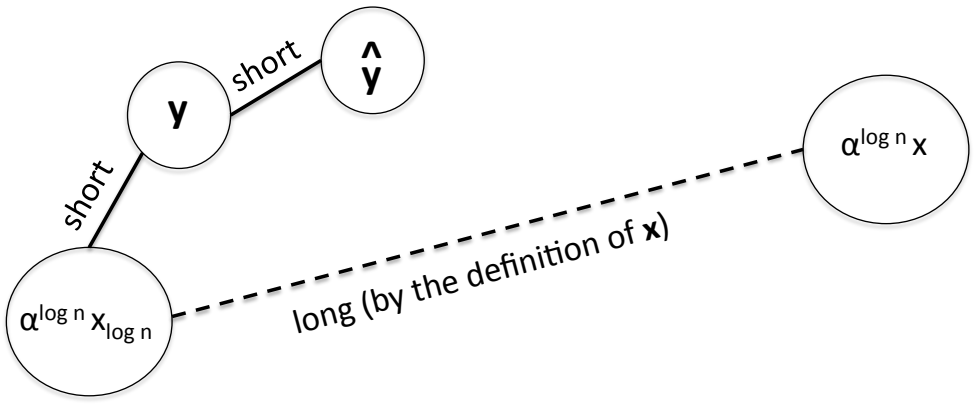


Figure 3: The triangle inequality implies that the vector $\mathbf{x}^*$ computed by Bob from $\hat{\mathbf{y}}$ must be $\mathbf{x}_{\log n}$.

$\mathbf{x} \in X$ other than $\mathbf{x}_{\log n}$, $\alpha^{\log n}\mathbf{x}$ is far from $\mathbf{x}_{\log n}$ and hence also far from $\hat{\mathbf{y}}$. We conclude that $\alpha^{\log n}\mathbf{x}_{\log n}$ is closer to $\hat{\mathbf{y}}$ than any other scaled vector from $X$.

We now supply the details.

(i) Recall that $\mathbf{y}$ is the superposition (i.e., sum) of scaled versions of the vectors $\mathbf{x}_1, \ldots, \mathbf{x}_{\log n}$. $\mathbf{y} - \alpha^{\log n}\mathbf{x}_{\log n}$ is just $\mathbf{y}$ with the last contributor omitted. Thus,

$$\|\mathbf{y} - \alpha^{\log n}\mathbf{x}_{\log n}\|_1 = \sum_{j=1}^{\log n - 1} \alpha^j k.$$

Assuming that $\alpha \geq \max\{2, 200c\}$, where $c$ is the constant that $\mathbf{A}$ and $R$ satisfy in (2), we can bound from above the geometric sum and derive

$$\|\mathbf{y} - \alpha^{\log n}\mathbf{x}_{\log n}\|_1 \leq \frac{.01}{c} k\alpha^{\log n}. \tag{4}$$

(ii) By considering the $n - k$ coordinates of $\mathbf{y}$ other than the $k$ that are non-zero in $\mathbf{x}_{\log n}$, we can upper bound the residual $\text{res}(\mathbf{y})$ by the contributions to the $\ell_1$ weight by $\alpha\mathbf{x}_1, \ldots, \alpha^{\log n - 1}\mathbf{x}_{\log n - 1}$. The sum of these contributions is $k\sum_{j=1}^{\log n - 1}\alpha^j$, which we already bounded above in (4). In light of the recovery guarantee (2), we have

$$\|\hat{\mathbf{y}} - \mathbf{y}\|_1 \leq .01k\alpha^{\log n}. \tag{5}$$

(iii) Let $\mathbf{x}, \mathbf{x}' \in X$ be distinct. By the definition of $X$, $d_H(\mathbf{x}, \mathbf{x}') \geq .1k$. Since $\mathbf{x}, \mathbf{x}'$ are both 0-1 vectors,

$$\|\alpha^{\log n}\mathbf{x} - \alpha^{\log n}\mathbf{x}'\|_1 \geq .1k\alpha^{\log n}. \tag{6}$$

Combining (4) and (5) with the triangle inequality, we have

$$\|\hat{\mathbf{y}} - \alpha^{\log n}\mathbf{x}_{\log n}\|_1 \leq .02k\alpha^{\log n}. \tag{7}$$

Meanwhile, for every other $\mathbf{x} \in X$, combining (6) and (7) with the triangle inequality gives

$$\|\hat{\mathbf{y}} - \alpha^{\log n}\mathbf{x}\|_1 \geq .08k\alpha^{\log n}. \tag{8}$$

Inequalities (7) and (8) imply that Bob's nearest-neighbor computation will indeed recover $\mathbf{x}_{\log n}$.

You'd be right to regard the analysis so far with skepticism. The same reason that Bob can recover $\mathbf{x}_{\log n}$ — because $\mathbf{y}$ is just a scaled version of $\mathbf{x}_{\log n}$, plus some noise — suggests that Bob should not be able to recover the other $\mathbf{x}_j$'s, and hence unable to solve INDEX for indices $i$ outside of the last block of Alice's input.

The key observation is that, after recovering $\mathbf{x}_{\log n}$, Bob can "subtract it out" without any further communication from Alice and then recover $x_{\log n-1}$. Iterating this argument allows Bob to reconstruct all of $\mathbf{x}_1, \ldots, \mathbf{x}_{\log n}$ and hence solve INDEX, no matter what his index $i$ is.

In more detail, suppose Bob has already reconstructed $\mathbf{x}_{j+1}, \ldots, \mathbf{x}_{\log n}$. He's then in a position to form

$$\mathbf{z} = \alpha^{j+1}\mathbf{x}_{j+1} + \cdots + \alpha^{\log n}\mathbf{x}_{\log n}. \tag{9}$$

Then, $\mathbf{y} - \mathbf{z}$ equals the first $j$ contributors to $\mathbf{y}$ — subtracting $\mathbf{z}$ undoes the last $\log n - j$ of them — and is therefore just a scaled version of $\mathbf{x}_j$, plus some relatively small noise (the scaled $\mathbf{x}_\ell$'s with $\ell < j$). This raises the hope that Bob can recover a scaled version of $\mathbf{x}_j$ from $\mathbf{A}(\mathbf{y} - \mathbf{z})$. How can Bob get his hands on the latter vector? (He doesn't know $\mathbf{y}$, and we don't want Alice to send any more bits to Bob.) The trick is to use the linearity of $\mathbf{A}$ — Bob knows $\mathbf{A}$ and $\mathbf{z}$ and hence can compute $\mathbf{Az}$, Alice has already sent him $\mathbf{Ay}$, so Bob just computes $\mathbf{Ay} - \mathbf{Az} = \mathbf{A}(\mathbf{y} - \mathbf{z})$!

After computing $\mathbf{A}(\mathbf{y} - \mathbf{z})$, Bob invokes the recovery algorithm $R$ to obtain a vector $\hat{\mathbf{w}} \in \mathbb{R}^n$ that satisfies

$$\|\hat{\mathbf{w}} - (\mathbf{y} - \mathbf{z})\|_1 \leq c \cdot \text{res}(\mathbf{y} - \mathbf{z}),$$

and computes (by brute force) the vector $\mathbf{x} \in X$ minimizing $\|\hat{\mathbf{w}} - \alpha^j\mathbf{x}\|_1$. The minimizing vector is $\mathbf{x}_j$ — the reader should check that the proof of this is word-for-word the same as our recovery proof for $\mathbf{x}_{\log n}$, with every "$\log n$" replaced by "$j$," every "$\mathbf{y}$" replaced by "$\mathbf{y} - \mathbf{z}$," and every "$\hat{\mathbf{y}}$" replaced by "$\hat{\mathbf{w}}$."

This completes the implementation of step (4) of the protocol, and hence of the reduction from INDEX to the design of a sensing matrix $\mathbf{A}$ and recovery algorithm $R$ that satisfy (2). We conclude that $\mathbf{A}$ must have $m = \Omega(k \log \frac{n}{k})$, which completes the proof of Theorem 3.1.

## 3.4 Lower Bounds for Randomized Recovery

We proved the lower bound in Theorem 3.1 only for fixed matrices $\mathbf{A}$ and deterministic recovery algorithms $R$. This is arguably the most relevant case, but it's also worth asking whether or not better positive results (i.e., fewer rows) are possible for a randomized recovery requirement, where recovery can fail with constant probability. Superficially, the randomization can come from two sources: first, one can use a distribution over matrices $\mathbf{A}$; second, the recovery algorithm (given $\mathbf{Ax}$) can be randomized. Since we're not worrying

about computational efficiency, we can assume without loss of generality that $R$ is deterministic — a randomized recovery algorithm can be derandomized just be enumerating over all of its possible executions and taking the majority vote.

Formally, the relaxed requirement for a positive result is: there exists a constant $c \geq 1$, a distribution $D$ over $m \times n$ matrices $\mathbf{A}$, and a (deterministic) recovery algorithm $R$ such that, for every $\mathbf{x} \in \mathbb{R}^n$, with probability at least $2/3$ (over the choice of $\mathbf{A}$), $R$ returns a vector $\mathbf{x}' \in \mathbb{R}^n$ that satisfies

$$\|\mathbf{x}' - \mathbf{x}\|_1 \leq c \cdot \text{res}(\mathbf{x}).$$

The lower bound in Theorem 3.1 applies even to such randomized solutions. The obvious idea is to follow the proof of Theorem 3.1 to show that a randomized recovery guarantee yields a randomized protocol for INDEX. Since even randomized protocols for the latter problem require linear communication, this would imply the desired lower bound.

The first attempt at modifying the proof of Theorem 3.1 has Alice and Bob using the public coins to pick a sensing matrix $\mathbf{A}$ at random from the assumed distribution — thus, $\mathbf{A}$ is known to both Alice and Bob with no communication. Given the choice of $\mathbf{A}$, Alice sends $\mathbf{Ay}$ to Bob as before and Bob runs the assumed recovery algorithm $R$. With probability at least $2/3$, the result is a vector $\hat{\mathbf{y}}$ from which Bob can recover $\mathbf{x}_{\log n}$. The issue is that Bob has to run $R$ on $\log n$ different inputs, once to recover each of $\mathbf{x}_1, \ldots, \mathbf{x}_{\log n}$, and there is a failure probability of $\frac{1}{3}$ each time.

The obvious fix is to reduce the failure probability by independent trials. So Alice and Bob use the public coins to pick $\ell = \Theta(\log \log n)$ matrices $\mathbf{A}^1, \ldots, \mathbf{A}^\ell$ independently from the assumed distribution. Alice sends $\mathbf{A}^1\mathbf{y}, \ldots, \mathbf{A}^\ell\mathbf{y}$ to Bob, and Bob runs the recovery algorithm $R$ on each of them and computes the corresponding vectors $\mathbf{x}_1, \ldots, \mathbf{x}_\ell \in X$. Except with probability $O(1/\log n)$, a majority of the $\mathbf{x}_j$'s will be the vector $\mathbf{x}_{\log n}$. By a Union Bound over the $\log n$ iterations of Bob's recovery algorithm, Bob successfully reconstructs each of $\mathbf{x}_1, \ldots, \mathbf{x}_{\log n}$ with probability at least $2/3$, completing the randomized protocol for INDEX. This protocol has communication cost $O(m \log n \log \log n)$ and the lower bound for INDEX is $\Omega(\log |X| \log n)$, which yields a lower bound of $m = \Omega(k \log \frac{n}{k} / \log \log n)$ for randomized recovery.

The reduction in the proof of Theorem 3.1 can be modified in a different way to avoid the $\log \log n$ factor and prove the same lower bound of $m = \Omega(k \log \frac{n}{k})$ that we established for deterministic recovery. The trick is to modify the problem being reduced from (previously INDEX) so that it becomes easier — and therefore solvable assuming only a randomized recovery guarantee — subject to its randomized communication complexity remaining linear.

The modified problem is called AUGMENTED INDEX— it's a contrived problem but has proved technically convenient in several applications. Alice gets an input $\mathbf{x} \in \{0,1\}^\ell$. Bob gets an index $i \in \{1, 2, \ldots, \ell\}$ *and also the subsequent bits* $x_{i+1}, \ldots, x_\ell$ *of Alice's input.* This problem is obviously only easier than INDEX, but it's easy to show that its deterministic one-way communication complexity is $\ell$ (see the Exercises). With some work, it can be shown that its randomized one-way communication complexity is $\Omega(\ell)$ (see [2, 4, 6]).

The reduction in Theorem 3.1 is easily adapted to show that a randomized approximate sparse recovery guarantee with matrices with $m$ rows yields a randomized one-way commu-

nication protocol for AUGMENTED INDEX with $\log|X|\log n$-bit inputs with communication cost $O(m\log n)$ (and hence $m = \Omega(\log|X|) = \Omega(k\log\frac{n}{k})$). We interpret Alice's input in the same way as before. Alice and Bob use the public coins to a pick a matrix $\mathbf{A}$ from the assumed distribution and Alice sends $\mathbf{Ay}$ to Bob. Bob is given an index $i \in \{1,2,\ldots,\log|X|\log n\}$ that belongs to some block $j$. Bob is also given all bits of Alice's input after the $i$th one. These bits include $\text{enc}(\mathbf{x}_{j+1}),\ldots,\text{enc}(\mathbf{x}_{\log n})$, so Bob can simply compute $\mathbf{z}$ as in (9) (with no error) and invoke the recovery algorithm $R$ (once). Whenever $\mathbf{A}$ is such that the guarantee (2) holds for $\mathbf{y}-\mathbf{z}$, Bob will successfully reconstruct $\mathbf{x}_j$ and therefore compute the correct answer to AUGMENTED INDEX.

### 3.5 Digression

One could ask if communication complexity is "really needed" for this proof of Theorem 3.1. Churlish observers might complain that, due to the nature of communication complexity lower bounds (like those last lecture), this proof of Theorem 3.1 is "just counting."[13] While not wrong, this attitude is counterproductive. The fact is that adopting the language and mindset of communication complexity has permitted researchers to prove results that had previously eluded many smart people — in this case, the ends justifies the means.

The biggest advantage of using the language of communication complexity is that one naturally thinks in terms of reductions between different lower bounds.[14] Reductions can repurpose a single counting argument, like our lower bound for INDEX, for lots of different problems. Many of the more important lower bounds derived from communication complexity, including today's main result, involve quite clever reductions, and it would be difficult to devise from scratch the corresponding counting arguments.

## References


[1] N. Alon and J. H. Spencer. *The Probabilistic Method*. Wiley, 2008. Third Edition.

[2] Z. Bar-Yossef, T. S. Jayram, R. Krauthgamer, and R. Kumar. The sketching complexity of pattern matching. In *Proceedings of APPROX-RANDOM*, pages 261–272, 2004.

[3] E. J. Candes, J. K. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete fourier information. *IEEE Transactions on Information Theory*, 52(2):489–509, 2006.

[4] K. Do Ba, P. Indyk, E. Price, and D. P. Woodruff. Lower bounds for sparse recovery. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1190–1197, 2010.


[13] Critics said (and perhaps still say) the same thing about the probabilistic method [1].

[14] Thinking in terms of reductions seems to be a "competitive advantage" of theoretical computer scientists — there are several examples where a reduction-based approach yielded new progress on old and seemingly intractable mathematical problems.

[5] D. L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

[6] 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.

[7] A. Moitra. Algorithmic aspects of machine learning. Lecture notes, 2014.

[8] 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.