Electronic Colloquium on Computational Complexity, Report No. 156 (2015)

Communication Complexity (for Algorithm Designers)

Tim Roughgarden

ISSN 1433-8092

Preface

The best algorithm designers prove both possibility and impossibility results — both upper and lower bounds. For example, every serious computer scientist knows a collection of canonical NP-complete problems and how to reduce them to other problems of interest. Communication complexity offers a clean theory that is extremely useful for proving lower bounds for lots of different fundamental problems. Many of the most significant algorithmic consequences of the theory follow from its most elementary aspects.

This document collects the lecture notes from my course "Communication Complexity (for Algorithm Designers)," taught at Stanford in the winter quarter of 2015. The two primary goals of the course are:

(1) Learn several canonical problems in communication complexity that are useful for proving lower bounds for algorithms (Disjointness, Index, Gap-Hamming, etc.).

(2) Learn how to reduce lower bounds for fundamental algorithmic problems to communication complexity lower bounds.

Along the way, we'll also:

(3) Get exposure to lots of cool computational models and some famous results about them — data streams and linear sketches, compressive sensing, space-query time trade-offs in data structures, sublinear-time algorithms, and the extension complexity of linear programs.

(4) Scratch the surface of techniques for proving communication complexity lower bounds (fooling sets, corruption arguments, etc.).

Readers are assumed to be familiar with undergraduate-level algorithms, as well as the statements of standard large deviation inequalities (Markov, Chebyshev, and Chernoff-Hoeffding).

The course begins in Lectures 1–3 with the simple case of one-way communication protocols — where only a single message is sent — and their relevance to algorithm design. Each of these lectures depends on the previous one. Many of the "greatest hits" of communication complexity applications, including lower bounds for small-space streaming algorithms and compressive sensing, are already implied by lower bounds for one-way

protocols. Reasoning about one-way protocols also provides a gentle warm-up to the standard model of general two-party communication protocols, which is the subject of Lecture 4. Lectures 5–8 translate communication complexity lower bounds into lower bounds in several disparate problem domains: the extension complexity of polytopes, data structure design, algorithmic game theory, and property testing. Each of these final four lectures depends only on Lecture 4.

The course Web page (`http://theory.stanford.edu/~tim/w15/w15.html`) contains links to relevant large deviation inequalities, links to many of the papers cited in these notes, and a partial list of exercises. Lecture notes and videos on several other topics in theoretical computer science are available from my Stanford home page.

I am grateful to the Stanford students who took the course, for their many excellent questions: Josh Alman, Dylan Cable, Brynmor Chapman, Michael Kim, Arjun Puranik, Okke Schrijvers, Nolan Skochdopole, Dan Stubbs, Joshua Wang, Huacheng Yu, Lin Zhai, and several auditors whose names I've forgotten. I am also indebted to Alex Andoni, Parikshit Gopalan, Ankur Moitra, and C. Seshadhri for their advice on some of these lectures. The writing of these notes was supported in part by NSF award CCF-1215965.

I always appreciate suggestions and corrections from readers.

Tim Roughgarden
474 Gates Building, 353 Serra Mall
Stanford, CA 94305
Email: `tim@cs.stanford.edu`
WWW: `http://theory.stanford.edu/~tim/`

Contents

List of Figures

Lecture 1

Data Streams: Algorithms and Lower Bounds

1.1 Preamble

This class is mostly about impossibility results — lower bounds on what can be accomplished by algorithms. However, our perspective will be unapologetically that of an algorithm designer.[1] We'll learn lower bound technology on a "need-to-know" basis, guided by fundamental algorithmic problems that we care about (perhaps theoretically, perhaps practically). That said, we will wind up learning quite a bit of complexity theory — specifically, communication complexity — along the way. We hope this viewpoint makes this course and these notes complementary to the numerous excellent courses, books (Jukna (2012) and Kushilevitz and Nisan (1996)), and surveys (e.g. Lee and Shraibman (2009); Lovász (1990); Chattopadhyay and Pitassi (2010); Razborov (2011)) on communication complexity.[2] The theme of communication complexity lower bounds also provides a convenient excuse to take a guided tour of numerous models, problems, and algorithms that are central to modern research in the theory of algorithms but missing from many algorithms textbooks: streaming algorithms, space-time trade-offs in data structures, compressive sensing, sublinear algorithms, extended formulations for linear programs, and more.

Why should an algorithm designer care about lower bounds? The best mathematical researchers can work on an open problem simultaneously from both sides. Even if you have a strong prior belief about whether a given mathematical statement is true or false, failing to prove one direction usefully informs your efforts to prove the other. (And for most us, the prior belief is wrong surprisingly often!) In algorithm design, working on both sides means striving simultaneously for better algorithms and for better lower bounds. For example, a good undergraduate algorithms course teaches you both how to design polynomial-time algorithms and how to prove that a problem is NP-complete — since when you encounter a new computational problem in your research or workplace, both are distinct possibilities. There are many other algorithmic problems where the fundamental difficulty is not the amount of time required, but rather concerns communication or information transmission. The goal of this course is to equip you with the basic tools of communication complexity — its canonical hard problems, the canonical reductions from computation in various models to

[1] Already in this lecture, over half our discussion will be about algorithms and upper bounds!

[2] See Pătraşcu (2009) for a series of four blog posts on data structures that share some spirit with our approach.

the design of low-communication protocols, and a little bit about its lower bound techniques — in the service of becoming a better algorithm designer.

This lecture and the next study the *data stream* model of computation. There are some nice upper bounds in this model (see Sections 1.4 and 1.5), and the model also naturally motivates a severe but useful restriction of the general communication complexity setup (Section 1.7). We'll cover many computational models in the course, so whenever you get sick of one, don't worry, a new one is coming up around the corner.

1.2 The Data Stream Model

The data stream model is motivated by applications in which the input is best thought of as a firehose — packets arriving to a network switch at a torrential rate, or data being generated by a telescope at a rate of one exobyte per day. In these applications, there's no hope of storing all the data, but we'd still like to remember useful summary statistics about what we've seen.

Alternatively, for example in database applications, it could be that data is not thrown away but resides on a big, slow disk. Rather than incurring random access costs to the data, one would like to process it sequentially once (or a few times), perhaps overnight, remembering the salient features of the data in a limited main memory for real-time use. The daily transactions of Amazon or Walmart, for example, could fall into this category.

Formally, suppose data elements belong to a known universe $U = \{1, 2, \ldots, n\}$. The input is a stream $x_1, x_2, \ldots, x_m \in U$ of elements that arrive one-by-one. Our algorithms will not assume advance knowledge of m, while our lower bounds will hold even if m is known a priori. With space $\approx m \log_2 n$, it is possible to store all of the data. The central question in data stream algorithms is: what is possible, and what is impossible, using a one-pass algorithm and much less than $m \log n$ space? Ideally, the space usage should be sublinear or even logarithmic in n and m. We're not going to worry about the computation time used by the algorithm (though our positive results in this model have low computational complexity, anyway).

Many of you will be familiar with a streaming or one-pass algorithm from the following common interview question. Suppose an array A, with length m, is promised to have a *majority element* — an element that occurs strictly more than $m/2$ times. A simple one-pass algorithm, which maintains only the current candidate majority element and a counter for it — so $O(\log n + \log m)$ bits — solves this problem. (If you haven't seen this algorithm before, see the Exercises.) This can be viewed as an exemplary small-space streaming algorithm.[3]

[3] Interestingly, the promise that a majority element exists is crucial. A consequence of the next lecture is that there is no small-space streaming algorithm to check whether or not a majority element exists!

1.3 Frequency Moments

Next we introduce *the* canonical problems in the field of data stream algorithms: computing the *frequency moments* of a stream. These were studied in the paper that kickstarted the field (Alon et al., 1999), and the data stream algorithms community has been obsessed with them ever since.

Fix a data stream $x_1, x_2, \ldots, x_m \in U$. For an element $j \in U$, let $f_j \in \{0, 1, 2, \ldots, m\}$ denote the number of times that j occurs in the stream. For a non-negative integer k, the *kth frequency moment* of the stream is

$$F_k := \sum_{j \in U} f_j^k. \tag{1.1}$$

Note that the bigger k is, the more the sum in (1.1) is dominated by the largest frequencies. It is therefore natural to define

$$F_\infty = \max_{j \in U} f_j$$

as the largest frequency of any element of U.

Let's get some sense of these frequency moments. F_1 is boring — since each data stream element contributes to exactly one frequency f_j, $F_1 = \sum_{j \in U} f_j$ is simply the stream length m. F_0 is the number of distinct elements in the stream (we're interpreting $0^0 = 0$) — it's easy to imagine wanting to compute this quantity, for example a network switch might want to know how many different TCP flows are currently going through it. F_∞ is the largest frequency, and again it's easy to imagine wanting to know this — for example to detect a denial-of-service attack at a network switch, or identify the most popular product on Amazon yesterday. Note that computing F_∞ is related to the aforementioned problem of detecting a majority element. Finally, $F_2 = \sum_{j \in U} f_2^2$ is sometimes called the "skew" of the data — it is a measure of how non-uniform the data stream is. In a database context, it arises naturally as the size of a "self-join" — the table you get when you join a relation with itself on a particular attribute, with the f_j's being the frequencies of various values of this attribute. Having estimates of self-join (and more generally join) sizes at the ready is useful for query optimization, for example. We'll talk about F_2 extensively in the next section.[4]

It is trivial to compute all of the frequency moments in $O(m \log n)$ space, just by storing the x_i's, or in $O(n \log m)$, space, just by computing and storing the f_j's (a $\log m$-bit counter for each of the n universe elements). Similarly, F_1 is trivial to compute in $O(\log m)$ space (via a counter), and F_0 in $O(n)$ space (via a bit vector). Can we do better?

Intuitively, it might appear difficult to improve over the trivial solution. For F_0, for example, it seems like you have to know which elements you've already seen (to avoid double-counting them), and there's an exponential (in n) number of different possibilities for

[4]The problem of computing F_2 and the solution we give for it are also quite well connected to other important concepts, such as compressive sensing and dimensionality reduction.

what you might have seen in the past. As we'll see, this is good intuition for deterministic algorithms, and for (possibly randomized) exact algorithms. Thus, the following positive result is arguably surprising, and very cool.[5]

Theorem 1.1 (Alon et al. 1999) *Both F_0 and F_2 can be approximated, to within a $(1 \pm \epsilon)$ factor with probability at least $1 - \delta$, in space*

$$O\left((\epsilon^{-2}(\log n + \log m) \log \tfrac{1}{\delta}\right). \tag{1.2}$$

Theorem 1.1 refers to two different algorithms, one for F_0 and one for F_2. We cover the latter in detail below. Section 1.5 describes the high-order bit of the F_0 algorithm, which is a modification of the earlier algorithm of Flajolet and Martin (1983), with the details in the exercises. Both algorithms are randomized, and are approximately correct (to within $(1 \pm \epsilon)$) most of the time (except with probability δ). Also, the $\log m$ factor in (1.2) is not needed for the F_0 algorithm, as you might expect. Some further optimization are possible; see Section 1.4.3.

The first reason to talk about Theorem 1.1 is that it's a great result in the field of algorithms — if you only remember one streaming algorithm, the one below might as well be the one.[6] You should never tire of seeing clever algorithms that radically outperform the "obvious solution" to a well-motivated problem. And Theorem 1.1 should serve as inspiration to any algorithm designer — even when at first blush there is no non-trivial algorithm for problem in sight, the right clever insight can unlock a good solution.

On the other hand, there unfortunately are some important problems out there with no non-trivial solutions. And it's important for the algorithm designer to know which ones they are — the less effort wasted on trying to find something that doesn't exist, the more energy is available for formulating the motivating problem in a more tractable way, weakening the desired guarantee, restricting the problem instances, and otherwise finding new ways to make algorithmic progress. A second interpretation of Theorem 1.1 is that it illuminates why such lower bounds can be so hard to prove. A lower bound is responsible for showing that every algorithm, even fiendishly clever ones like those employed for Theorem 1.1, cannot make significant inroads on the problem.

1.4 Estimating F_2: The Key Ideas

In this section we give a nearly complete proof of Theorem 1.1 for the case of $F_2 = \sum_{j \in U} f_2^2$ estimation (a few details are left to the Exercises).

[5] The Alon-Matias-Szegedy paper (Alon et al., 1999) ignited the field of streaming algorithms as a hot area, and for this reason won the 2005 Gödel Prize (a "test of time"-type award in theoretical computer science). The paper includes a number of other upper and lower bounds as well, some of which we'll cover shortly.

[6] Either algorithm, for estimating F_0 or for F_2, could serve this purpose. We present the F_2 estimation algorithm in detail, because the analysis is slightly slicker and more canonical.

1.4.1 The Basic Estimator

The high-level idea is very natural, especially once you start thinking about randomized algorithms.

1. Define a randomized unbiased estimator of F_2, which can be computed in one pass. Small space seems to force a streaming algorithm to lose information, but maybe it's possible to produce a result that's correct "on average."

2. Aggregate many independent copies of the estimator, computed in parallel, to get an estimate that is very accurate with high probability.

This is very hand-wavy, of course — does it have any hope of working? It's hard to answer that question without actually doing some proper computations, so let's proceed to the estimator devised in Alon et al. (1999).

The Basic Estimator:[7]

1. Let $h : U \to \{\pm 1\}$ be a function that associates each universe element with a random sign. On a first reading, to focus on the main ideas, you should assume that h is a totally random function. Later we'll see that relatively lightweight hash functions are good enough (Section 1.4.2), which enables a small-space implementation of the basic ideas.

2. Initialize $Z = 0$.

3. Every time a data stream element $j \in U$ appears, add $h(j)$ to Z. That is, increment Z if $h(j) = +1$ and decrement Z if $h(j) = -1$.[8]

4. Return the estimate $X = Z^2$.

Remark 1.2 A crucial point: since the function h is fixed once and for all before the data stream arrives, an element $j \in U$ is treated consistently every time it shows up. That is, Z is either incremented every time j shows up or is decremented every time j shows up. In the end, element j contributes $h(j)f_j$ to the final value of Z.

First we need to prove that the basic estimator is indeed unbiased.

Lemma 1.3 *For every data stream,*

$$\boldsymbol{E}_h[X] = F_2.$$

[7] This is sometimes called the "tug-of-war" estimator.

[8] This is the "tug of war," between elements j with $h(j) = +1$ and those with $h(j) = -1$.

Proof: We have

$$\begin{aligned}
\mathbf{E}[X] &= \mathbf{E}\left[Z^2\right] \\
&= \mathbf{E}\left[\left(\sum_{j\in U} h(j)f_j\right)^2\right] \\
&= \mathbf{E}\left[\sum_{j\in U} \underbrace{h(j)^2}_{=1} f_j^2 + 2\sum_{j<\ell} h(j)h(\ell)f_j f_\ell\right] \\
&= \underbrace{\sum_{j\in U} f_j^2}_{=F_2} + 2\sum_{j<\ell} f_j f_\ell \underbrace{\mathbf{E}_h[h(j)h(\ell)]}_{=0} \qquad (1.3) \\
&= F_2, \qquad (1.4)
\end{aligned}$$

where in (1.3) we use linearity of expectation and the fact that $h(j) \in \{\pm 1\}$ for every j, and in (1.4) we use the fact that, for every distinct j, ℓ, all four sign patterns for $(h(j), h(\ell))$ are equally likely. ■

Note the reason for both incrementing and decrementing in the running sum Z — it ensures that the "cross terms" $h(j)h(\ell)f_j f_\ell$ in our basic estimator $X = Z^2$ cancel out in expectation. Also note, for future reference, that the only time we used the assumption that h is a totally random function was in (1.4), and we only used the property that all four sign patterns for $(h(j), h(\ell))$ are equally likely — that h is "pairwise independent."

Lemma 1.3 is not enough for our goal, since the basic estimator X is not guaranteed to be close to its expectation with high probability. A natural idea is to take the average of many independent copies of the basic estimator. That is, we'll use t independent functions $h_1, h_2, \ldots, h_t : U \to \{\pm 1\}$ to define estimates $X_1, \ldots, X_t$. On the arrival of a new data stream element, we update all t of the counters $Z_1, \ldots, Z_t$ appropriately, with some getting incremented and others decremented. Our final estimate will be

$$Y = \frac{1}{t}\sum_{i=1}^{t} X_i.$$

Since the X_i's are unbiased estimators, so is Y (i.e., $\mathbf{E}_{h_1,\ldots,h_t}[Y] = F_2$). To see how quickly the variance decreases with the number t of copies, note that

$$\mathbf{Var}[Y] = \mathbf{Var}\left[\frac{1}{t}\sum_{i=1}^{t} X_i\right] = \frac{1}{t^2}\sum_{i=1}^{t}\mathbf{Var}[X_i] = \frac{\mathbf{Var}[X]}{t},$$

where X denotes a single copy of the basic estimator. That is, averaging reduces the variance by a factor equal to the number of copies. Unsurprisingly, the number of copies t (and in

the end, the space) that we need to get the performance guarantee that we want is governed by the variance of the basic estimator. So there's really no choice but to roll up our sleeves and compute it.

Lemma 1.4 *For every data stream,*

$$\boldsymbol{Var}_h[X] \leq 2F_2^2.$$

Lemma 1.4 states the standard deviation of the basic estimator is in the same ballpark as its expectation. That might sound ominous, but it's actually great news — a constant (depending on ϵ and δ only) number of copies is good enough for our purposes. Before proving Lemma 1.4, let's see why.

Corollary 1.5 *For every data stream, with $t = \frac{2}{\epsilon^2\delta}$,*

$$\boldsymbol{Pr}_{h_1,\ldots,h_t}[Y \in (1 \pm \epsilon) \cdot F_2] \geq 1 - \delta.$$

Proof: Recall that *Chebyshev's inequality* is the inequality you want when bounding the deviation of a random variable from its mean parameterized by the number of standard deviations. Formally, it states that for every random variable Y with finite first and second moments, and every $c > 0$,

$$\mathbf{Pr}[|Y - \mathbf{E}[Y]| > c] \leq \frac{\mathbf{Var}[Y]}{c^2}. \tag{1.5}$$

Note that (1.5) is non-trivial (i.e., probability less than 1) once c exceeds Y's standard deviation, and the probability goes down quadratically with the number of standard deviations. It's a simple inequality to prove; see the separate notes on tail inequalities for details.

We are interested in the case where Y is the average of t basic estimators, with variance as in Lemma 1.4. Since we want to guarantee a $(1 \pm \epsilon)$-approximation, the deviation c of interest to us is ϵF_2. We also want the right-hand side of (1.5) to be equal to δ. Using Lemma 1.4 and solving, we get $t = 2/\epsilon^2\delta$.[9] ■

We now stop procrastinating and prove Lemma 1.4.

Proof of Lemma 1.4: Recall that

$$\mathbf{Var}[X] = \mathbf{E}\left[X^2\right] - \left(\underbrace{\mathbf{E}[X]}_{=F_2 \text{ by Lemma 1.3}}\right)^2. \tag{1.6}$$

[9] The dependence on $\frac{1}{\delta}$ can be decreased to logarithmic; see Section 1.4.3.

Zooming in on the $\mathbf{E}\big[X^2\big]$ term, recall that X is already defined as the square of the running sum Z, so X^2 is Z^4. Thus,

$$\mathbf{E}\big[X^2\big] = \mathbf{E}\left[\left(\sum_{j\in U} h(j)f_j\right)^4\right]. \tag{1.7}$$

Expanding the right-hand side of (1.7) yields $|U|^4$ terms, each of the form $h(j_1)h(j_2)h(j_3)h(j_4)f_{j_1}f_{j_2}f_{j_3}f_{j_4}$. (Remember: the h-values are random, the f-values are fixed.) This might seem unwieldy. But, just as in the computation (1.4) in the proof of Lemma 1.3, most of these are zero in expectation. For example, suppose j_1, j_2, j_3, j_4 are distinct. Condition on the h-values of the first three. Since $h(j_4)$ is equally likely to be $+1$ or -1, the conditional expected value (averaged over the two cases) of the corresponding term is 0. Since this holds for all possible values of $h(j_1), h(j_2), h(j_3)$, the unconditional expectation of this term is also 0. This same argument applies to any term in which some element $j \in U$ appears an odd number of times. Thus, when the dust settles, we have

$$\begin{aligned}\mathbf{E}\big[X^2\big] &= \mathbf{E}\left[\sum_{j\in U}\underbrace{h(j)^4}_{=1}f_j^4 + 6\sum_{j<\ell}\underbrace{h(j)^2}_{=1}\underbrace{h(\ell)^2}_{=1}f_j^2f_\ell^2\right] \\ &= \sum_{j\in U} f_j^4 + 6\sum_{j<\ell} f_j^2 f_\ell^2,\end{aligned} \tag{1.8}$$

where the "6" appears because a given $h(j)^2h(\ell)^2f_j^2f_\ell^2$ term with $j < \ell$ arises in $\binom{4}{2} = 6$ different ways.

Expanding terms, we see that

$$F_2^2 = \sum_{j\in U} f_j^4 + 2\sum_{j<\ell} f_j^2 f_\ell^2$$

and hence

$$\mathbf{E}\big[X^2\big] \le 3F_2^2.$$

Recalling (1.6) proves that $\mathbf{Var}[X] \le 2F_2^2$, as claimed. ■

Looking back over the proof of Lemma 1.4, we again see that we only used the fact that h is random in a limited way. In (1.8) we used that, for every set of four distinct universe elements, their 16 possible sign patterns (under h) were equally likely. (This implies the required property that, if j appears in a term an odd number of times, then even after conditioning on the h-values of all other universe elements in the term, $h(j)$ is equally likely to be $+1$ or -1.) That is, we only used the "4-wise independence" of the function h.

1.4.2 Small-Space Implementation via 4-Wise Independent Hash Functions

Let's make sure we're clear on the final algorithm.

1. Choose functions $h_1, \ldots, h_t : U \to \{\pm 1\}$, where $t = \frac{2}{\epsilon^2 \delta}$.
2. Initialize $Z_i = 0$ for $i = 1, 2, \ldots, t$.
3. When a new data stream element $j \in U$ arrives, add $h_i(j)$ to Z_i for every $i = 1, 2, \ldots, t$.
4. Return the average of the Z_i^2's.

Last section proved that, if the h_i's are chosen uniformly at random from all functions, then the output of this algorithm lies in $(1 \pm \epsilon)F_2$ with probability at least $1 - \delta$.

How much space is required to implement this algorithm? There's clearly a factor of $\frac{2}{\epsilon^2 \delta}$, since we're effectively running this many streaming algorithms in parallel, and each needs its own scratch space. How much space does each of these need? To maintain a counter Z_i, which always lies between $-m$ and m, we only need $O(\log m)$ bits. But it's easy to forget that we have to also store the function h_i. Recall from Remark 1.2 the reason: we need to treat an element $j \in U$ consistently every time it shows up in the data stream. Thus, once we choose a sign $h_i(j)$ for j we need to remember it forevermore. Implemented naively, with h_i a totally random function, we would need to remember one bit for each of the possibly $\Omega(n)$ elements that we've seen in the past, which is a dealbreaker.

Fortunately, as we noted after the proofs of Lemmas 1.3 and 1.4, our entire analysis has relied only on 4-wise independence — that when we look at an arbitrary 4-tuple of universe elements, the projection of h on their 16 possible sign patterns is uniform. (Exercise: go back through this section in detail and double-check this claim.) And happily, there are small families of simple hash functions that possess this property.

Fact 1.6 *For every universe U with $n = |U|$, there is a family $\mathcal{H}$ of 4-wise independent hash functions (from U to $\{\pm 1\}$) with size polynomial in n.*

Fact 1.6 and our previous observations imply that, to enjoy an approximation of $(1 \pm \epsilon)$ with probability at least $1 - \delta$, our streaming algorithm can get away with choosing the functions $h_1, \ldots, h_t$ uniformly and independently from $\mathcal{H}$.

If you've never seen a construction of a k-wise independent family of hash functions with $k > 2$, check out the Exercises for details. The main message is to realize that you shouldn't be scared of them — heavy machinery is not required. For example, it suffices to map the elements of U injectively into a suitable finite field (of size roughly $|U|$), and then let $\mathcal{H}$ be the set of all cubic polynomials (with all operations occurring in this field). The final output is then +1 if the polynomial's output (viewed as an integer) is even, and -1 otherwise. Such a hash function is easily specified with $O(\log n)$ bits (just list its four coefficients), and can also be evaluated in $O(\log n)$ space (which is not hard, but we won't say more about it here).

Putting it all together, we get a space bound of

$$O\left(\underbrace{\frac{1}{\epsilon^2\delta}}_{\#\text{ of copies}} \cdot \left(\underbrace{\log m}_{\text{counter}} + \underbrace{\log n}_{\text{hash function}}\right)\right). \tag{1.9}$$

1.4.3 Further Optimizations

The bound in (1.9) is worse than that claimed in Theorem 1.1, with a dependence on $\frac{1}{\delta}$ instead of $\log \frac{1}{\delta}$. A simple trick yields the better bound. In Section 1.4.1, we averaged t copies of the basic estimator to accomplish two conceptually different things: to improve the approximation ratio to $(1 \pm \epsilon)$, for which we suffered an $\frac{1}{\epsilon^2}$ factor, and to improve the success probability to $1 - \delta$, for which we suffered an additional $\frac{1}{\delta}$. It is more efficient to implement these improvements one at a time, rather than in one shot. The smarter implementation first uses $\approx \frac{1}{\epsilon^2}$ copies to obtain an approximation of $(1 \pm \epsilon)$ with probably at least $\frac{2}{3}$ (say). To boost the success probability from $\frac{2}{3}$ to $1 - \delta$, it is enough to run $\approx \log \frac{1}{\delta}$ different copies of this solution, and then take the *median* of their $\approx \log \frac{1}{\delta}$ different estimates. Since we expect at least two-thirds of these estimates to lie in the interval $(1 \pm \epsilon)F_2$, it is very likely that the median of them lies in this interval. The details are easily made precise using a Chernoff bound argument; see the Exercises for details.

Second, believe it or not, the $\log m$ term in Theorem 1.1 can be improved to $\log \log m$. The reason is that we don't need to count the Z_i's exactly, only approximately and with high probability. This relaxed counting problem can be solved using *Morris's algorithm*, which can be implemented as a streaming algorithm that uses $O(\epsilon^{-2} \log \log m \log \frac{1}{\delta})$ space. See the Exercises for further details.

1.5 Estimating F_0: The High-Order Bit

Recall that F_0 denotes the number of distinct elements present in a data stream. The high-level idea of the F_0 estimator is the same as for the F_2 estimator above. The steps are to define a basic estimator that is essentially unbiased, and then reduce the variance by taking averages and medians. (Really making this work takes an additional idea; see Bar-Yossef et al. (2002b) and the Exercises.)

The basic estimator for F_0 — originally from Flajolet and Martin (1983) and developed further in Alon et al. (1999) and Bar-Yossef et al. (2002b) — is as simple as but quite different from that used to estimate F_2. The first step is to choose a random permutation h of U.[10] Then, just remember (using $O(\log n)$ space) the minimum value of $h(x)$ that ever shows up in the data stream.

[10] Or rather, a simple hash function with the salient properties of a random permutation.

Why use the minimum? One intuition comes from the suggestive match between the idempotence of F_0 and of the minimum — adding duplicate copies of an element to the input has no effect on the answer.

Given the minimum $h(x)$-value in the data stream, how do we extract from it an estimate of F_0, the number of distinct elements? For intuition, think about the uniform distribution on $[0, 1]$ (Figure 1.1). Obviously, the expected value of one draw from the distribution is $\frac{1}{2}$. For two i.i.d. draws, simple calculations show that the expected minimum and maximum are $\frac{1}{3}$ and $\frac{2}{3}$, respectively. More generally, the expected order statistics of k i.i.d. draws split the interval into $k + 1$ segments of equal length. In particular, the expected minimum is $\frac{1}{k+1}$. In other words, if you are told that some number of i.i.d. draws were taken from the uniform distribution on $[0, 1]$ and the smallest draw was c, you might guess that there were roughly $1/c$ draws.

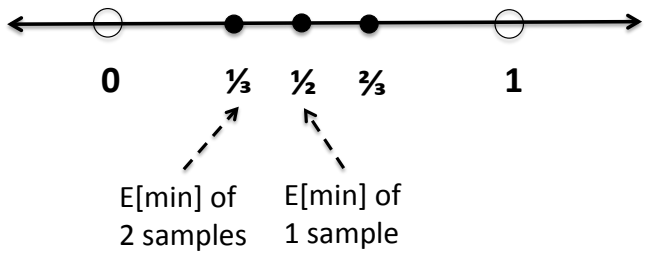


Figure 1.1 The expected order statistics of i.i.d. samples from the uniform distribution on the unit interval are spaced out evenly over the interval.

Translating this idea to our basic F_0 estimator, if there are k distinct elements in a data stream $x_1, \ldots, x_m$, then there are k different (random) hash values $h(x_i)$, and we expect the smallest of these to be roughly $|U|/k$. This leads to the basic estimator $X = |U|/(\min_{i=1}^{m} h(x_i))$. Using averaging and an extra idea to reduce the variance, and medians to boost the success probability, this leads to the bound claimed in Theorem 1.1 (without the $\log m$ term). The details are outlined in the exercises.

Remark 1.7 You'd be right to ask if this high-level approach to probabilistic and approximate estimation applies to all of the frequency moments, not just F_0 and F_2. The approach can indeed be used to estimate F_k for all k. However, the variance of the basic estimators will be different for different frequency moments. For example, as k grows, the statistic F_k becomes quite sensitive to small changes in the input, resulting in probabilistic estimators with large variance, necessitating a large number of independent copies to obtain a good approximation. More generally, no frequency moment F_k with $k \notin \{0, 1, 2\}$ can be computed using only a logarithmic amount of space (more details to come).

1.6 Can We Do Better?

Theorem 1.1 is a fantastic result. But a good algorithm designer is never satisfied, and always wants more. So what are the weaknesses of the upper bounds that we've proved so far?

1. We only have interesting positive results for F_0 and F_2 (and maybe F_1, if you want to count that). What about for $k > 2$ and $k = \infty$?

2. Our F_0 and F_2 algorithms only approximate the corresponding frequency moment. Can we compute it exactly, possibly using a randomized algorithm?

3. Our F_0 and F_2 algorithms are randomized, and with probability δ fail to provide a good approximation. (Also, they are Monte Carlo algorithms, in that we can't tell when they fail.) Can we compute F_0 or F_2 deterministically, at least approximately?

4. Our F_0 and F_2 algorithms use $\Omega(\log n)$ space. Can we reduce the dependency of the space on the universe size?[11]

5. Our F_0 and F_2 algorithms use $\Omega(\epsilon^{-2})$ space. Can the dependence on ϵ^{-1} be improved? The ϵ^{-2} dependence can be painful in practice, where you might want to take $\epsilon = .01$, resulting in an extra factor of 10,000 in the space bound. An improvement to $\approx \epsilon^{-1}$, for example, would be really nice.

Unfortunately, we can't do better — the rest of this lecture and the next (and the exercises) explain why *all* of these compromises are necessary for positive results. This is kind of amazing, and it's also pretty amazing that we can prove it without overly heavy machinery. Try to think of other basic computational problems where, in a couple hours of lecture and with minimal background, you can explain complete proofs of both a non-trivial upper bound and an unconditional (independent of P vs. NP, etc.) matching lower bound.[12]

1.7 One-Way Communication Complexity

We next describe a simple and clean formalism that is extremely useful for proving lower bounds on the space required by streaming algorithms to perform various tasks. The model will be a quite restricted form of the general communication model that we study later — and this is good for us, because the restriction makes it easier to prove lower bounds. Happily, even lower bounds for this restricted model typically translate to lower bounds for streaming algorithms.

[11] This might seem like a long shot, but you never know. Recall our comment about reducing the space dependency on m from $O(\log m)$ to $O(\log \log m)$ via probabilistic approximate counters.

[12] OK, comparison-based sorting, sure. And we'll see a couple others later in this course. But I don't know of that many examples!

In general, communication complexity is a sweet spot. It is a general enough concept to capture the essential hardness lurking in many different models of computation, as we'll see throughout the course. At the same time, it is possible to prove numerous different lower bounds in the model — some of these require a lot of work, but many of the most important ones are easier that you might have guessed. These lower bounds are "unconditional" — they are simply true, and don't depend on any unproven (if widely believed) conjectures like $P \neq NP$. Finally, because the model is so clean and free of distractions, it naturally guides one toward the development of the "right" mathematical techniques needed for proving new lower bounds.

In (two-party) communication complexity, there are two parties, Alice and Bob. Alice has an input $\mathbf{x} \in \{0,1\}^a$, Bob an input $\mathbf{y} \in \{0,1\}^b$. Neither one has any idea what the other's input is. Alice and Bob want to cooperate to compute a Boolean function (i.e., a predicate) $f : \{0,1\}^a \times \{0,1\}^b \to \{0,1\}$ that is defined on their joint input. We'll discuss several examples of such functions shortly.

For this lecture and the next, we can get away with restricting attention to *one-way communication protocols*. All that is allowed here is the following:

1. Alice sends Bob a message $\mathbf{z}$, which is a function of her input $\mathbf{x}$ only.

2. Bob declares the output $f(\mathbf{x}, \mathbf{y})$, as a function of Alice's message $\mathbf{z}$ and his input $\mathbf{y}$ only.

Since we're interested in both deterministic and randomized algorithms, we'll discuss both deterministic and randomized one-way communication protocols.

The *one-way communication complexity* of a Boolean function f is the minimum worst-case number of bits used by any one-way protocol that correctly decides the function. (Or for randomized protocols, that correctly decides it with probability at least 2/3.) That is, it is

$$\min_{\mathcal{P}} \max_{\mathbf{x},\mathbf{y}} \{\text{length (in bits) of Alice's message } \mathbf{z} \text{ when Alice's input is } \mathbf{x}\},$$

where the minimum ranges over all correct protocols.

Note that the one-way communication complexity of a function f is always at most a, since Alice can just send her entire a-bit input $\mathbf{x}$ to Bob, who can then certainly correctly compute f. The question is to understand when one can do better. This will depend on the specific function f. For example, if f is the parity function (i.e., decide whether the total number of 1s in $(\mathbf{x}, \mathbf{y})$ is even or odd), then the one-way communication complexity of f is 1 (Alice just sends the parity of $\mathbf{x}$ to Bob, who's then in a position to figure out the parity of $(\mathbf{x}, \mathbf{y})$).

1.8 Connection to Streaming Algorithms

If you care about streaming algorithms, then you should also care about one-way communication complexity. Why? Because of the unreasonable effectiveness of the following two-step plan to proving lower bounds on the space usage of streaming algorithms.

1. Small-space streaming algorithms imply low-communication one-way protocols.

2. The latter don't exist.

Both steps of this plan are quite doable in many cases.

Does the connection in the first step above surprise you? It's the best kind of statement — genius and near-trivial at the same time. We'll be formal about it shortly, but it's worth remembering a cartoon meta-version of the connection, illustrated in Figure 1.2. Consider a problem that can be solved using a streaming algorithm S that uses space only s. How can we use it to define a low-communication protocol? The idea is for Alice and Bob to treat their inputs as a stream $(\mathbf{x}, \mathbf{y})$, with all of $\mathbf{x}$ arriving before all of $\mathbf{y}$. Alice can feed $\mathbf{x}$ into S without communicating with Bob (she knows $\mathbf{x}$ and S). After processing $\mathbf{x}$, S's state is completely summarized by the s bits in its memory. Alice sends these bits to Bob. Bob can then simply restart the streaming algorithm S seeded with this initial memory, and then feed his input $\mathbf{y}$ to the algorithm. The algorithm S winds up computing some function of $(\mathbf{x}, \mathbf{y})$, and Alice only needs to communicate s bits to Bob to make it happen. The communication cost of the induced protocol is exactly the same as the space used by the streaming algorithm.

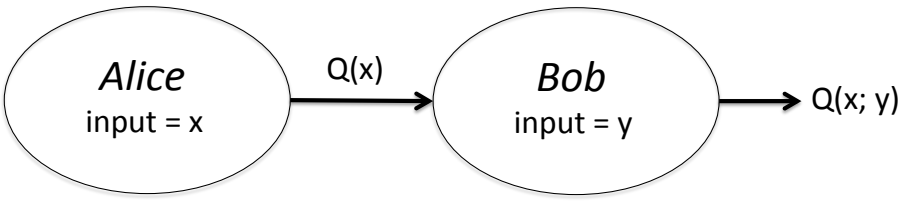


Figure 1.2 Why a small-space streaming algorithm induces a low-communication one-way protocol. Alice runs the streaming algorithm on her input, sends the memory contents of the algorithm to Bob, and Bob resumes the execution of the algorithm where Alice left off on his input.

1.9 The Disjointness Problem

To execute the two-step plan above to prove lower bounds on the space usage of streaming algorithms, we need to come up with a Boolean function that (i) can be reduced to a

streaming problem that we care about and (ii) does not admit a low-communication one-way protocol.

1.9.1 Disjointness Is Hard for One-Way Communication

If you only remember one problem that is hard for communication protocols, it should be the DISJOINTNESS problem. This is the canonical hard problem in communication complexity, analogous to satisfiability (SAT) in the theory of NP-completeness. We'll see more reductions from the DISJOINTNESS problem than from any other in this course.

In an instance of DISJOINTNESS, both Alice and Bob hold n-bit vectors $\mathbf{x}$ and $\mathbf{y}$. We interpret these as characteristic vectors of two subsets of the universe $\{1, 2, \ldots, n\}$, with the subsets corresponding to the "1" coordinates. We then define the Boolean function $DISJ$ in the obvious way, with $DISJ(\mathbf{x}, \mathbf{y}) = 0$ if there is an index $i \in \{1, 2, \ldots, n\}$ with $x_i = y_i = 1$, and $DISJ(\mathbf{x}, \mathbf{y}) = 1$ otherwise.

To warm up, let's start with an easy result.

Proposition 1.8 *Every deterministic one-way communication protocol that computes the function $DISJ$ uses at least n bits of communication in the worst case.*

That is, the trivial protocol is optimal among deterministic protocols.[13] The proof follows pretty straightforwardly from the Pigeonhole Principle — you might want to think it through before reading the proof below.

Formally, consider any one-way communication protocol where Alice always sends at most $n - 1$ bits. This means that, ranging over the 2^n possible inputs $\mathbf{x}$ that Alice might have, she only sends 2^{n-1} distinct messages. By the Pigeonhole Principle, there are distinct messages $\mathbf{x}^1$ and $\mathbf{x}^2$ where Alice sends the same message $\mathbf{z}$ to Bob. Poor Bob, then, has to compute $DISJ(\mathbf{x}, \mathbf{y})$ knowing only $\mathbf{z}$ and $\mathbf{y}$ and not knowing $\mathbf{x}$ — $\mathbf{x}$ could be $\mathbf{x}^1$, or it could be $\mathbf{x}^2$. Letting i denote an index in which $\mathbf{x}^1$ and $\mathbf{x}^2$ differ (there must be one), Bob is really in trouble if his input $\mathbf{y}$ happens to be the ith basis vector (all zeroes except $y_i = 1$). For then, whatever Bob says upon receiving the message $\mathbf{z}$, he will be wrong for exactly one of the cases $\mathbf{x} = \mathbf{x}^1$ or $\mathbf{x} = \mathbf{x}^2$. We conclude that the protocol is not correct.

A stronger, and more useful, lower bound also holds.

Theorem 1.9 *Every randomized one-way protocol*[14] *that, for every input $(\mathbf{x}, \mathbf{y})$, correctly decides the function $DISJ$ with probability at least $\frac{2}{3}$, uses $\Omega(n)$ communication in the worst case.*

[13] We'll see later that the communication complexity remains n even when we allow general communication protocols.

[14] There are different flavors of randomized protocols, such as "public-coin" vs. "private-coin" versions. These distinctions won't matter until next lecture, and we elaborate on them then.

The probability in Theorem 1.9 is over the coin flips performed by the protocol (there is no randomness in the input, which is "worst-case"). There's nothing special about the constant $\frac{2}{3}$ in the statement of Theorem 1.9 — it can be replaced by any constant strictly larger than $\frac{1}{2}$.

Theorem 1.9 is certainly harder to prove than Proposition 1.8, but it's not too bad — we'll kick off next lecture with a proof.[15] For the rest of this lecture, we'll take Theorem 1.9 on faith and use it to derive lower bounds on the space needed by streaming algorithms.

1.9.2 Space Lower Bound for F_∞ (even with Randomization and Approximation)

Recall from Section 1.6 that the first weakness of Theorem 1.1 is that it applies only to F_0 and F_2 (and F_1 is easy). The next result shows that, assuming Theorem 1.9, there is no sublinear-space algorithm for computing F_∞, even probabilistically and approximately.

Theorem 1.10 (Alon et al. 1999) *Every randomized streaming algorithm that, for every data stream of length m, computes F_∞ to within a $(1 \pm .2)$ factor with probability at least $2/3$ uses space $\Omega(\min\{m, n\})$.*

Theorem 1.10 rules out, in a strong sense, extending our upper bounds for F_0, F_1, F_2 to all F_k. Thus, the different frequency moments vary widely in tractability in the streaming model.[16]

Proof of Theorem 1.10: The proof simply implements the cartoon in Figure 1.2, with the problems of computing F_∞ (in the streaming model) and Disjointness (in the one-way communication model). In more detail, let S be a space-s streaming algorithm that for every data stream, with probability at least $2/3$, outputs an estimate in $(1 \pm .2)F_\infty$. Now consider the following one-way communication protocol $\mathcal{P}$ for solving the Disjointness problem (given an input $(\mathbf{x}, \mathbf{y})$):

1. Alice feeds into S the indices i for which $x_i = 1$; the order can be arbitrary. Since Alice knows S and $\mathbf{x}$, this step requires no communication.

2. Alice sends S's current memory state σ to Bob. Since S uses space s, this can be communicated using s bits.

3. Bob resumes the streaming algorithm S with the memory state σ, and feeds into S the indices i for which $y_i = 1$ (in arbitrary order).

[15] A more difficult and important result is that the communication complexity of Disjointness remains $\Omega(n)$ even if we allow arbitrary (not necessarily one-way) randomized protocols. We'll use this stronger result several times later in the course. We'll also briefly discuss the proof in Section 4.3.4 of Lecture 4.

[16] For finite k strictly larger than 2, the optimal space of a randomized $(1 \pm \epsilon)$-approximate streaming algorithm turns out to be roughly $\Theta(n^{1-1/2k})$ (Bar-Yossef et al., 2002a; Chakrabarti et al., 2003; Indyk and Woodruff, 2005). See the exercises for a bit more about these problems.

4. Bob declares "disjoint" if and only if S's final answer is at most 4/3.

To analyze this reduction, observe that the frequency of an index $i \in \{1, 2, \ldots, n\}$ in the data stream induced by $(\mathbf{x}, \mathbf{y})$ is 0 if $x_i = y_i = 0$, 1 if exactly one of x_i, y_i is 1, and 2 if $x_i = y_i = 2$. Thus, F_∞ of this data stream is 2 if $(\mathbf{x}, \mathbf{y})$ is a "no" instance of Disjointness, and is at most 1 otherwise. By assumption, for every "yes" (respectively, "no") input $(\mathbf{x}, \mathbf{y})$, with probability at least 2/3 the algorithm S outputs an estimate that is at most 1.2 (respectively, at least 2/1.2); in this case, the protocol $\mathcal{P}$ correctly decides the input $(\mathbf{x}, \mathbf{y})$. Since $\mathcal{P}$ is a one-way protocol using s bits of communication, Theorem 1.9 implies that $s = \Omega(n)$. Since the data stream length m is n, this reduction also rules out $o(m)$-space streaming algorithms for the problem. ■

Remark 1.11 (The Heavy Hitters Problem) Theorem 1.10 implies that computing the maximum frequency is a hard problem in the streaming model, at least for worst-case inputs. As mentioned, the problem is nevertheless practically quite important, so it's important to make progress on it despite this lower bound. For example, consider the following relaxed version, known as the "heavy hitters" problem: for a parameter k, if there are any elements with frequency bigger than m/k, then find one or all such elements. When k is constant, there are good solutions to this problem: the exercises outline the "Mishra-Gries" algorithm, and the "Count-Min Sketch" and its variants also give good solutions (Charikar et al., 2004; Cormode and Muthukrishnan, 2005).[17] The heavy hitters problem captures many of the applications that motivated the problem of computing F_∞.

1.9.3 Space Lower Bound for Randomized Exact Computation of F_0 and F_2

In Section 1.6 we also criticized our positive results for F_0 and F_2 — to achieve them, we had to make two compromises, allowing approximation and a non-zero chance of failure. The reduction in the proof of Theorem 1.10 also implies that merely allowing randomization is not enough.

Theorem 1.12 (Alon et al. 1999) *For every non-negative integer $k \neq 1$, every randomized streaming algorithm that, for every data stream, computes F_∞ exactly with probability at least 2/3 uses space $\Omega(\min\{n, m\})$.*

The proof of Theorem 1.12 is almost identical to that of Theorem 1.9. The reason the proof of Theorem 1.9 rules out approximation (even with randomization) is because F_∞ differs by a factor of 2 in the two different cases ("yes" and "no" instances of Disjointness).

[17] This does not contradict Theorem 1.9 — in the hard instances of F_∞ produced by that proof, all frequencies are in $\{0, 1, 2\}$ and hence there are no heavy hitters.

For finite k, the correct value of F_k will be at least slightly different in the two cases, which is enough to rule out a randomized algorithm that is exact at least two-thirds of the time.[18]

The upshot of Theorem 1.12 is that, even for F_0 and F_2, approximation is essential to obtain a sublinear-space algorithm. It turns out that randomization is also essential — every deterministic streaming algorithm that always outputs a $(1 \pm \epsilon)$-estimate of F_k (for any $k \neq 1$) uses linear space Alon et al. (1999). The argument is not overly difficult — see the Exercises for the details.

1.10 Looking Backward and Forward

Assuming that randomized one-way communication protocols require $\Omega(n)$ communication to solve the DISJOINTNESS problem (Theorem 1.9), we proved that some frequency moments (in particular, F_∞) cannot be computed in sublinear space, even allowing randomization and approximation. Also, both randomization and approximation are essential for our sublinear-space streaming algorithms for F_0 and F_2.

The next action items are:

1. Prove Theorem 1.9.

2. Revisit the five compromises we made to obtain positive results (Section 1.6). We've showed senses in which the first three compromises are necessary. Next lecture we'll see why the last two are needed, as well.

[18] Actually, this is not quite true (why?). But if Bob also knows the number of 1's in Alice's input (which Alice can communicate in $\log_2 n$ bits, a drop in the bucket), then the exact computation of F_k allows Bob to distinguish "yes" and "no" inputs of DISJOINTNESS (for any $k \neq 1$).

Lecture 2

Lower Bounds for One-Way Communication: Disjointness, Index, and Gap-Hamming

2.1 The Story So Far

Recall from last lecture the simple but useful model of one-way communication complexity. Alice has an input $\mathbf{x} \in \{0,1\}^a$, Bob has an input $\mathbf{y} \in \{0,1\}^b$, and the goal is to compute a Boolean function $f : \{0,1\}^a \times \{0,1\}^b \to \{0,1\}$ of the joint input $(\mathbf{x}, \mathbf{y})$. The players communicate as in Figure 2.1: Alice sends a message $\mathbf{z}$ to Bob as a function of $\mathbf{x}$ only (she doesn't know Bob's input $\mathbf{y}$), and Bob has to decide the function f knowing only $\mathbf{z}$ and $\mathbf{y}$ (he doesn't know Alice's input $\mathbf{x}$). The *one-way communication complexity* of f is the smallest number of bits communicated (in the worst case over $(\mathbf{x}, \mathbf{y})$) of any protocol that computes f. We'll sometimes consider deterministic protocols but are interested mostly in randomized protocols, which we'll define more formally shortly.

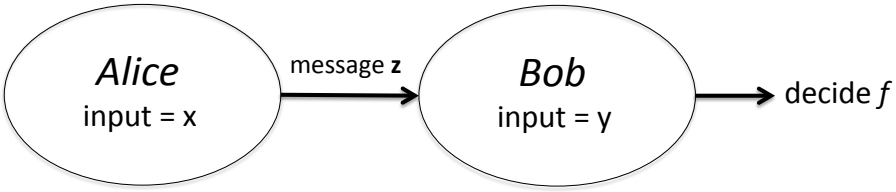


Figure 2.1 A one-way communication protocol. Alice sends a message to Bob that depends only on her input; Bob makes a decision based on his input and Alice's message.

We motivated the one-way communication model through applications to streaming algorithms. Recall the data stream model, where a data stream $x_1, \ldots, x_m \in U$ of elements from a universe of $n = |U|$ elements arrive one by one. The assumption is that there is insufficient space to store all of the data, but we'd still like to compute useful statistics of it via a one-pass computation. Last lecture, we showed that very cool and non-trivial positive results are possible in this model. We presented a slick and low-space ($O(\epsilon^{-2}(\log n + \log m) \log \frac{1}{\delta})$) streaming algorithm that, with probability at least $1 - \delta$, computes a $(1 \pm \epsilon)$-approximation of $F_2 = \sum_{j \in U} f_j^2$, the skew of the data. (Recall that $f_j \in \{0, 1, 2, \ldots, m\}$ is the number of times that j appears in the stream.) We also mentioned the main idea

(details in the homework) for an analogous low-space streaming algorithm that estimates F_0, the number of distinct elements in a data stream.

Low-space streaming algorithms S induce low-communication one-way protocols P, with the communication used by P equal to the space used by S. Such reductions typically have the following form. Alice converts her input $\mathbf{x}$ to a data stream and feeds it into the assumed space-s streaming algorithm S. She then sends the memory of S (after processing $\mathbf{x}$) to Bob; this requires only s bits of communication. Bob then resumes S's execution at the point that Alice left off, and feeds a suitable representation of his input $\mathbf{y}$ into S. When S terminates, it has computed some kind of useful function of $(\mathbf{x}, \mathbf{y})$ with only s bits of communication. The point is that lower bounds for one-way communication protocols — which, as we'll see, we can actually prove in many cases — imply lower bounds on the space needed by streaming algorithms.

Last lecture we used without proof the following result (Theorem 1.9).[1]

Theorem 2.1 *The one-way communication complexity of the* Disjointness *problem is* $\Omega(n)$*, even for randomized protocols.*

We'll be more precise about the randomized protocols that we consider in the next section. Recall that an input of Disjointness is defined by $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$, which we view as characteristic vectors of two subsets of $\{1, 2, \ldots, n\}$, and the output should be "0" is there is an index i with $x_i = y_i = 1$ and "1" otherwise.

We used Theorem 1.9 to prove a few lower bounds on the space required by streaming algorithms. A simple reduction showed that every streaming algorithm that computes F_∞, the maximum frequency, even approximately and with probability 2/3, needs linear (i.e., $\Omega(\min\{n, m\})$) space. This is in sharp contrast to our algorithms for approximating F_0 and F_2, which required only logarithmic space. The same reduction proves that, for F_0 and F_2, exact computation requires linear space, even if randomization is allowed. A different simple argument (see the homework) shows that randomization is also essential for our positive results: every deterministic streaming algorithm that approximates F_0 or F_2 up to a small constant factor requires linear space.

In today's lecture we'll prove Theorem 1.9, introduce and prove lower bounds for a couple of other problems that are hard for one-way communication, and prove via reductions some further space lower bounds for streaming algorithms.

2.2 Randomized Protocols

There are many different flavors of randomized communication protocols. Before proving any lower bounds, we need to be crystal clear about exactly which protocols we're talking about. The good news is that, for algorithmic applications, we can almost always focus

[1] Though we did prove it for the special case of deterministic protocols, using a simple Pigeonhole Principle argument.

on a particular type of randomized protocols. By default, we adopt the following four assumptions and rules of thumb. The common theme behind them is we want to allow as permissible a class of randomized protocols as possible, to maximize the strength of our lower bounds and the consequent algorithmic applications.

Public coins. First, unless otherwise noted, we consider *public-coin* protocols. This means that, before Alice and Bob ever show up, a deity writes an infinite sequence of perfectly random bits on a blackboard visible to both Alice and Bob. Alice and Bob can freely use as many of these random bits as they want — it doesn't contribute to the communication cost of the protocol.

The *private coins* model might seem more natural to the algorithm designer — here, Alice and Bob just flip their own random coins as needed. Coins flipped by one player are unknown to the other player unless they are explicitly communicated.[2] Note that every private-coins protocol can be simulated with no loss by a public-coins protocol: for example, Alice uses the shared random bits 1, 3, 5, etc. as needed, while Bob used the random bits 2, 4, 6, etc.

It turns out that while public-coin protocols are strictly more powerful than private-coin protocols, for the purposes of this course, the two models have essentially the same behavior. In any case, our lower bounds will generally apply to public-coin (and hence also private-coin) protocols.

A second convenient fact about public-coin randomized protocols is that they are equivalent to distributions over deterministic protocols. Once the random bits on the blackboard have been fixed, the protocol proceeds deterministically. Conversely, every distribution over deterministic protocols (with rational probabilities) can be implemented via a public-coin protocol — just use the public coins to choose from the distribution.

Two-sided error. We consider randomized algorithms that are allowed to error with some probability on every input $(\mathbf{x}, \mathbf{y})$, whether $f(\mathbf{x}, \mathbf{y}) = 0$ or $f(\mathbf{x}, \mathbf{y}) = 1$. A stronger requirement would be one-sided error — here there are two flavors, one that forbids false positives (but allows false negatives) and one the forbids false negatives (but allows false positives). Clearly, lower bounds that apply to protocols with two-sided error are at least as strong as those for protocols with one-sided error — indeed, the latter lower bounds are often much easier to prove (at least for one of the two sides). Note that the one-way protocols induces by the streaming algorithms in the last lecture are randomized protocols with two-sided error. There are other problems for which the natural randomized solutions have only one-sided error.[3]

Arbitrary constant error probability. A simple but important fact is that all constant error probabilities $\epsilon \in (0, \frac{1}{2})$ yield the same communication complexity, up to a

[2] Observe that the one-way communication protocols induced by streaming algorithms are private-coin protocols — random coins flipped during the first half of the data stream are only available to the second half if they are explicitly stored in memory.

[3] One can also consider "zero-error" randomized protocols, which always output the correct answer but use a random amount of communication. We won't need to discuss such protocols in this course.

constant factor. The reason is simple: the success probability of a protocol can be boosted through amplification (i.e., repeated trials).[4] In more detail, suppose P uses k bits on communication and has success at least 51% on every input. Imagine repeating P 10000 times. To preserve one-way-ness of the protocol, all of the repeated trials need to happen in parallel, with the public coins providing the necessary 10000 independent random strings. Alice sends 10000 messages to Bob, Bob imagines answering each one — some answers will be "1," others "0" — and concludes by reporting the majority vote of the 10000 answers. In expectation 5100 of the trials give the correct answer, and the probability that more than 5000 of them are correct is big (at least 90%, say). In general, a constant number of trials, followed by a majority vote, boosts the success probability of a protocol from any constant bigger than $\frac{1}{2}$ to any other constant less than 1. These repeated trials increase the amount of communication by only a constant factor. See the exercises and the separate notes on Chernoff bounds for further details.

This argument justifies being sloppy about the exact (constant) error of a two-sided protocol. For upper bounds, we'll be content to achieve error 49% — it can be reduced to an arbitrarily small constant with a constant blow-up in communication. For lower bounds, we'll be content to rule out protocols with error %1 — the same communication lower bounds hold, modulo a constant factor, even for protocols with error 49%.

Worst-case communication. When we speak of the communication used by a randomized protocol, we take the worst case over inputs $(\mathbf{x}, \mathbf{y})$ *and* over the coin flips of the protocol. So if a protocol uses communication at most k, then Alice always sends at most k bits to Bob.

This definition seems to go against our guiding rule of being as permissive as possible. Why not measure only the expected communication used by a protocol, with respect to its coin flips? This objection is conceptually justified but technically moot — for protocols that can err, passing to the technically more convenient worst-case measure can only increase the communication complexity of a problem by a constant factor.

To see this, consider a protocol R that, for every input $(\mathbf{x}, \mathbf{y})$, has two-sided error at most 1/3 (say) and uses at most k bits of communication on average over its coin flips. This protocol uses at most $10k$ bits of communication at least 90% of the time — if it used more than $10k$ bits more than 10% of the time, its expected communication cost would be more than k. Now consider the following protocol R': simulate R for up to $10k$ steps; if R fails to terminate, then abort and output an arbitrary answer. The protocol R' always sends at most $10k$ bits of communication and has error at most that of R, plus 10% (here, $\approx 43\%$). This error probability of R' can be reduced back down (to $\frac{1}{3}$, or whatever) through repeated trials, as before.

In light of these four standing assumptions and rules, we can restate Theorem 1.9 as follows.

[4] We mentioned a similar "median of means" idea last lecture (developed further in the homework), when we discussed how to reduce the $\frac{1}{\delta}$ factor in the space usage of our streaming algorithms to a factor of $\log \frac{1}{\delta}$.

Theorem 2.2 *Every public-coin randomized one-way protocol for* DISJOINTNESS *that has two-sided error at most a constant $\epsilon \in (0, \frac{1}{2})$ uses $\Omega(\min\{n, m\})$ communication in the worst case (over inputs and coin flips).*

Now that we are clear on the formal statement of our lower bound, how do we prove it?

2.3 Distributional Complexity

Randomized protocols are much more of a pain to reason about than deterministic protocols. For example, recall our Pigeonhole Principle-based argument last lecture for deterministic protocols: if Alice holds an n-bit input and always sends at most $n-1$ bits, then there are distinct inputs $\mathbf{x}, \mathbf{x}'$ such that Alice sends the same message $\mathbf{z}$. (For DISJOINTNESS, this ambiguity left Bob in a lurch.) In a randomized protocol where Alice always sends at most $n-1$ bits, Alice can use a different distribution over $(n-1)$-bit messages for each of her 2^n inputs $\mathbf{x}$, and the naive argument breaks down. While Pigeonhole Proof-type arguments can sometimes be pushed through for randomized protocols, this section introduces a different approach.

Distributional complexity is the main methodology by which one proves lower bounds on the communication complexity of randomized algorithms. The point is to reduce the goal to proving lower bounds for *deterministic protocols only*, with respect to a suitably chosen input distribution.

Lemma 2.3 (Yao 1983) *Let D be a distribution over the space of inputs $(\mathbf{x}, \mathbf{y})$ to a communication problem, and $\epsilon \in (0, \frac{1}{2})$. Suppose that every deterministic one-way protocol P with*

$$\boldsymbol{Pr}_{(\mathbf{x},\mathbf{y})\sim D}[P \text{ wrong on } (\mathbf{x}, \mathbf{y})] \leq \epsilon$$

has communication cost at least k. Then every (public-coin) randomized one-way protocol R with (two-sided) error at most ϵ on every input has communication cost at least k.

In the hypothesis of Lemma 2.3, all of the randomness is in the input — P is deterministic, $(\mathbf{x}, \mathbf{y})$ is random. In the conclusion, all of the randomness is in the protocol R — the input is arbitrary but fixed, while the protocol can flip coins. Not only is Lemma 2.3 extremely useful, but it is easy to prove.

Proof of Lemma 2.3: Let R be a randomized protocol with communication cost less than k. Recall that such an R can be written as a distribution over deterministic protocols, call them $P_1, P_2, \ldots, P_s$. Recalling that the communication cost of a randomized protocol is defined as the worst-case communication (over both inputs and coin flips), each deterministic protocol P_i always uses less than k bits of communication. By assumption,

$$\mathbf{Pr}_{(\mathbf{x},\mathbf{y})\sim D}[P_i \text{ wrong on } (\mathbf{x}, \mathbf{y})] > \epsilon$$

for $i = 1, 2, \ldots, s$. Averaging over the P_i's, we have

$$\mathbf{Pr}_{(\mathbf{x},\mathbf{y})\sim D;R}[R \text{ wrong on } (\mathbf{x},\mathbf{y})] > \epsilon.$$

Since the maximum of a set of numbers is at least is average, there exists an input $(\mathbf{x},\mathbf{y})$ such that

$$\mathbf{Pr}_R[R \text{ wrong on } (\mathbf{x},\mathbf{y})] > \epsilon,$$

which completes the proof. ■

The converse of Lemma 2.3 also holds — whatever the true randomized communication complexity of a problem, there exists a bad distribution D over inputs that proves it (Yao, 1983). The proof is by strong linear programming duality or, equivalently, von Neumann's Minimax Theorem for zero-sum games (see the exercises for details). Thus, the distributional methodology is "complete" for proving lower bounds — one "only" needs to find the right distribution D over inputs. In general this is a bit of a dark art, though in today's application D will just be the uniform distribution.

2.4 The INDEX Problem

We prove Theorem 2.2 in two steps. The first step is to prove a linear lower bound on the randomized communication complexity of a problem called INDEX, which is widely useful for proving one-way communication complexity lower bounds. The second step, which is easy, reduces INDEX to DISJOINTNESS.

In an instance of INDEX, Alice gets an n-bit string $\mathbf{x} \in \{0,1\}^n$ and Bob gets an integer $i \in \{1, 2, \ldots, n\}$, encoded in binary using $\approx \log_2 n$ bits. The goal is simply to compute x_i, the ith bit of Alice's input.

Intuitively, since Alice has no idea which of her bits Bob is interested in, she has to send Bob her entire input. This intuition is easy to make precise for deterministic protocols, by a Pigeonhole Principle argument. The intuition also holds for randomized protocols, but the proof takes more work.

Theorem 2.4 (Kremer et al. 1999) *The randomized one-way communication complexity of* INDEX *is* $\Omega(n)$.

With a general communication protocol, where Bob can also send information to Alice, INDEX is trivial to solve using only $\approx \log_2 n$ bits of information — Bob just sends i to Alice. Thus INDEX nicely captures the difficulty of designing non-trivial one-way communication protocols, above and beyond the lower bounds that already apply to general protocols.

Theorem 2.4 easily implies Theorem 2.2.

Proof of Theorem 2.2: We show that DISJOINTNESS reduces to INDEX. Given an input $(\mathbf{x}, i)$ of INDEX, Alice forms the input $\mathbf{x}' = \mathbf{x}$ while Bob forms the input $\mathbf{y}' = e_i$; here e_i is the

standard basis vector, with a "1" in the ith coordinate and "0"s in all other coordinates. Then, $(\mathbf{x}', \mathbf{y}')$ is a "yes" instance of DISJOINTNESS if and only if $x_i = 0$. Thus, every one-way protocol for INDEX induces one for DISJOINTNESS, with the same communication cost and error probability. ■

We now prove Theorem 2.4. While some computations are required, the proof is conceptually pretty straightforward.

Proof of Theorem 2.4: We apply the distributional complexity methodology. This requires positing a distribution D over inputs. Sometimes this takes creativity. Here, the first thing you'd try — the uniform distribution D, where $\mathbf{x}$ and i are chosen independently and uniformly at random — works.

Let c be a sufficiently small constant (like .1 or less) and assume that n is sufficiently large (like 300 or more). We'll show that every deterministic one-way protocol that uses at most cn bits of communication has error (w.r.t. D) at least $\frac{1}{8}$. By Lemma 2.3, this implies that every randomized protocol has error at least $\frac{1}{8}$ on some input. Recalling the discussion about error probabilities in Section 2.2, this implies that for every error $\epsilon' > 0$, there is a constant $c' > 0$ such that every randomized protocol that uses at most $c'n$ bits of communication has error bigger than ϵ'.

Fix a deterministic one-way protocol P that uses at most cn bits of communication. Since P is deterministic, there are only 2^{cn} distinct messages $\mathbf{z}$ that Alice ever sends to Bob (ranging over the 2^n possible inputs $\mathbf{x}$). We need to formalize the intuition that Bob typically (over $\mathbf{x}$) doesn't learn very much about $\mathbf{x}$, and hence typically (over i) doesn't know what x_i is.

Suppose Bob gets a message $\mathbf{z}$ from Alice, and his input is i. Since P is deterministic, Bob has to announce a bit, "0" or "1," as a function of $\mathbf{z}$ and i only. (Recall Figure 2.1). Holding $\mathbf{z}$ fixed and considering Bob's answers for each of his possible inputs $i = 1, 2, \ldots, n$, we get an n-bit vector — Bob's *answer vector* $\mathbf{a}(\mathbf{z})$ when he receives message $\mathbf{z}$ from Alice. Since there are at most 2^{cn} possible messages $\mathbf{z}$, there are at most 2^{cn} possible answer vectors $\mathbf{a}(\mathbf{z})$.

Answer vectors are a convenient way to express the error of the protocol P, with respect to the randomness in Bob's input. Fix Alice's input $\mathbf{x}$, which results in the message $\mathbf{z}$. The protocol is correct if Bob holds an input i with $\mathbf{a}(\mathbf{z})_i = x_i$, and incorrect otherwise. Since Bob's index i is chosen uniformly at random, and independently of $\mathbf{x}$, we have

$$\mathbf{Pr}_i[P \text{ is incorrect} \mid \mathbf{x}, \mathbf{z}] = \frac{d_H(\mathbf{x}, \mathbf{a}(\mathbf{z}))}{n}, \tag{2.1}$$

where $d_H(\mathbf{x}, \mathbf{a}(\mathbf{z}))$ denotes the Hamming distance between the vectors $\mathbf{x}$ and $\mathbf{a}(\mathbf{z})$ (i.e., the number of coordinates in which they differ). Our goal is to show that, with constant probability over the choice of $\mathbf{x}$, the expression (2.1) is bounded below by a constant.

Let $A = \{\mathbf{a}(\mathbf{z}(\mathbf{x})) : \mathbf{x} \in \{0,1\}^n\}$ denote the set of all answer vectors used by the protocol P. Recall that $|A| \leq 2^{cn}$. Call Alice's input $\mathbf{x}$ *good* if there exists an answer vector $\mathbf{a} \in A$

with $d_H(\mathbf{x}, \mathbf{a}) < \frac{n}{4}$, and *bad* otherwise. Geometrically, you should think of each answer vector $\mathbf{a}$ as the center of a ball of radius $\frac{n}{4}$ in the Hamming cube — the set $\{0,1\}^n$ equipped with the Hamming metric. See Figure 2.2. The next claim states that, because there aren't too many balls (only 2^{cn} for a small constant c) and their radii aren't too big (only $\frac{n}{4}$), the union of all of the balls is less than half of the Hamming cube.[5]

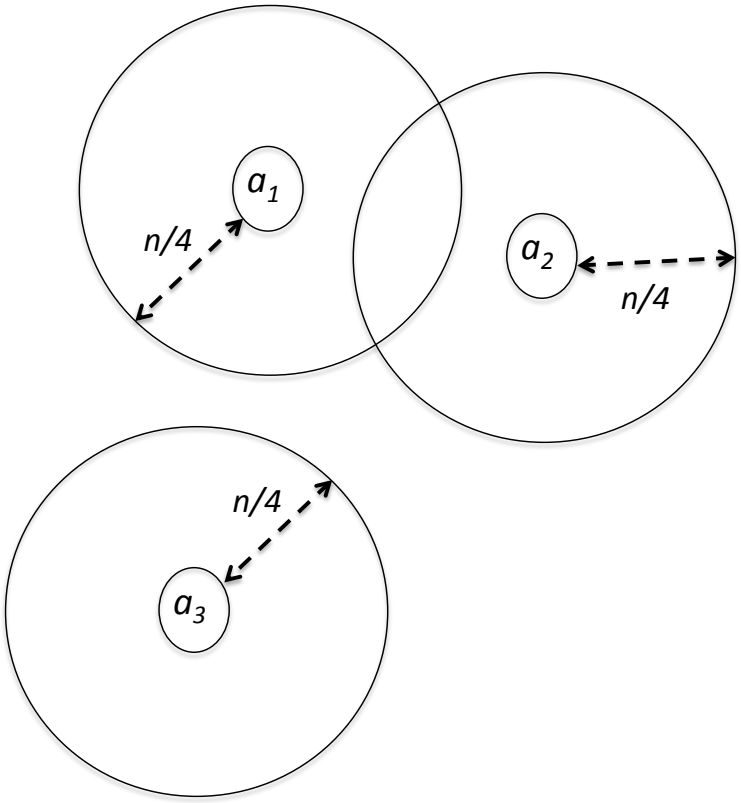


Figure 2.2 Balls of radius $n/4$ in the Hamming metric, centered at the answer vectors used by the protocol P.

Claim: Provided c is sufficiently small and n is sufficiently large, there are at least 2^{n-1} bad inputs $\mathbf{x}$.

[5]More generally, the following is good intuition about the Hamming cube for large n: as you blow up a ball of radius r around a point, the ball includes very few points until r is almost equal to $n/2$; the ball includes roughly half the points for $r \approx n/2$; and for r even modestly larger than r, the ball contains almost all of the points.

Before proving the claim, let's see why it implies the theorem. We can write

$$\begin{aligned}\mathbf{Pr}_{(\mathbf{x},\mathbf{y})\sim D}[D \text{ wrong on } (\mathbf{x},\mathbf{y})] &= \underbrace{\mathbf{Pr}[\mathbf{x} \text{ is good}] \cdot \mathbf{Pr}[D \text{ wrong on } (\mathbf{x},\mathbf{y}) \,|\, \mathbf{x} \text{ is good}]}_{\geq 0} \\ &\quad + \underbrace{\mathbf{Pr}[\mathbf{x} \text{ is bad}]}_{\geq 1/2 \text{ by Claim}} \cdot \mathbf{Pr}[D \text{ wrong on } (\mathbf{x},\mathbf{y}) \,|\, \mathbf{x} \text{ is bad}]\,.\end{aligned}$$

Recalling (2.1) and the definition of a bad input $\mathbf{x}$, we have

$$\begin{aligned}\mathbf{Pr}_{(\mathbf{x},\mathbf{y})}[D \text{ wrong on } (\mathbf{x},\mathbf{y}) \,|\, \mathbf{x} \text{ is bad}] &= \mathbf{E}_{\mathbf{x}}\left[\frac{d_H(\mathbf{x}, \mathbf{a}(\mathbf{z}(\mathbf{x})))}{n} \;\middle|\; \mathbf{x} \text{ is bad}\right] \\ &\geq \mathbf{E}_{\mathbf{x}}\left[\underbrace{\min_{\mathbf{a}\in A} \frac{d_H(\mathbf{x},\mathbf{a})}{n}}_{\geq 1/4 \text{ since } \mathbf{x} \text{ is bad}} \;\middle|\; \mathbf{x} \text{ is bad}\right] \\ &\geq \frac{1}{4}.\end{aligned}$$

We conclude that the protocol P errs on the distribution D with probability at last $\frac{1}{8}$, which implies the theorem. We conclude by proving the claim.

Proof of Claim: Fix some answer vector $\mathbf{a} \in A$. The number of inputs $\mathbf{x}$ with Hamming distance at most $\frac{n}{4}$ from $\mathbf{a}$ is

$$\underbrace{1}_{\mathbf{a}} + \underbrace{\binom{n}{1}}_{d_H(\mathbf{x},\mathbf{a})=1} + \underbrace{\binom{n}{2}}_{d_H(\mathbf{x},\mathbf{a})=2} + \cdots + \underbrace{\binom{n}{n/4}}_{d_H(\mathbf{x},\mathbf{a})=n/2}. \tag{2.2}$$

Recalling the inequality

$$\binom{n}{k} \leq \left(\frac{en}{k}\right)^k,$$

which follows easily from Stirling's approximation of the factorial function (see the exercises), we can crudely bound (2.2) above by

$$n(4e)^{n/4} = n2^{\log_2(4e)\frac{n}{4}} \leq n2^{.861n}.$$

The total number of good inputs $\mathbf{x}$ — the union of all the balls — is at most $|A|2^{.861n} \leq 2^{(.861+c)n}$, which is at most 2^{n-1} for c sufficiently small (say .1) and n sufficiently large (at least 300, say). ■

2.5 Where We're Going

Theorem 2.4 completes our first approach to proving lower bounds on the space required by streaming algorithms to compute certain statistics. To review, we proved from scratch that INDEX is hard for one-way communication protocols (Theorem 2.4), reduced INDEX to DISJOINTNESS to extend the lower bound to the latter problem (Theorem 2.2), and reduced DISJOINTNESS to various streaming computations (last lecture). See also Figure 2.3. Specifically, we showed that linear space is necessary to compute the highest frequency in a data stream (F_∞), even when randomization and approximation are allowed, and that linear space is necessary to compute exactly F_0 or F_2 by a randomized streaming algorithm with success probability 2/3.

$$\underbrace{\text{INDEX}}_{\text{Theorem 2.4}} \xrightarrow{\text{Theorem 2.2}} \text{DISJOINTNESS} \xrightarrow{\text{Lecture 1}} \text{STREAMING}$$

Figure 2.3 Review of the proof structure of linear (in $\min\{n, m\}$) space lower bounds for streaming algorithms. Lower bounds travel from left to right.

We next focus on the dependence on the approximation parameter ϵ required by a streaming algorithm to compute a $(1 \pm \epsilon)$-approximation of a frequency moment. Recall that the streaming algorithms that we've seen for F_0 and F_2 have quadratic dependence on ϵ^{-1}. Thus an approximation of 1% would require a blowup of 10,000 in the space. Obviously, it would be useful to have algorithms with a smaller dependence on ϵ^{-1}. We next prove that space quadratic in ϵ^{-1} is necessary, even allowing randomization and even for F_0 and F_2, to achieve a $(1 \pm \epsilon)$-approximation.

Happily, we'll prove this via reductions, and won't need to prove from scratch any new communication lower bounds. We'll follow the path in Figure 2.4. First we introduce a new problem, also very useful for proving lower bounds, called the GAP-HAMMING problem. Second, we give a quite clever reduction from INDEX to GAP-HAMMING. Finally, it is straightforward to show that one-way protocols for GAP-HAMMING with sublinear communication induce streaming algorithms that can compute a $(1 \pm \epsilon)$-approximation of F_0 or F_2 in $o(\epsilon^{-2})$ space.

$$\underbrace{\text{INDEX}}_{\text{Theorem 2.4}} \xrightarrow{\text{Theorem 2.5}} \text{GAP-HAMMING} \xrightarrow{\text{Section 2.6.2}} \text{STREAMING}$$

Figure 2.4 Proof plan for $\Omega(\epsilon^{-2})$ space lower bounds for (randomized) streaming algorithms that approximate F_0 or F_2 up to a $1 \pm \epsilon$ factor. Lower bounds travel from left to right.

2.6 The Gap-Hamming Problem

Our current goal is to prove that every streaming algorithm that computes a $(1 \pm \epsilon)$-approximation of F_0 or F_2 needs $\Omega(\epsilon^{-2})$ space. Note that we're not going to prove this when $\epsilon \ll 1/\sqrt{n}$, since we can always compute a frequency moment exactly in linear or near-linear space. So the extreme case of what we're trying to prove is that a $(1 \pm \frac{1}{\sqrt{n}})$-approximation requires $\Omega(n)$ space. This special case already requires all of the ideas needed to prove a lower bound of $\Omega(\epsilon^{-2})$ for all larger ϵ as well.

2.6.1 Why Disjointness Doesn't Work

Our goal is also to prove this lower bound through reductions, rather than from scratch. We don't know too many hard problems yet, and we'll need a new one. To motivate it, let's see why Disjointness is not good enough for our purposes.

Suppose we have a streaming algorithm S that gives a $(1 \pm \frac{1}{\sqrt{n}})$-approximation to F_0 — how could we use it to solve Disjointness? The obvious idea is to follow the reduction used last lecture for F_∞. Alice converts her input $\mathbf{x}$ of Disjointness and converts it to a stream, feeds this stream into S, sends the final memory state of S to Bob, and Bob converts his input $\mathbf{y}$ of Disjointness into a stream and resumes S's computation on it. With healthy probability, S returns a $(1 \pm \frac{1}{\sqrt{n}})$-approximation of F_0 of the stream induced by $(\mathbf{x}, \mathbf{y})$. But is this good for anything?

Suppose $(\mathbf{x}, \mathbf{y})$ is a "yes" instance to Disjointness. Then, F_0 of the corresponding stream is $|\mathbf{x}| + |\mathbf{y}|$, where $|\cdot|$ denotes the number of 1's in a bit vector. If $(\mathbf{x}, \mathbf{y})$ is a "no" instance of Disjointness, then F_0 is somewhere between $\max\{|\mathbf{x}|, |\mathbf{y}|\}$ and $|\mathbf{x}| + |\mathbf{y}| - 1$. A particularly hard case is when $|\mathbf{x}| = |\mathbf{y}| = n/2$ and $\mathbf{x}, \mathbf{y}$ are either disjoint or overlap in exactly one element — F_0 is then either n or $n - 1$. In this case, a $(1 \pm \frac{1}{\sqrt{n}})$-approximation of F_0 translates to additive error $\sqrt{n}$, which is nowhere near enough resolution to distinguish between "yes" and "no" instances of Disjointness.

2.6.2 Reducing Gap-Hamming to F_0 Estimation

A $(1 \pm \frac{1}{\sqrt{n}})$-approximation of F_0 is insufficient to solve Disjointness— but perhaps there is some other hard problem that it does solve? The answer is yes, and the problem is estimating the Hamming distance between two vectors $\mathbf{x}, \mathbf{y}$ — the number of coordinates in which $\mathbf{x}, \mathbf{y}$ differ.

To see the connection between F_0 and Hamming distance, consider $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$ and the usual data stream (with elements in $U = \{1, 2, \ldots, n\}$) induced by them. As usual, we can interpret $\mathbf{x}, \mathbf{y}$ as characteristic vectors of subsets A, B of U (Figure 2.5). Observe that the Hamming distance $d_H(\mathbf{x}, \mathbf{y})$ is the just the size of the symmetric difference, $|A \setminus B| + |B \setminus A|$. Observe also that $F_0 = |A \cup B|$, so $|A \setminus B| = F_0 - |B|$ and $|B \setminus A| = F_0 - |A|$, and hence

$d_H(\mathbf{x}, \mathbf{y}) = 2F_0 - |\mathbf{x}| - |\mathbf{y}|$. Finally, Bob knows $|\mathbf{y}|$, and Alice can send $|\mathbf{x}|$ to Bob using $\log_2 n$ bits.

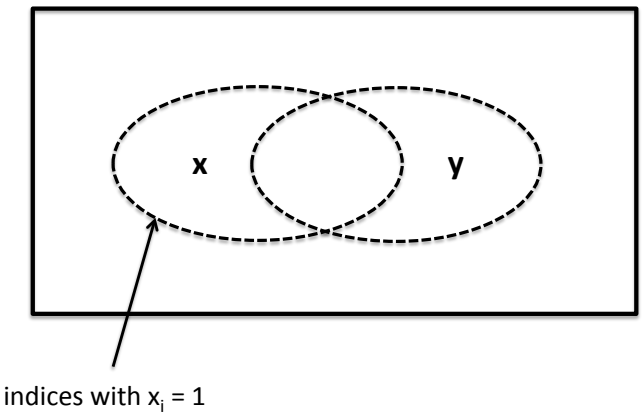


Figure 2.5 The Hamming distance between two bit vectors equals the size of the symmetric difference of the corresponding subsets of 1-coordinates.

The point is that a one-way protocol that computes F_0 with communication c yields a one-way protocol that computes $d_H(\mathbf{x}, \mathbf{y})$ with communication $c + \log_2 n$. More generally, a $(1 \pm \frac{1}{\sqrt{n}})$-approximation of F_0 yields a protocol that estimates $d_H(\mathbf{x}, \mathbf{y})$ up to $2F_0/\sqrt{n} \leq 2\sqrt{n}$ additive error, with $\log_2 n$ extra communication.

This reduction from Hamming distance estimation to F_0 estimation is only useful to us if the former problem has large communication complexity. It's technically convenient to convert Hamming distance estimation into a decision problem. We do this using a "promise problem" — intuitively, a problem where we only care about a protocol's correctness when the input satisfies some conditions (a "promise"). Formally, for a parameter t, we say that a protocol correctly solves Gap-Hamming(t) if it outputs "1" whenever $d_H(\mathbf{x}, \mathbf{y}) < t - c\sqrt{n}$ and outputs "0" whenever $d_H(\mathbf{x}, \mathbf{y}) > t + c\sqrt{n}$, where c is a sufficiently small constant. Note that the protocol can output whatever it wants, without penalty, on inputs for which $d_H(\mathbf{x}, \mathbf{y}) = t \pm c\sqrt{n}$.

Our reduction above shows that, for every t, Gap-Hamming(t) reduces to the $(1 \pm \frac{c}{\sqrt{n}})$-approximation of F_0. Does it matter how we pick t? Remember we still need to prove that the Gap-Hamming(t) problem does not admit low-communication one-way protocols. If we pick $t = 0$, then the problem becomes a special case of the Equality problem (where $f(\mathbf{x}, \mathbf{y}) = 1$ if and only $\mathbf{x} = \mathbf{y}$). We'll see next lecture that the one-way randomized communication complexity of Equality is shockingly low — only $O(1)$ for public-coin protocols. Picking $t = n$ has the same issue. Picking $t = \frac{n}{2}$ seems more promising. For example, it's easy to certify a "no" instance of Equality— just exhibit an index where $\mathbf{x}$ and $\mathbf{y}$ differ. How would you succinctly certify that $d_H(\mathbf{x}, \mathbf{y})$ is either at least $\frac{n}{2} + \sqrt{n}$ or at

most $\frac{n}{2} - \sqrt{n}$? For more intuition, think about two vectors $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$ chosen uniformly at random. The expected Hamming distance between them is $\frac{n}{2}$, with a standard deviation of $\approx \sqrt{n}$. Thus deciding an instance of Gap-Hamming($\frac{n}{2}$) has the flavor of learning an unpredictable fact about two random strings, and it seems difficult to do this without learning detailed information about the particular strings at hand.

2.7]

Lower Bound on the One-Way Communication Complexity of Gap-Hamming

This section dispenses with the hand-waving and formally proves that every protocol that solves Gap-Hamming— with $t = \frac{n}{2}$ and c sufficiently small — requires linear communication.

Theorem 2.5 (Jayram et al. 2008; Woodruff 2004, 2007) *The randomized one-way communication complexity of* Gap-Hamming *is* $\Omega(n)$.

Proof: The proof is a randomized reduction from Index, and is more clever than the other reductions that we've seen so far. Consider an input to Index, where Alice holds an n-bit string $\mathbf{x}$ and Bob holds an index $i \in \{1, 2, \ldots, n\}$. We assume, without loss of generality, that n is odd and sufficiently large.

Alice and Bob generate, without any communication, an input $(\mathbf{x}', \mathbf{y}')$ to Gap-Hamming. They do this one bit at a time, using the publicly available randomness. To generate the first bit of the Gap-Hamming input, Alice and Bob interpret the first n public coins as a random string $\mathbf{r}$. Bob forms the bit $b = r_i$, the ith bit of the random string. Intuitively, Bob says "I'm going to pretend that $\mathbf{r}$ is actually Alice's input, and report the corresponding answer r_i." Meanwhile, Alice checks whether $d_H(\mathbf{x}, \mathbf{r}) < \frac{n}{2}$ or $d_H(\mathbf{x}, \mathbf{r}) > \frac{n}{2}$. (Since n is odd, one of these holds.) In the former case, Alice forms the bit $a = 1$ to indicate that $\mathbf{r}$ is a decent proxy for her input $\mathbf{x}$. Otherwise, she forms the bit $a = 0$ to indicate that $1 - \mathbf{r}$ would have been a better approximation of reality (i.e., of $\mathbf{x}$).

The key and clever point of the proof is that a and b are correlated — positively if $x_i = 1$ and negatively if $x_i = 0$, where $\mathbf{x}$ and i are the given input to Index. To see this, condition on the $n - 1$ bits of $\mathbf{r}$ other than i. There are two cases. In the first case, $\mathbf{x}$ and $\mathbf{r}$ agree on strictly less than or strictly greater than $(n-1)/2$ of the bits so-far. In this case, a is already determined (to 0 or 1, respectively). Thus, in this case, $\mathbf{Pr}[a = b] = \mathbf{Pr}[a = r_i] = \frac{1}{2}$, using that r_i is independent of all the other bits. In the second case, amongst the $n - 1$ bits of $\mathbf{r}$ other than r_i, exactly half of them agree with $\mathbf{x}$. In this case, $a = 1$ if and only if $x_i = r_i$. Hence, if $x_i = 1$, then a and b always agree (if $r_i = 1$ then $a = b = 1$, if $r_i = 0$ then $a = b = 0$). If $x_i = 0$, then a and b always disagree (if $r_i = 1$, then $a = 0$ and $b = 1$, if $r_i = 0$, then $a = 1$ and $b = 0$).

The probability of the second case is the probability of getting $(n-1)/2$ "heads" out of $n - 1$ coin flips, which is $\binom{n-1}{(n-1)/2}$. Applying Stirling's approximation of the factorial

function shows that this probability is bigger than you might have expected, namely $\approx \frac{c'}{\sqrt{n}}$ for a constant c' (see Exercises for details). We therefore have

$$\begin{aligned}\mathbf{Pr}[a=b] &= \underbrace{\mathbf{Pr}[\text{Case 1}]}_{1-\frac{c'}{\sqrt{n}}} \cdot \underbrace{\mathbf{Pr}[a=b \,|\, \text{Case 1}]}_{=\frac{1}{2}} + \underbrace{\mathbf{Pr}[\text{Case 2}]}_{\frac{c'}{\sqrt{n}}} \cdot \underbrace{\mathbf{Pr}[a=b \,|\, \text{Case 2}]}_{\text{1 or 0}} \\ &= \begin{cases} \frac{1}{2} - \frac{c'}{\sqrt{n}} & \text{if } x_i = 1 \\ \frac{1}{2} + \frac{c'}{\sqrt{n}} & \text{if } x_i = 0. \end{cases}\end{aligned}$$

This is pretty amazing when you think about it — Alice and Bob have no knowledge of each other's inputs and yet, with shared randomness but no explicit communication, can generate bits correlated with x_i![6]

The randomized reduction from Index to Gap-Hamming now proceeds as one would expect. Alice and Bob repeat the bit-generating experiment above m independent times to generate m-bit inputs $\mathbf{x}'$ and $\mathbf{y}'$ of Gap-Hamming. Here $m = qn$ for a sufficiently large constant q. The expected Hamming distance between $\mathbf{x}'$ and $\mathbf{y}'$ is at most $\frac{m}{2} - c'\sqrt{m}$ (if $x_i = 1$) or at least $\frac{m}{2} + c'\sqrt{m}$ (if $x_i = 0$). A routine application of the Chernoff bound (see Exercises) implies that, for a sufficiently small constant c and large constant q, with probability at least $\frac{8}{9}$ (say), $d_H(\mathbf{x}', \mathbf{y}') < \frac{m}{2} - c\sqrt{m}$ (if $x_i = 1$) and $d_H(\mathbf{x}', \mathbf{y}') > \frac{m}{2} + c\sqrt{m}$ (if $x_i = 0$). When this event holds, Alice and Bob can correctly compute the answer to the original input $(\mathbf{x}, i)$ to Index by simply invoking any protocol P for Gap-Hamming on the input $(\mathbf{x}', \mathbf{y}')$. The communication cost is that of P on inputs of length $m = \Theta(n)$. The error is at most the combined error of the randomized reduction and of the protocol P — whenever the reduction and P both proceed as intended, the correct answer to the Index input $(\mathbf{x}, i)$ is computed.

Summarizing, our randomized reduction implies that, if there is a (public-coin) randomized protocol for Gap-Hamming with (two-sided) error $\frac{1}{3}$ and sublinear communication, then there is a randomized protocol for Index with error $\frac{4}{9}$. Since we've ruled out the latter, the former does not exist. ■

Combining Theorem 2.5 with our reduction from Gap-Hamming to estimating F_∞, we've proved the following.

Theorem 2.6 *There is a constant $c > 0$ such that the following statement holds: There is no sublinear-space randomized streaming algorithm that, for every data stream, computes F_0 to within a $1 \pm \frac{c}{\sqrt{n}}$ factor with probability at least $2/3$.*

A variation on the same reduction proves the same lower bound for approximating F_2; see the Exercises.

[6] This would clearly not be possible with a private-coin protocol. But we'll see later than the (additive) difference between the private-coin and public-coin communication complexity of a problem is $O(\log n)$, so a linear communication lower bound for one type automatically carries over to the other type.

Our original goal was to prove that the $(1 \pm \epsilon)$-approximate computation of F_0 requires space $\Omega(\epsilon^{-2})$, when $\epsilon \geq \frac{1}{\sqrt{n}}$. Theorem 2.6 proves this in the special case where $\epsilon = \Theta(\frac{1}{\sqrt{n}})$. This can be extended to larger ϵ by a simple "padding" trick. Fix your favorite values of n and $\epsilon \geq \frac{1}{\sqrt{n}}$ and modify the proof of Theorem 2.6 as follows. Reduce from Gap-Hamming on inputs of length $m = \Theta(\epsilon^{-2})$. Given an input $(\mathbf{x}, \mathbf{y})$ of Gap-Hamming, form $(\mathbf{x}', \mathbf{y}')$ by appending $n - m$ zeroes to $\mathbf{x}$ and $\mathbf{y}$. A streaming algorithm with space s that estimates F_0 on the induced data stream up to a $(1 \pm \epsilon)$ factor induces a randomized protocol that solves this special case of Gap-Hamming with communication s. Theorem 2.5 implies that every randomized protocol for the latter problem uses communication $\Omega(\epsilon^{-2})$, so this lower bound carries over to the space used by the streaming algorithm.

Lecture 3

Lower Bounds for Compressive Sensing

3.1 An Appetizer: Randomized Communication Complexity of Equality

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 ϵ, where ϵ is any constant less than $\frac{1}{2}$.

Theorem 3.1 (Yao 1979) *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 3.1 illustrates this point in a starker and clearer way.

Theorem 3.1 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 3.1 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]

[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.

[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.

Proof of Theorem 3.1: 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 ith 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 ith random bit is a 0, then the final inner products are also a and b. If the ith 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 3.1 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 3.1 makes crucial use of public coins. We'll see later that the private-coin one-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 randomized communication complexity of a problem is at most $O(\log n)$ more than its public-coin randomized communication complexity.

The protocol in the proof of Theorem 3.1 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 3.2 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 3.1, 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 3.1 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 3.1 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.

3.2 Sparse Recovery

3.2.1 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 "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 3.1).

3.2.2 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,

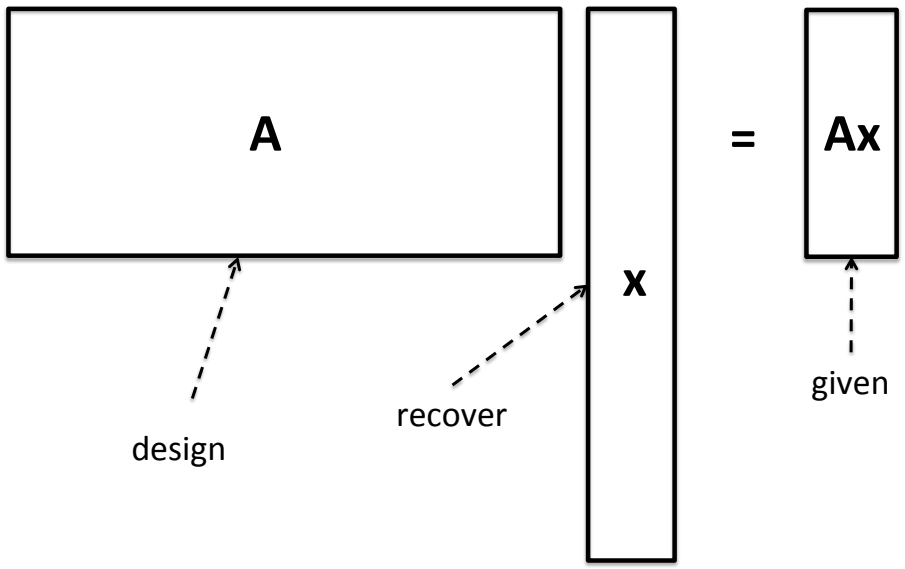


Figure 3.1 The basic compressive sensing setup. The goal is to design a matrix $\mathbf{A}$ such that an unknown sparse signal $\mathbf{x}$ can be recovered from the linear measurements $\mathbf{Ax}$.

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 3.1), for fixed distinct $\mathbf{x}, \mathbf{x}' \in X$, we have

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

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 $\mathbf{x}, \mathbf{x}' \in X$, we have

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

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]

[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.

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.

3.2.3 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.

3.2.4 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 "approximately 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 ℓ_1 norm by its $n-k$ coordinates with smallest magnitudes. Recall that the ℓ_1 norm is just $\|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i|$. If we imagine

[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 $\approx \log \binom{n}{k}$.

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

$$\|\mathbf{x}' - \mathbf{x}\|_1 \leq c \cdot \text{res}(\mathbf{x}). \tag{3.1}$$

Here c is a constant, like 2. The guarantee (3.1) is very compelling. First, if $\mathbf{x}$ is exactly k-sparse, then $\text{res}(\mathbf{x}) = 0$ and so (3.1) 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 ℓ_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 (3.1) 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 3.3 (Candes et al. 2006; Donoho 2006) *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 (3.1) for every $\mathbf{x} \in \mathbb{R}^n$.*

Fact 3.3 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]

3.3 A Lower Bound for Sparse Recovery

3.3.1 Context

At the end of Section 3.2.2, 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? Fact 3.3 provides a resounding affirmative answer. Second, can we get away with an even

[6] See e.g. Moitra (2014) for an introduction to such positive results about sparse recovery. Lecture #9 of the instructor's CS264 course also gives a brief overview.

smaller value of m — even fewer measurements? In this section we prove a relatively recent result of Do Ba et al. (2010), 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 3.4 (Do Ba et al. 2010) *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* (3.1)*, 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 3.4 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 3.3. Given how fundamental the problem is, and how much a non-trivial improvement could potentially matter in practice, this question is well worth asking.

3.3.2 Proof of Theorem 3.4: First Attempt

Recall that we first proved our upper bound of $m = O(k \log \frac{n}{k})$ in the toy setting of Section 3.2.2, and then stated in Fact 3.3 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 (3.1) 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]

[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 (3.1) 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 $\Theta(m)$.

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 in (3.1) — 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. Moitra (2014)).[10] Our argument above only invoked the requirement (3.1) 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 (3.1) 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.

3.3.3 Proof of Theorem 3.4

A Communication Complexity Perspective

We can interpret the failed proof attempt in Section 3.3.2 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 $\mathbf{Ax}$.

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

$$\|\mathbf{x}' - \mathbf{x}\|_1 \leq c \cdot \text{res}(\mathbf{x}). \tag{3.2}$$

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

[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 $\mathbf{Ax}$.

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 3.3.4.

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 $O(m \log n)$.

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 3.5 *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 3.5 is reminiscent of the fact that there are large error-correcting codes with large distance. One way to prove Lemma 3.5 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 3.3.2 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 3.5, fixes the issue by restricting to vectors that all look very different from one another.

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 3.3.2.[11] The parameters offer some strong clues as to what 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 3.5). 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 jth block as $\text{enc}(\mathbf{x}_j)$ for some $\mathbf{x}_j \in X$. See Figure 3.2.

(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]

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

[12] Thus even the deterministic communication lower bound for INDEX, which is near-trivial to prove (Lectures 1–2), has very interesting implications. See Section 3.3.4 for the stronger implications provided by the randomized communication complexity lower bound.

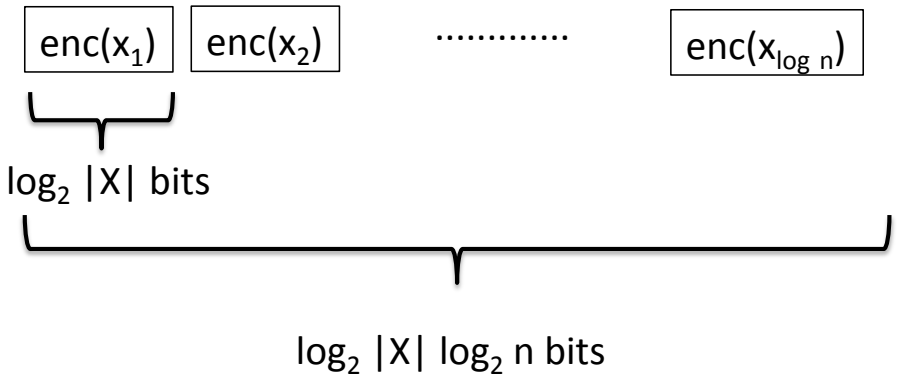


Figure 3.2 In the reduction, Alice interprets her $\log|X| \log n$-bit input as $\log n$ blocks, with each block encoding some vector $x_j \in X$.

For step (2), let $\alpha \geq 2$ be a sufficiently large constant, depending on the constant c in the recovery guarantee (3.2) that the matrix $\mathbf{A}$ and algorithm R satisfy. Then

$$\mathbf{y} = \sum_{i=1}^{\log n} \alpha^j \mathbf{x}_j. \tag{3.3}$$

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 ℓ_1 norm of $\mathbf{y}$ is simply $k\sum_{j=1}^{\log n} \alpha^j$. In particular, since α 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.

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 α) the largest contributor to the vector $\mathbf{y}$ Alice computes in (3.3). 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

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

where $\text{res}(\mathbf{y})$ is the contribution to $\mathbf{y}$'s ℓ_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 ℓ_1 norm — $\text{argmin}_{\mathbf{x}\in X} \|\hat{\mathbf{y}} - \alpha^{\log n}\mathbf{x}\|_1$. 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 3.3. Briefly: (i) because α 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 (3.2), $\hat{\mathbf{y}}$ 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 $\mathbf{x} \in X$ 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.

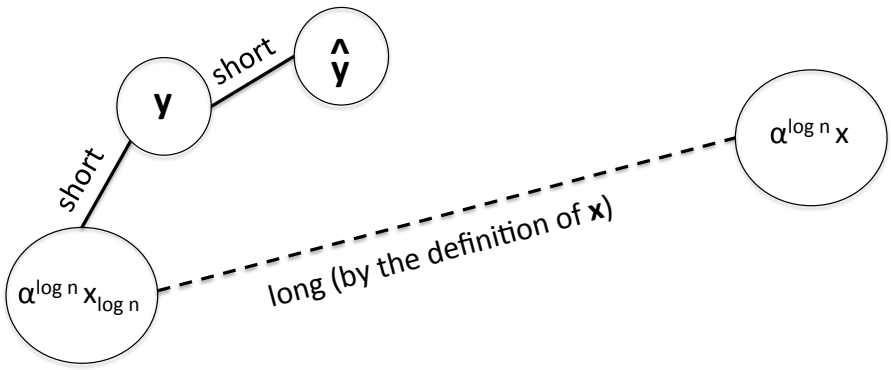


Figure 3.3 The triangle inequality implies that the vector $\mathbf{x}^*$ computed by Bob from $\hat{\mathbf{y}}$ must be $\mathbf{x}_{\log n}$.

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 (3.2), we can bound from above the geometric sum and derive

$$\|\mathbf{y} - \alpha^{\log n}\mathbf{x}_{\log n}\|_1 \leq \frac{.01}{c} k\alpha^{\log n}. \tag{3.4}$$

(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 ℓ_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 (3.4). In light of the recovery guarantee (3.2), we have

$$\|\hat{\mathbf{y}} - \mathbf{y}\|_1 \leq .01k\alpha^{\log n}. \tag{3.5}$$

(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,

$$\|\alpha^{\log n}\mathbf{x} - \alpha^{\log n}\mathbf{x}'\|_1 \geq .1k\alpha^{\log n}. \tag{3.6}$$

Combining (3.4) and (3.5) with the triangle inequality, we have

$$\|\hat{\mathbf{y}} - \alpha^{\log n}\mathbf{x}_{\log n}\|_1 \leq .02k\alpha^{\log n}. \tag{3.7}$$

Meanwhile, for every other $\mathbf{x} \in X$, combining (3.6) and (3.7) with the triangle inequality gives

$$\|\hat{\mathbf{y}} - \alpha^{\log n}\mathbf{x}\|_1 \geq .08k\alpha^{\log n}. \tag{3.8}$$

Inequalities (3.7) and (3.8) 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

$$\mathbf{z} = \alpha^{j+1}\mathbf{x}_{j+1} + \cdots + \alpha^{\log n}\mathbf{x}_{\log n}. \tag{3.9}$$

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 (3.2). We conclude that $\mathbf{A}$ must have $m = \Omega(k \log \frac{n}{k})$, which completes the proof of Theorem 3.4.

3.3.4 Lower Bounds for Randomized Recovery

We proved the lower bound in Theorem 3.4 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 3.4 applies even to such randomized solutions. The obvious idea is to follow the proof of Theorem 3.4 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 3.4 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 3.4 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 ℓ (see the Exercises). With some work, it can be shown that its randomized one-way communication complexity is $\Omega(\ell)$ (see Bar-Yossef et al. (2004); Do Ba et al. (2010); Miltersen et al. (1998)).

The reduction in Theorem 3.4 is easily adapted to show that a randomized approximate sparse recovery guarantee with matrices with m rows yields a randomized one-way communication 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 ith 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 (3.9) (with no error) and invoke the recovery algorithm R (once). Whenever $\mathbf{A}$ is such that the guarantee (3.2) holds for $\mathbf{y} - \mathbf{z}$, Bob will successfully reconstruct $\mathbf{x}_j$ and therefore compute the correct answer to AUGMENTED INDEX.

3.3.5 Digression

One could ask if communication complexity is "really needed" for this proof of Theorem 3.4. Churlish observers might complain that, due to the nature of communication complexity lower bounds (like those last lecture), this proof of Theorem 3.4 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

[13] Critics said (and perhaps still say) the same thing about the probabilistic method (Alon and Spencer, 2008).

[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.

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.

Lecture 4

Boot Camp on Communication Complexity

4.1 Preamble

This lecture covers the most important basic facts about deterministic and randomized communication protocols in the general two-party model, as defined by Yao (1979). Some version of this lecture would normally be the first lecture in a course on communication complexity. How come it's the fourth one here?

The first three lectures were about one-way communication complexity — communication protocols where there is only one message, from Alice to Bob — and its applications. One reason we started with the one-way model is that several of the "greatest hits" of algorithmic lower bounds via communication complexity, such as space lower bounds for streaming algorithms and row lower bounds for compressive sensing matrices, already follow from communication lower bounds for one-way protocols. A second reason is that considering only one-way protocols is a gentle introduction to what communication protocols look like. There are already some non-trivial one-way protocols, like our randomized protocol for Equality. On the other hand, proving lower bounds for one-way protocols is much easier than proving them for general protocols, so it's also a good introduction to lower bound proofs.

The rest of our algorithmic applications require stronger lower bounds that apply to more than just one-way protocols. This lecture gives a "boot camp" on the basic model. We won't say much about applications in this lecture, but the final five lectures all focus on applications. We won't prove any hard results today, and focus instead on definitions and vocabulary, examples, and some easy results. One point of today's lecture is to get a feel for what's involved in proving a communication lower bound for general protocols. It generally boils down to a conceptually simple, if sometimes mathematically challenging, combinatorial problem — proving that a large number of "rectangles" of a certain type are need to cover a matrix.[1]

[1] There are many other methods for proving communication lower bounds, some quite deep and exotic (see e.g. Lee and Shraibman (2009)), but all of our algorithmic applications can ultimately be derived from combinatorial covering-type arguments. For example, we're not even going to mention the famous "rank lower bound." For your edification, some other lower bound methods are discussed in the Exercises.

4.2 Deterministic Protocols

4.2.1 Protocols

We are still in the two-party model, where Alice has an input $\mathbf{x} \in X$ unknown to Bob, and Bob has an input $\mathbf{y} \in Y$ unknown to Alice. (Most commonly, $X = Y = \{0,1\}^n$.) A deterministic communication protocol specifies, as function of the messages sent so far, whose turn it is to speak. A protocol always specifies when the communication has ended and, in each end state, the value of the computed bit. Alice and Bob can coordinate in advance to decide upon the protocol, and both are assumed to cooperative fully. The only constraint faced by the players is that what a player says can depend only on what the player knows — his or her own input, and the history of all messages sent so far.

Like with one-way protocols, we define the cost of a protocol as the maximum number of bits it ever sends, ranging over all inputs. The communication complexity of a function is then the minimum communication cost of a protocol that correctly computes it.

The key feature of general communication protocols absent from the special case of one-way protocols is *interaction* between the two players. Intuitively, interaction should allow the players to communicate much more efficiently. Let's see this in a concrete example.

4.2.2 Example: Clique-Independent Set

The following problem might seem contrived, but it is fairly central in communication complexity. There is a graph $G = (V, E)$ with $|V| = n$ that is known to both players. Alice's private input is a clique C of G — a subset of vertices such that $(u, v) \in E$ for every distinct $u, v \in C$. Bob's private input is an independent set I of G — a subset of vertices such that $(u, v) \notin E$ for every distinct $u, v \in I$. (There is no requirement that C or I is maximal.) Observe that C and I are either disjoint, or they intersect in a single vertex (Figure 4.1). The players' goal is to figure out which of these two possibilities is the case. Thus, this problem is a special case of Disjointness, where players' sets are not arbitrary but rather a clique and an independent set from a known graph.

The naive communication protocol for solving the problem using $\Theta(n)$ bits — Alice can send the characteristic vector of C to Bob, or Bob the characteristic vector of I to Alice, and then the other player computes the correct answer. Since the number of cliques and independent sets of a graph is generally exponential in the number n of vertices, this protocol cannot be made significantly more communication-efficient via a smarter encoding. An easy reduction from Index shows that one-way protocols, including randomized protocols, require $\Omega(n)$ communication (exercise).

The players can do much better by interacting. Here is the protocol.

1. If there is a vertex $v \in C$ with $\deg(v) < \frac{n}{2}$, then Alice sends the name of an arbitrary such vertex to Bob ($\approx \log_2 n$ bits).

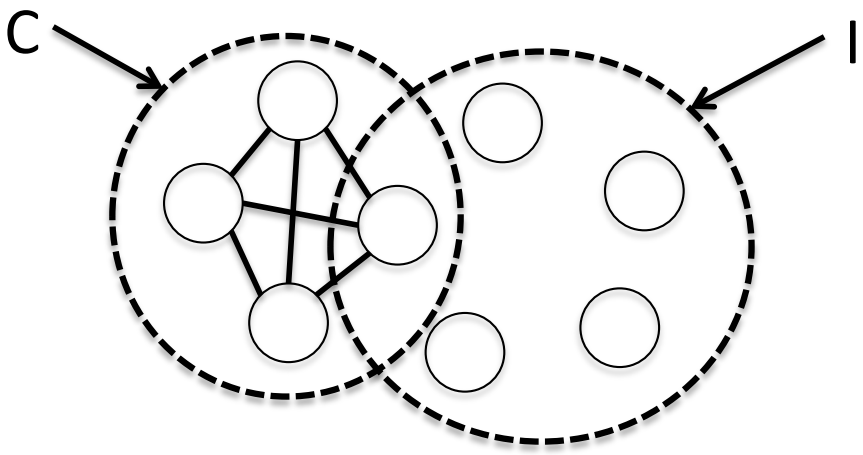


Figure 4.1 A clique C and an independent set I overlap in zero or one vertices.

a) Bob announces whether or not $v \in I$ (1 bit). If so, the protocol terminates with conclusion "not disjoint."

b) Otherwise, Alice and Bob recurse on the subgraph H induced by v and its neighbors.

[Note: H contains at most half the nodes of G. It contains all of C and, if I intersects C, it contains the vertex in their intersection. C and I intersect in G if and only if their projections to H intersect in H.]

2. Otherwise, Alice sends a "NULL" message to Bob ($\approx \log_2 n$ bits).

3. If there is a vertex $v \in I$ with $\deg(v) \geq \frac{n}{2}$, then Bob sends the name of an arbitrary such vertex to Bob ($\approx \log_2 n$ bits).

   a) Alice announces whether or not $v \in C$ (1 bit). If so, the protocol terminates ("not disjoint").

   b) If not, Alice and Bob recurse on the subgraph H induced by v and its non-neighbors.

   [Note: H contains at most half the nodes of G. It contains all of I and, if C intersects I, it contains the vertex in their intersection. Thus the function's answer in H is the same as that in G.]

4. Otherwise, Bob terminates the protocol and declares "disjoint."

   [Disjointness is obvious since, at this point in the protocol, we know that $\deg(v) < \frac{n}{2}$ for all $v \in C$ and $\deg(v) \geq \frac{n}{2}$ for all $v \in I$.]

Since each iteration of the protocol uses $O(\log n)$ bits of communication and cuts the number of vertices of the graph in half (or terminates), the total communication is $O(\log^2 n)$. As previously noted, such a result is impossible without interaction between the players.

4.2.3 Trees and Matrices

The Clique-Independent Set problem clearly demonstrates that we need new lower bound techniques to handle general communication protocols — the straightforward Pigeonhole Principle arguments that worked for one-way protocols are not going to be good enough. At first blush this might seem intimidating — communication protocols can do all sorts of crazy things, so how can we reason about them in a principled way? How can we connect properties of a protocol to the function that it computes? Happily, we can quickly build up some powerful machinery for answering these questions.

First, we observe that deterministic communication protocols are really just binary trees. We'll almost never use this fact directly, but it should build some confidence that protocols are familiar and elementary mathematical objects.

The connection is easiest to see by example; see Figure 4.2. Consider the following protocol for solving Equality with $n = 2$ (i.e., $f(\mathbf{x}, \mathbf{y}) = 1$ if and only if $\mathbf{x} = \mathbf{y}$). Alice begins by sending her first bit. If Bob's first bit is different, he terminates the protocol and announces "not equal." If Bob's first bit is the same, then he transmits the same bit back. In this case, Alice then sends her second bit. At this point, Bob knows Alice's whole input and can therefore compute the correct answer.

In Figure 4.2, each node corresponds to a possible state of the protocol, and is labeled with the player whose turn it is to speak. Thus the labels alternate with the levels, with the root belonging to Alice.[2] There are 10 leaves, representing the possible end states of the protocol. There are two leaves for the case where Alice and Bob have different first bits and the protocol terminates early, and eight leaves for the remaining cases where Alice and Bob have the same first bit. Note that the possible transcripts of the protocol are in one-to-one correspondence with the root-leaf nodes of the tree — we use leaves and transcripts interchangeably below.

We can view the leaves as a partition $\{Z(\ell)\}$ of the input space $X \times Y$, with $Z(\ell)$ the inputs $(\mathbf{x}, \mathbf{y})$ such that the protocol terminates in the leaf ℓ. In our example, there are 10 leaves for the 16 possible inputs $(\mathbf{x}, \mathbf{y})$, so different inputs can generate the same transcript — more on this shortly.

Next note that we can represent a function (from $(\mathbf{x}, \mathbf{y})$ to $\{0, 1\}$) a matrix. In contrast to the visualization exercise above, we'll use this matrix representation *all the time*. The rows are labeled with the set X of possible inputs of Alice, the columns with the set Y of possible inputs of Bob. Entry $(\mathbf{x}, \mathbf{y})$ of the matrix is $f(\mathbf{x}, \mathbf{y})$. Keep in mind that this matrix is fully known to both Alice and Bob when they agree on a protocol.

[2] In general, players need not alternate turns in a communication protocol.

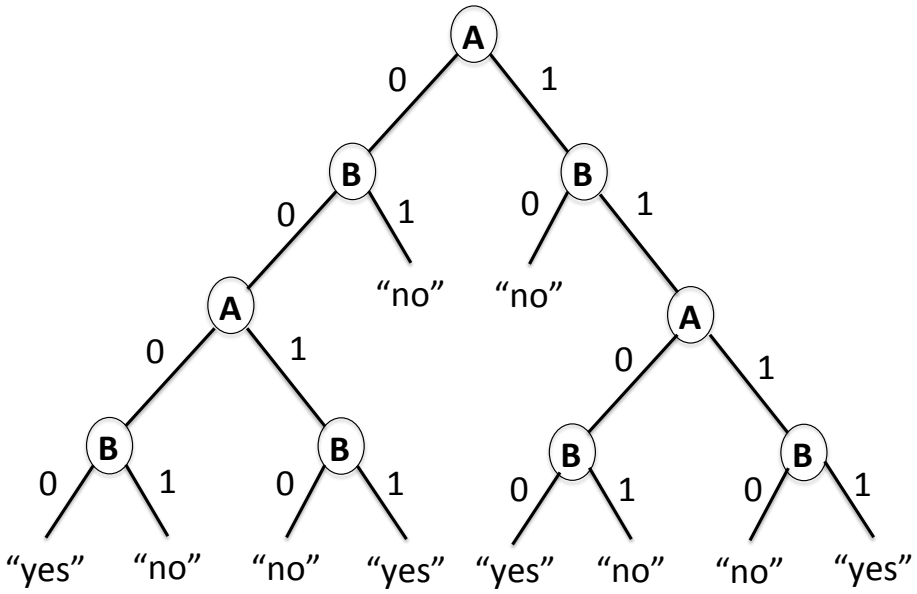


Figure 4.2 The binary tree induced by a communication protocol for EQUALITY with $n = 2$.

For example, suppose that $X = Y = \{0,1\}^2$, resulting in 4×4 matrices. EQUALITY then corresponds to the identity matrix:

$$\begin{array}{c} \\ 00 \\ 01 \\ 10 \\ 11 \end{array} \begin{array}{c} \begin{array}{cccc} 00 & 01 & 10 & 11 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \tag{4.1}$$

If we define the GREATER-THAN function as 1 whenever $\mathbf{x}$ is at least $\mathbf{y}$ (where $\mathbf{x}$ and $\mathbf{y}$ are interpreted as non-negative integers, written in binary), then we just fill in the lower triangle with 1s:

$$\begin{array}{c} \\ 00 \\ 01 \\ 10 \\ 11 \end{array} \begin{array}{c} \begin{array}{cccc} 00 & 01 & 10 & 11 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \end{array}$$

We also write out the matrix for DISJOINTNESS, which is somewhat more inscrutable:

$$\begin{array}{c} \\ 00 \\ 01 \\ 10 \\ 11 \end{array} \begin{array}{c} \begin{array}{cccc} 00 & 01 & 10 & 11 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

4.2.4 Protocols and Rectangles

How can we reason about the behavior of a protocol? Just visualizing them as trees is not directly useful. We know that simple Pigeonhole Principle-based arguments are not strong enough, but it still feels like we want some kind of counting argument.

To see what might be true, let's run the 2-bit EQUALITY protocol depicted in Figure 4.2 and track its progress using the matrix in (4.1). Put yourself in the shoes of an outside observer, who knows neither $\mathbf{x}$ nor $\mathbf{y}$, and makes inferences about $(\mathbf{x}, \mathbf{y})$ as the protocol proceeds. When the protocol terminates, we'll have carved up the matrix into 10 pieces, one for each leaf of protocol tree — the protocol transcript reveals the leaf to an outside observer, but nothing more.

Before the protocol beings, all 16 inputs are fair game. After Alice sends her first bit, the outside observer can narrow down the possible inputs into a set of 8 — the top 8 if Alice sent a 0, the bottom 8 if she sent a 1. The next bit sent gives away whether or not Bob's first bit is a 0 or 1, so the outsider observer learns which quadrant the input lies in. Interestingly, in the northeastern and southwestern quadrants, all of the entries are 0. In these cases, even though ambiguity remains about exactly what the input $(\mathbf{x}, \mathbf{y})$ is, the function's value $f(\mathbf{x}, \mathbf{y})$ has been determined (it is 0, whatever the input). It's no coincidence that these two regions correspond to the two leaves of the protocol in Figure 4.2 that stop early, with the correct answer. If the protocol continues further, then Alice's second bit splits the northwestern and southeastern quadrants into two, and Bob's final bit splits them again, now into singleton regions. In these cases, an outside observer learns the entire input $(\mathbf{x}, \mathbf{y})$ from the protocol's transcript.[3]

What have we learned? We already knew that every protocol induces a partition of the input space $X \times Y$, with one set for each leaf or, equivalently, for each distinct transcript. At least for the particular protocol that we just studied, each of the sets has a particularly nice submatrix form (Figure 4.3). This is true in general, in the following sense.

[3] It's also interesting to do an analogous thought experiment from the perspective of one of the players. For example, consider Bob's perspective when the input is (00,01). Initially Bob knows that the input lies in the second column but is unsure of the row. After Alice's first message, Bob knows that the input is in the second column and one of the first two rows. Bob still cannot be sure about the correct answer, so the protocol proceeds.

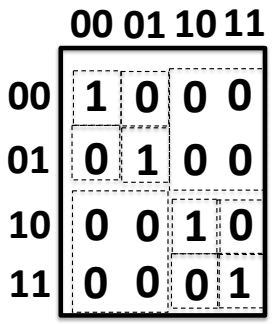


Figure 4.3 The partition of the input space $X \times Y$ according to the 10 different transcripts that can be generated by the EQUALITY protocol.

Lemma 4.1 (Rectangles) *For every transcript* $\mathbf{z}$ *of a deterministic protocol* P*, the set of inputs* $(\mathbf{x}, \mathbf{y})$ *that generate* $\mathbf{z}$ *are a* rectangle*, of the form* $A \times B$ *for* $A \subseteq X$ *and* $B \subseteq Y$*.*

A rectangle just means a subset of the input space $X \times Y$ that can be written as a product. For example, the set $\{(00, 00), (11, 11)\}$ is *not* a rectangle, while the set $\{(00, 00), (11, 00), (00, 11), (11, 11)\}$ is. In general, a subset $S \subseteq X \times Y$ is a rectangle if and only if it is closed under "mix and match," meaning that whenever $(\mathbf{x}_1, \mathbf{y}_1)$ and $(\mathbf{x}_2, \mathbf{y}_2)$ are in S, so are $(\mathbf{x}_1, \mathbf{y}_2)$ and $(\mathbf{x}_2, \mathbf{y}_1)$ (see the Exercises).

Don't be misled by our example (Figure 4.3), where the rectangles induced by our protocol happen to be "contiguous." For example, if we keep the protocol the same but switch the order in which we write down the rows and columns corresponding to 01 and 10, we get an analogous decomposition in which the two large rectangles are not contiguous. In general, you shouldn't even think of X and Y as ordered sets. Rectangles are sometimes called *combinatorial* rectangles to distinguish them from "geometric" rectangles and to emphasize this point.

Lemma 4.1 is extremely important, though its proof is straightforward — we just follow the protocol like in our example above. Intuitively, each step of a protocol allows an outside observer to narrow down the possibilities for $\mathbf{x}$ while leaving the possibilities for $\mathbf{y}$ unchanged (if Alice speaks) or vice versa (if Bob speaks).

Proof of Lemma 4.1: Fix a deterministic protocol P. We proceed by induction on the number of bits exchanged. For the base case, all inputs $X \times Y$ begin with the empty transcript. For the inductive step, consider an arbitrary t-bit transcript-so-far $\mathbf{z}$ generated by P, with $t \geq 1$. Assume that Alice was the most recent player to speak; the other case is analogous. Let $\mathbf{z}'$ denote $\mathbf{z}$ with the final bit $b \in \{0, 1\}$ lopped off. By the inductive hypothesis, the set of inputs that generate $\mathbf{z}'$ has the form $A \times B$. Let $A_b \subseteq A$ denote the inputs $\mathbf{x} \in A$ such that, in the protocol P, Alice sends the bit b given the transcript $\mathbf{z}'$. (Recall that the message sent by a player is a function only of his or her private input and the history of the protocol

so far.) Then the set of inputs that generate $\mathbf{z}$ are $A_b \times B$, completing the inductive step. ■

Note that Lemma 4.1 makes no reference to a function f — it holds for any deterministic protocol, whether or not it computes a function that we care about. In Figure 4.3, we can clearly see an additional property of all of the rectangles — with respect to the matrix in (4.1), every rectangle is *monochromatic*, meaning all of its entries have the same value. This is true for any protocol that correctly computes a function f.

Lemma 4.2 *If a deterministic protocol P computes a function f, then every rectangle induced by P is monochromatic in the matrix $M(f)$.*

Proof: Consider an arbitrary combinatorial rectangle $A \times B$ induced by P, with all inputs in $A \times B$ inducing the same transcript. The output of P is constant on $A \times B$. Since P correctly computes f, f is also constant on $A \times B$. ■

Amazingly, the minimal work we've invested so far already yields a powerful technique for lower bounding the deterministic communication complexity of functions.

Theorem 4.3 *Let f be a function such that every partition of $M(f)$ into monochromatic rectangles requires at least t rectangles. Then the deterministic communication complexity of f is at least $\log_2 t$.*

Proof: A deterministic protocol with communication cost c can only generate 2^c distinct transcripts — equivalently, its (binary) protocol tree can only have 2^c leaves. If such a protocol computes the function f, then by Lemmas 4.1 and 4.2 it partitions $M(f)$ into at most 2^c monochromatic rectangles. By assumption, $2^c \geq t$ and hence $c \geq \log_2 t$. ■

Rather than applying Theorem 4.3 directly, we'll almost always be able to prove a stronger and simpler condition. To partition a matrix, one needs to cover all of its entries with disjoint sets. The disjointness condition is annoying. So by a *covering* of a 0-1 matrix, we mean a collection of subsets of entries whose union includes all of its elements — overlaps between these sets are allowed. See Figure 4.4.

Corollary 4.4 *Let f be a function such that every covering of $M(f)$ by monochromatic rectangles requires at least t rectangles. Then the deterministic communication complexity of f is at least $\log_2 t$.*

Communication complexity lower bounds proved using covers — including all of those proved in Section 4.2.5 — automatically apply also to more general "nondeterministic" communication protocols, as well as randomized protocols with 1-sided error. We'll discuss this more next lecture, when it will be relevant.

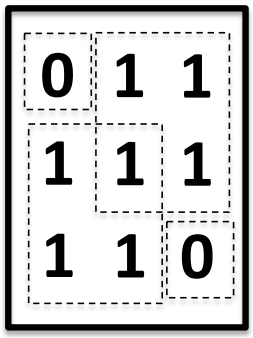


Figure 4.4 A covering by four monochromatic rectangles that is not a partition.

4.2.5 Lower Bounds for Equality and Disjointness

Armed with Corollary 4.4, we can quickly prove communication lower bounds for some functions of interest. For example, recall that when f is the Equality function, the matrix $M(f)$ is the identity. The key observation about this matrix is: *a monochromatic rectangle that includes a "1" contains only one element.* The reason is simple: such a rectangle is not allowed to contain any 0's since it is monochromatic, and if it included a second 1 it would pick up some 0-entries as well (recall that rectangles are closed under "mix and match"). Since there are 2^n 1's in the matrix, every covering by monochromatic rectangles (even of just the 1's) has size 2^n.

Corollary 4.5 *The deterministic communication complexity of* Equality *is at least* n.[4]

The exact same argument gives the same lower bound for the Greater-Than function.

Corollary 4.6 *The deterministic communication complexity of* Greater-Than *is at least* n.

We can generalize this argument as follows. A *fooling set* for a function f is a subset $F \subseteq X \times Y$ of inputs such that:

(i) f is constant on F;

(ii) for each distinct pair $(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2) \in F$, at least one of $(\mathbf{x}_1, \mathbf{y}_2), (\mathbf{x}_2, \mathbf{y}_1)$ has the opposite f-value.

[4]The 0s can be covered using another 2^n monochromatic rectangles, one per row (rectangles need not be "contiguous"!). This gives a lower bound of $n + 1$. The trivial upper has Alice sending her input to Bob and Bob announcing the answer, which is a $(n + 1)$-bit protocol. Analogous "+1" improvements are possible for the other examples in this section.

Since rectangles are closed under the "mix and match" operation, (i) and (ii) imply that every monochromatic rectangle contains at most one element of F.

Corollary 4.7 *If F is a fooling set for f, then the deterministic communication complexity of f is at least $\log_2 |F|$.*

For Equality and Greater-Than, we were effectively using the fooling set $F = \{(\mathbf{x}, \mathbf{x}) : \mathbf{x} \in \{0,1\}^n\}$.

The fooling set method is powerful enough to prove a strong lower bound on the deterministic communication complexity of Disjointness.

Corollary 4.8 *The deterministic communication complexity of* Disjointness *is at least n.*

Proof: Take $F = \{(\mathbf{x}, \mathbf{1} - \mathbf{x}) : \mathbf{x} \in \{0,1\}^n\}$ — or in set notation, $\{(S, S^c) : S \subseteq \{1, 2, \ldots, n\}\}$. The set F is a fooling set — it obviously consists only of "yes" inputs of Disjointness, while for every $S \neq T$, either $S \cap T^c \neq \emptyset$ or $T \cap S^c \neq \emptyset$ (or both). See Figure 4.5. Since $|F| = 2^n$, Corollary 4.7 completes the proof. ■

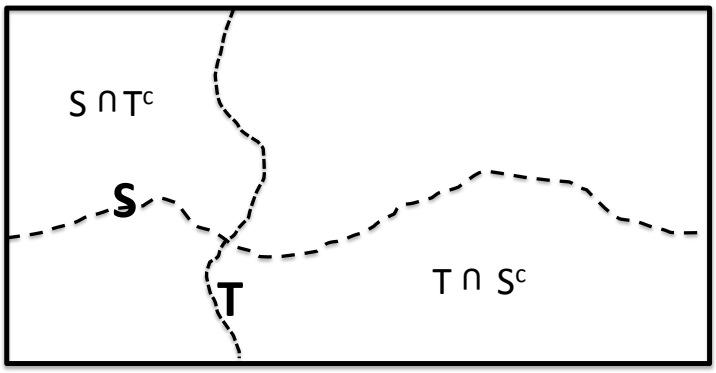


Figure 4.5 If S and T are different sets, then either S and T^c or T and S^c are not disjoint.

4.2.6 Take-Aways

A key take-away point from this section is that, using covering arguments, we can prove the lower bounds that we want on the deterministic communication complexity of many functions of interest. These lower bounds apply also to nondeterministic protocols (discussed next week) and randomized protocols with 1-sided error.

As with one-way communication complexity, proving stronger lower bounds that apply also to randomized protocols with two-sided error is more challenging. Since we're usually

perfectly happy with a good randomized algorithm — recall the F_2 estimation algorithm from Section 1.4 — such lower bounds are very relevant for algorithmic applications. They are our next topic.

4.3 Randomized Protocols

4.3.1 Default Parameter Settings

Our discussion of randomized one-way communication protocols in Section 2.2 remains equally relevant for general protocols. Our "default parameter settings" for such protocols will be the same.

Public coins. By default, we work with public-coin protocols, where Alice and Bob have shared randomness in the form of an infinite sequence of perfectly random bits written on a blackboard in public view. Such protocols are more powerful than private-coin protocols, but not by much (Theorem 4.9). Recall that public-coin randomized protocols are equivalent to distributions over deterministic protocols.

Two-sided error. We allow a protocol to error with constant probability ($\frac{1}{3}$ by default), whether or not the correct answer is "1" or "0." This is the most permissive error model.

Arbitrary constant error probability. Recall that all constant error probabilities in $(0, \frac{1}{2})$ are the same — changing the error changes the randomized communication complexity by only a constant factor (by the usual "independent trials" argument, detailed in the exercises). Thus for upper bounds, we'll be content to achieve error 49%; for lower bounds, it is enough to rule out low-communication protocols with error %1.

Worst-case communication. We define the communication cost of a randomized protocol as the maximum number of bits ever communicated, over all choices of inputs and coin flips. Measuring the expected communication (over the protocol's coin flips) could reduce the communication complexity of a problem, but only by a constant factor.

4.3.2 Newman's Theorem: Public- vs. Private-Coin Protocols

We mentioned a few times that, for our purposes, it usually won't matter whether we consider public-coin or private-coin randomized protocols. What we meant is the following result.

Theorem 4.9 (Newman's Theorem (1991)) *If there is a public-coin protocol for a function f with n-bit inputs that has two-sided error $1/3$ and communication cost c, then there is a private-coin protocol for the problem that has two-sided error $1/3$ and communication cost $O(c + \log n)$.*

Thus, for problems with public-coin randomized communication complexity $\Omega(\log n)$, like most of the problems that we'll study in this course, there is no difference between the

communication complexity of the public-coin and private-coin variants (modulo constant factors).

An interesting exception is EQUALITY. Last lecture, we gave a public-coin protocol — one-way, even — with constant communication complexity. Theorem 4.9 only implies an upper bound of $O(\log n)$ communication for private-coin protocols. (One can also give such a private-coin protocol directly, see the Exercises.) There is also a matching lower bound of $\Omega(\log n)$ for the private-coin communication complexity of EQUALITY. (This isn't very hard to prove, but we won't have an occasion to do it.) Thus public-coin protocols can save $\Theta(\log n)$ bits of communication over private-coin protocols, but no more.

Proof of Theorem 4.9: Let P denote a public-coin protocol with two-sided error 1/3. We begin with a thought experiment. Fix an input $(\mathbf{x}, \mathbf{y})$, with $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$. If we run P on this input, a public string $\mathbf{r}_1$ of random bits is consumed and the output of the protocol is correct with probability at least 2/3. If we run it again, a second (independent) random string $\mathbf{r}_2$ is consumed and another (independent) answer is given, again correct with probability at least 2/3. After t such trials and the consumption of random strings $\mathbf{r}_1, \ldots, \mathbf{r}_t$, P produces t answers. We expect at least 2/3 of these to be correct, and Chernoff bounds (with $\delta = \Theta(1)$ and $\mu = \Theta(t)$) imply that at least 60% of these answers are correct with probability at least $1 - \exp\{-\Theta(t)\}$.

We continue the thought experiment by taking a Union Bound over the $2^n \cdot 2^n = 2^{2n}$ choices of the input $(\mathbf{x}, \mathbf{y})$. With probability at least $1 - 2^{2n} \cdot \exp\{-\Theta(t)\}$ over the choice of $\mathbf{r}_1, \ldots, \mathbf{r}_t$, for every input $(\mathbf{x}, \mathbf{y})$, running the protocol P with these random strings yields at least $.6t$ (out of t) correct answers. In this event, the single sequence $\mathbf{r}_1, \ldots, \mathbf{r}_t$ of random strings "works" simultaneously for all inputs $(\mathbf{x}, \mathbf{y})$. Provided we take $t = cn$ with a large enough constant c, this probability is positive. In particular, such a set $\mathbf{r}_1, \ldots, \mathbf{r}_t$ of random strings exist.

Here is the private-coin protocol.

(0) Before receiving their inputs, Alice and Bob agree on a set of strings $\mathbf{r}_1, \ldots, \mathbf{r}_t$ with the property that, for every input $(\mathbf{x}, \mathbf{y})$, running P t times with the random strings $\mathbf{r}_1, \ldots, \mathbf{r}_t$ yields at least 60% correct answers.

(1) Alice picks an index $i \in \{1, 2, \ldots, t\}$ uniformly at random and sends it to Bob. This requires $\approx \log_2 t = \Theta(\log n)$ bit of communication (recall $t = \Theta(n)$).

(2) Alice and Bob simulate the private-coin protocol P as if they had public coins given by $\mathbf{r}_i$.

By the defining property of the $\mathbf{r}_i$'s, this (private-coin) protocol has error 40%. As usual, this can be reduced to 1/3 via a constant number of independent repetitions followed by taking the majority answer. The resulting communication cost is $O(c + \log n)$, as claimed. ■

We stated and proved Theorem 4.9 for general protocols, but the exact same statement holds (with the same proof) for the one-way protocols that we studied in Lectures 1– 3.

4.3.3 Distributional Complexity

Randomized protocols are significantly harder to reason about than deterministic ones. For example, we've seen that a deterministic protocol can be thought of as a partition of the input space into rectangles. A randomized protocol is a distribution over such partitions. While a deterministic protocol that computes a function f induces only monochromatic rectangles, this does not hold for randomized protocols (which can err with some probability).

We can make our lives somewhat simpler by using Yao's Lemma to translate distributional lower bounds for deterministic protocols to worst-case lower bounds for randomized protocols. Recall the lemma from Lecture 2 (Lemma 2.3).

Lemma 4.10 (Yao 1983) *Let D be a distribution over the space of inputs $(\mathbf{x}, \mathbf{y})$ to a communication problem, and $\epsilon \in (0, \frac{1}{2})$. Suppose that every deterministic protocol P with*

$$\boldsymbol{Pr}_{(\mathbf{x},\mathbf{y})\sim D}[P \text{ wrong on } (\mathbf{x}, \mathbf{y})] \leq \epsilon$$

has communication cost at least k. Then every (public-coin) randomized protocol R with (two-sided) error at most ϵ on every input has communication cost at least k.

We proved Lemma 2.3 in Lecture 2 for one-way protocols, but the same proof holds verbatim for general communication protocols. Like in the one-way case, Lemma 2.3 is a "complete" proof technique — whatever the true randomized communication complexity, there is a hard distribution D over inputs that can in principle be used to prove it (recall the Exercises).

Summarizing, proving lower bounds for randomized communication complexity reduces to:

1. Figuring out a "hard distribution" D over inputs.
2. Proving that every low-communication deterministic protocol has large error w.r.t. inputs drawn from D.

Of course, this is usually easier said than done.

4.3.4 Case Study: Disjointness

Overview

We now return to the Disjointness problem. In Lecture 2 we proved that the *one-way* randomized communication complexity of this problem is linear (Theorem 2.2). We did this by reducing Index to Disjointness— the former is just a special case of the latter, where

one player has a singleton set (i.e., a standard basis vector). We used Yao's Lemma (with D the uniform distribution) and a counting argument (about the volume of small-radius balls in the Hamming cube, remember?) to prove that the one-way randomized communication complexity of INDEX is $\Omega(n)$. Unfortunately, for general communication protocols, the communication complexity of INDEX is obviously $O(\log n)$ — Bob can just send his index i to Alice using $\approx \log_2 n$ bits, and Alice can compute the function. So, it's back to the drawing board.

The following is a major and useful technical achievement.

Theorem 4.11 (Kalyanasundaram and Schnitger 1992; Razborov 1992) *The randomized communication complexity of* DISJOINTNESS *is* $\Omega(n)$.

Theorem 4.11 was originally proved in Kalyanasundaram and Schnitger (1992); the simplified proof in Razborov (1992) has been more influential. More recently, all the cool kids prove Theorem 4.11 using "information complexity" arguments; see Bar-Yossef et al. (2002a).

If you only remember one result from the entire field of communication complexity, it should be Theorem 4.11. The primary reason is that the problem is unreasonably effective for proving lower bounds for other algorithmic problems — almost every subsequent lecture will include a significant example. Indeed, many algorithm designers simply use Theorem 4.11 as a "black box" to prove lower bounds for other problems, without losing sleep over its proof.[5,6] As a bonus, proofs of Theorem 4.11 tend to showcase techniques that are reusable in other contexts.

For a trivial consequence of Theorem 4.11 — see future lectures for less obvious ones — let's return to the setting of streaming algorithms. Lectures 1 and 2 considered only one-pass algorithms. In some contexts, like a telescope that generates an exobyte of data per day, this is a hard constraint. In other settings, like database applications, a small constant number of passes over the data might be feasible (as an overnight job, for example). Communication complexity lower bounds for one-way protocols say nothing about two-pass algorithms, while those for general protocols do. Using Theorem 4.11, all of our $\Omega(m)$ space lower bounds for 1-pass algorithms become $\Omega(m/p)$ space lower bounds for p-pass algorithms, via the same reductions.[7] For example, we proved such lower bounds for computing F_∞, the highest frequency of an element, even with randomization and approximation, and for computing F_0 or F_2 exactly, even with randomization.

So how would one go about proving Theorem 4.11? Recall that Yao's Lemma reduces the proof to exhibiting a hard distribution D (a bit of dark art) over inputs and proving

[5]Similar to, for example, the PCP Theorem and the Parallel Repetition Theorem in the context of hardness of approximation (see e.g. Arora and Lund (1997)).

[6]There's no shame in this — life is short and there's lots of theorems that need proving.

[7]A p-pass space-s streaming algorithm S induces a communication protocol with $O(ps)$ communication, where Alice and Bob turn their inputs into data streams, repeatedly feed them into S, repeatedly sending the memory state of S back and forth to continue the simulation.

that all low-communication deterministic protocols have large error with respect to D (a potentially tough math problem). We next discuss each of these steps in turn.

Choosing a Hard Distribution

The uniform distribution over inputs is not a hard distribution for DISJOINTNESS. What is the probability that a random input $(\mathbf{x}, \mathbf{y})$ satisfies $f(\mathbf{x}, \mathbf{y}) = 1$? Independently in each coordinate i, there is a 25% probability that $x_i = y_i = 1$. Thus, $f(\mathbf{x}, \mathbf{y}) = 1$ with probability $(3/4)^n$. This means that the zero-communication protocol that always outputs "not disjoint" has low error with respect to this distribution. The moral is that a hard distribution D must, at the very least, have a constant probability of producing both "yes" and "no" instances.

The next idea, motivated by the Birthday Paradox, is to define D such that each of Alice and Bob receive a random subset of $\{1, 2, \ldots, n\}$ of size $\approx \sqrt{n}$. Elementary calculations show that a random instance $(\mathbf{x}, \mathbf{y})$ from D has a constant probability of satisfying each of $f(\mathbf{x}, \mathbf{y}) = 1$ and $f(\mathbf{x}, \mathbf{y}) = 0$.

An obvious issue with this approach is that there is a trivial deterministic protocol that uses $O(\sqrt{n} \log n)$ communication and has zero error: Alice (say) just sends her whole input to Bob by describing each of her $\sqrt{n}$ elements explicitly by name ($\approx \log_2 n$ bits each). So there's no way to prove a linear communication lower bound using this distribution. Babai et al. (1986) prove that one can at least prove a $\Omega(\sqrt{n})$ communication lower bound using this distribution, which is already quite a non-trivial result (more on this below). They also showed that for every product distribution D — meaning whenever the random choices of $\mathbf{x}$ and of $\mathbf{y}$ are independent — there is a zero-error deterministic protocol that uses only $O(\sqrt{n} \log n)$ bits of communication (see the Exercises).[8]

Summarizing, if we believe that DISJOINTNESS really requires $\Omega(n)$ communication to solve via randomized protocols, then we need to find a distribution D that meets all of the following criteria.

1. There is a constant probability that $f(\mathbf{x}, \mathbf{y}) = 1$ and that $f(\mathbf{x}, \mathbf{y}) = 0$. (Otherwise, a constant protocol works.)

2. Alice and Bob need to usually receive inputs that correspond to sets of size $\Omega(n)$. (Otherwise, one player can explicitly communicate his or her set.)

3. The random inputs $\mathbf{x}$ and $\mathbf{y}$ are correlated. (Otherwise, the upper bound from Babai et al. (1986) applies.)

4. It must be mathematically tractable to prove good lower bounds on the error of all deterministic communication protocols that use a sublinear amount of communication.

[8] This does not imply that a linear lower bound is impossible. The proof of the converse of Lemma 2.3 — that a tight lower bound on the randomized communication complexity of a problem can always be proved through a distributional lower bound for a suitable choice of D — generally makes use of distributions in which the choices of $\mathbf{x}$ and $\mathbf{y}$ are correlated.

Razborov (1992) proposed a distribution that obviously satisfies the first three properties and, less obviously, also satisfies the fourth. It is:

1. With probability 75%:

   a) $(\mathbf{x}, \mathbf{y})$ is chosen uniformly at random subject to:

      i. $\mathbf{x}, \mathbf{y}$ each have exactly $n/4$ 1's;
      ii. there is no index $i \in \{1, 2, \ldots, n\}$ with $x_i = y_i = 1$ (so $f(\mathbf{x}, \mathbf{y}) = 1$).

2. With probability 25%:

   a) $(\mathbf{x}, \mathbf{y})$ is chosen uniformly at random subject to:

      i. $\mathbf{x}, \mathbf{y}$ each have exactly $n/4$ 1's;
      ii. there is exactly one index $i \in \{1, 2, \ldots, n\}$ with $x_i = y_i = 1$ (so $f(\mathbf{x}, \mathbf{y}) = 0$).

Note that in both cases, the constraint on the number of indices i with $x_i = y_i = 0$ creates correlation between the choices of $\mathbf{x}$ and $\mathbf{y}$.

Proving Error Lower Bounds via Corruption Bounds

Even if you're handed a hard distribution over inputs, there remains the challenging task of proving a good error lower bound on low-communication deterministic protocols. There are multiple methods for doing this, with the *corruption method* being the most successful one so far. We outline this method next.

At a high level, the corruption method is a natural extension of the covering arguments of Section 4.2 to protocols that can err. Recall that for deterministic protocols, the covering approach argues that every covering of the matrix $M(f)$ of the function f by monochromatic rectangles requires a lot of rectangles. In our examples, we only bothered to argue about the 1-inputs of the function.[9] We'll do something similar here, weighted by the distribution D and allowing errors — arguing that there's significant mass on the 1-inputs of f, and that a lot of nearly monochromatic rectangles are required to cover them all.

Precisely, suppose you have a distribution D over the inputs of a problem so that the "1-mass" of D, meaning $\mathbf{Pr}_{(\mathbf{x},\mathbf{y})\sim D}[f(x, y) = 1]$, is at least a constant, say .5. The plan is to prove two properties.

(1) For every deterministic protocol P with error at most a sufficiently small constant ϵ, at least 25% of the 1-mass of D is contained in "almost monochromatic 1-rectangles" of P (defined below). We'll see below that this is easy to prove in general by an averaging argument.

[9] Since f has only two outputs, it's almost without loss to pick a single output $z \in \{0, 1\}$ of f and lower bound only the number of monochromatic rectangles needed to cover all of the z's.

(2) An almost monochromatic 1-rectangle contains at most 2^{-c} mass of the distribution D, where c is as large as possible (ideally $c = \Omega(n)$). This is the hard step, and the argument will be different for different functions f and different input distributions D.

If we can establish (1) and (2), then we have a lower bound of $\Omega(2^{-c})$ on the number of rectangles induced by P, which proves that P uses communication $\Omega(c)$.[10]

Here's the formal definition of an *almost monochromatic 1-rectangle (AM1R)* $R = A \times B$ of a matrix $M(f)$ with respect to an input distribution D:

$$\mathbf{Pr}_{(\mathbf{x},\mathbf{y})\sim D}[(\mathbf{x},\mathbf{y}) \in R \text{ and } f(\mathbf{x},\mathbf{y}) = 0] \leq 8\epsilon \cdot \mathbf{Pr}_{(\mathbf{x},\mathbf{y})\sim D}[(\mathbf{x},\mathbf{y}) \in R \text{ and } f(\mathbf{x},\mathbf{y}) = 1]\,. \quad (4.2)$$

Here's why property (1) is true in general. Let P be a deterministic protocol with error at most ϵ with respect to D. Since P is deterministic, it partitions the matrix $M(f)$ into rectangles, and in each rectangle, P's output is constant. Let $R_1, \ldots, R_\ell$ denote the rectangles in which P outputs "1."

At least 50% of the 1-mass of D — and hence at least 25% of D's overall mass — must be contained in $R_1, \ldots, R_\ell$. For if not, on at least 25% of the mass of D, $f(\mathbf{x},\mathbf{y}) = 1$ while P outputs "0". This contradicts the assumption that P has error ϵ with respect to D (provided $\epsilon < .25$).

Also, at least 50% of the mass in $R_1, \ldots, R_\ell$ must lie in AM1Rs. For if not, using (4.2) and the fact that the total mass in $R_1, \ldots, R_\ell$ is at least .25, it would follow that D places more than $8\epsilon \cdot .125 = \epsilon$ mass on 0-inputs in $R_1, \ldots, R_\ell$. Since P outputs "1" on all of these inputs, this contradicts the assumption that P has error at most ϵ with respect to D. This completes the proof of step (1), which applies to every problem and every distribution D over inputs with 1-mass at least .5.

Step (2) is difficult and problem-specific. Babai et al. (1986), for their input distribution D over DISJOINTNESS inputs mentioned above, gave a proof of step (2) with $c = \Omega(\sqrt{n})$, thus giving an $\Omega(\sqrt{n})$ lower bound on the randomized communication complexity of the problem. Razborov (1992) gave, for his input distribution, a proof of step (2) with $c = \Omega(n)$, implying the desired lower bound for DISJOINTNESS. Sadly, we won't have time to talk about these and subsequent proofs (as in Bar-Yossef et al. (2002a)); perhaps in a future course.

[10] Why call it the "corruption method"? Because the argument shows that, if a deterministic protocol has low communication, then most of its induced rectangles that contain 1-inputs are also "corrupted" by lots of 0-inputs — its rectangles are so big that (4.2) fails. In turn, this implies large error.

Lecture 5

Lower Bounds for the Extension Complexity of Polytopes

5.1 Linear Programs, Polytopes, and Extended Formulations

5.1.1 Linear Programs for Combinatorial Optimization Problems

You've probably seen some polynomial-time algorithms for the problem of computing a maximum-weight matching of a bipartite graph.[1] Many of these, like the Kuhn-Tucker algorithm (Kuhn, 1955), are "combinatorial algorithms" that operate directly on the graph.

Linear programming is also an effective tool for solving many discrete optimization problems. For example, consider the following linear programming relaxation of the maximum-weight bipartite matching problem (for a weighted bipartite graph $G = (U, V, E, w)$):

$$\max \sum_{e \in E} w_e x_e \tag{5.1}$$

subject to

$$\sum_{e \in \delta(v)} x_e \leq 1 \tag{5.2}$$

for every vertex $v \in U \cup V$ (where $\delta(v)$ denotes the edges incident to v) and

$$x_e \geq 0 \tag{5.3}$$

for every edge $e \in E$.

In this formulation, each decision variable x_e is intended to encode whether an edge e is in the matching ($x_e = 1$) or not ($x_e = 0$). It is easy to verify that the vectors of $\{0, 1\}^E$ that satisfy the constraints (5.2) and (5.3) are precisely the characteristic vectors of the matchings of G, with the objective function value of the solution to the linear program equal to the total weight of the matching.

Since every characteristic vector of a matching satisfies (5.2) and (5.3), and the set of feasible solutions to the linear system defined by (5.2) and (5.3) is convex, the convex hull

[1] Recall that a graph is *bipartite* if its vertex set can be partitioned into two sets U and V such that every edge has one endpoint in each of U, V. Recall that a *matching* of a graph is a subset of edges that are pairwise disjoint.

of the characteristic vectors of matchings is contained in this feasible region.[2] Also note that every characteristic vector $\mathbf{x}$ of a matching is a vertex[3] of this feasible region — since all feasible solutions have all coordinates bounded by 0 and 1, the 0-1 vector $\mathbf{x}$ cannot be written as a non-trivial convex combination of other feasible solutions. The worry is does this feasible region contain anything other than the convex hull of characteristic vectors of matchings? Equivalently, does it have any vertices that are fractional, and hence do not correspond to matchings? (Note that integrality is not explicitly enforced by (5.2) or (5.3).)

A nice fact is that the vertices of the feasible region defined by (5.2) and (5.3) are precisely the characteristic vectors of matchings of G. This is equivalent to the Birkhoff-von Neumann theorem (see Exercises). There are algorithms that solve linear programs in polynomial time (and output a vertex of the feasible region, see e.g. Grötschel et al. (1988)), so this implies that the maximum-weight bipartite matching problem can be solved efficiently using linear programming.

How about the more general problem of maximum-weight matching in general (non-bipartite) graphs? While the same linear system (5.2) and (5.3) still contains the convex hull of all characteristic vectors of matchings, and these characteristic vectors are vertices of the feasible region, there are also other, fractional, vertices. To see this, consider the simplest non-bipartite graph, a triangle. Every matching contains at most 1 edge. But assigning $x_e = \frac{1}{2}$ for each of the edges e yields a fractional solution that satisfies (5.2) and (5.3). This solution clearly cannot be written as a convex combination of characteristic vectors of matchings.

It is possible to add to (5.2)–(5.3) additional inequalities — "odd cycle inequalities" stating that, for every odd cycle C of G, $\sum_{e \in C} x_e \leq (|C| - 1)/2$ — so that the resulting smaller set of feasible solutions is precisely the convex hull of the characteristic vectors of matchings. Unfortunately, many graphs have an exponential number of odd cycles. Is it possible to add only a polynomial number of inequalities instead? Unfortunately not — the convex hull of the characteristic vectors of matchings can have $2^{\Omega(n)}$ "facets" (Pulleyblank and Edmonds, 1974).[4] We define facets more formally in Section 5.3.1, but intuitively they are the "sides" of a polytope,[5] like the $2n$ sides of an n-dimensional cube. It is intuitively

[2]Recall that a set $S \subseteq \mathbb{R}^n$ is *convex* if it is "filled in," with $\lambda \mathbf{x} + (1 - \lambda)\mathbf{y} \in S$ whenever $\mathbf{x}, \mathbf{y} \in S$ and $\lambda \in [0, 1]$. Recall that the *convex hull* of a point set $P \subseteq \mathbb{R}^n$ is the smallest (i.e., intersection of all) convex set that contains it. Equivalently, it is the set of all finite convex combinations of points of P, where a convex combination has the form $\sum_{i=1}^{p} \lambda_i \mathbf{x}_i$ for non-negative λ_i's summing to 1 and $\mathbf{x}_1, \ldots, \mathbf{x}_p \in P$.

[3]There is an unfortunate clash of terminology when talking about linear programming relaxations of combinatorial optimization problems: a "vertex" might refer to a node of a graph or to a "corner" of a geometric set.

[4]This linear programming formulation still leads to a polynomial-time algorithm, but using fairly heavy machinery — the "ellipsoid method" (Khachiyan, 1979) and a "separation oracle" for the odd cycle inequalities (Padberg and Rao, 1982). There are also polynomial-time combinatorial algorithms for (weighted) non-bipartite matching, beginning with Edmonds (1965).

[5]A *polytope* is just a high-dimensional polygon — an intersection of halfspaces that is bounded or, equivalently, the convex hull of a finite set of points.

clear that a polytope with ℓ facets needs ℓ inequalities to describe — it's like cleaving a shape out of marble, with each inequality contributing a single cut. We conclude that there is no linear program with variables $\{x_e\}_{e \in E}$ of polynomial size that captures the maximum-weight (non-bipartite) matching problem.

5.1.2 Auxiliary Variables and Extended Formulations

The exponential lower bound above on the number of linear inequalities needed to describe the convex hull of characteristic vectors of matchings of a non-bipartite graph applies to linear systems in $\mathbb{R}^E$, with one dimension per edge. The idea of an *extended formulation* is to add a polynomial number of auxiliary decision variables, with the hope that radically fewer inequalities are needed to describe the region of interest in the higher-dimensional space.

This idea might sound like grasping at straws, but sometimes it actually works. For example, fix a positive integer n, and represent a permutation $\pi \in S_n$ by the n-vector $\mathbf{x}_\pi = (\pi(1), \pi(2), \ldots, \pi(n))$, with all coordinates in $\{1, 2, \ldots, n\}$. The *permutahedron* is the convex hull of all $n!$ such vectors. The permutahedron is known to have $2^{n/2} - 2$ facets (see e.g. Goemans (2014)), so a polynomial-sized linear description would seem out of reach.

Suppose we add n^2 auxiliary variables, y_{ij} for all $i, j \in \{1, 2, \ldots, n\}$. The intent is for y_{ij} to be a 0-1 variable that indicates whether or not $\pi(i) = j$ — in this case, the y_{ij}'s are the entries of the $n \times n$ permutation matrix that corresponds to π.

We next add a set of constraints to enforce the desired semantics of the y_{ij}'s (cf., (5.2) and (5.3)):

$$\sum_{j=1}^{n} y_{ij} \leq 1 \tag{5.4}$$

for $i = 1, 2, \ldots, n$;

$$\sum_{i=1}^{n} y_{ij} \leq 1 \tag{5.5}$$

for $j = 1, 2, \ldots, n$; and

$$y_{ij} \geq 0 \tag{5.6}$$

for all $i, j \in \{1, 2, \ldots, n\}$. We also add constraints that enforce consistency between the permutation encoded by the x_i's and by the y_{ij}'s:

$$x_i = \sum_{j=1}^{n} j y_{ij} \tag{5.7}$$

for all $i = 1, 2, \ldots, n$.

It is straightforward to check that the vectors $\mathbf{y} \in \{0, 1\}^{n^2}$ that satisfy (5.4)–(5.6) are precisely the permutation matrices. For such a $\mathbf{y}$ corresponding to a permutation π,

the constraints (5.7) force the x_i's to encode the same permutation π. Using again the Birkhoff-von Neumann Theorem, every vector $\mathbf{y} \in \mathbb{R}^{n^2}$ that satisfies (5.4)–(5.6) is a convex combination of permutation matrices (see Exercises). Constraint (5.7) implies that the x_i's encode the same convex combination of permutations. Thus, if we take the set of solutions in $\mathbb{R}^{n+n^2}$ that satisfy (5.4)–(5.7) and project onto the x-coordinates, we get exactly the permutahedron. This is what we mean by an *extended formulation* of a polytope.

To recap the remarkable trick we just pulled off: blowing up the number of variables from n to $n + n^2$ reduced the number of inequalities needed from $2^{n/2}$ to $n^2 + 3n$. This allows us to optimize a linear function over the permutahedron in polynomial time. Given a linear function (in the x_i's), we optimize it over the (polynomial-size) extended formulation, and retain only the x-variables of the optimal solution.

Given the utility of polynomial-size extended formulations, we'd obviously like to understand which problems have them. For example, does the non-bipartite matching problem admit such a formulation? The goal of this lecture is to develop communication complexity-based techniques for ruling out such polynomial-size extended formulations. We'll prove an impossibility result for the "correlation polytope" (Fiorini et al., 2015); similar (but much more involved) arguments imply that every extended formulation of the non-bipartite matching problem requires an exponential number of inequalities (Rothvoß, 2014).

Remark 5.1 (Geometric Intuition) It may seem surprising that adding a relatively small number of auxiliary variables can radically reduce the number of inequalities needed to describe a set — described in reverse, that projecting onto a subset of variables can massively blow up the number of sides. It's hard to draw (low-dimensional) pictures that illustrate this point. If you play around with projections of some three-dimensional polytopes onto the plane, you'll observe that non-facets of the high-dimensional polytope (edges) often become facets (again, edges) in the low-dimensional projection. Since the number of lower-dimensional faces of a polytope can be much bigger than the number of facets — already in the 3-D cube, there are 12 edges and only 6 sides — it should be plausible that a projection could significantly increase the number of facets.

5.2 Nondeterministic Communication Complexity

The connection between extended formulations of polytopes and communication complexity involves *nondeterministic* communication complexity. We studied this model implicitly in parts of Lecture 4; this section makes the model explicit.

Consider a function $f : X \times Y \to \{0, 1\}$ and the corresponding 0-1 matrix $M(f)$, with rows indexed by Alice's possible inputs and columns indexed by Bob's possible inputs. In Lecture 4 we proved that if every covering of $M(f)$ by monochromatic rectangles[6] requires

[6]Recall that a *rectangle* is a subset $S \subseteq X \times Y$ that has a product structure, meaning $S = A \times B$ for some $A \subseteq X$ and $B \subseteq Y$. Equivalently, S is closed under "mix and match:" whenever $(\mathbf{x}_1, \mathbf{y}_1)$ and $(\mathbf{x}_2, \mathbf{y}_2)$

at least t rectangles, then the deterministic communication complexity of f is at least $\log_2 t$ (Theorem 4.3). The reason is that every communication protocol computing f with communication cost c induces a partition of $M(f)$ into at most 2^c monochromatic rectangles, and partitions are a special case of coverings. See also Figure 5.1.

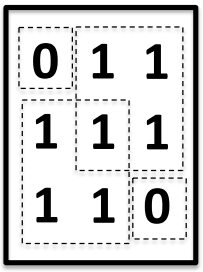


Figure 5.1 A covering by four monochromatic rectangles that is not a partition.

Communication complexity lower bounds that are proved through coverings are actually much stronger than we've let on thus far — they apply also to *nondeterministic* protocols, which we define next.

You presumably already have a feel for nondeterminism from your study of the complexity class NP. Recall that one way to define NP is as the problems for which membership can be verified in polynomial time. To see how an analog might work with communication protocols, consider the complement of the EQUALITY problem, ¬EQUALITY. If a third party wanted to convince Alice and Bob that their inputs $\mathbf{x}$ and $\mathbf{y}$ are different, it would not be difficult: just specify an index $i \in \{1, 2, \ldots, n\}$ for which $x_i \neq y_i$. Specifying an index requires $\log_2 n$ bits, and specifying whether or not $x_i = 0$ and $y_i = 1$ or $x_i = 1$ and $y_i = 0$ requires one additional bit. Given such a specification, Alice and Bob can check the correctness of this "proof of non-equality" without any communication. If $\mathbf{x} \neq \mathbf{y}$, there is always a $(\log_2 +1)$-bit proof that will convince Alice and Bob of this fact; if $\mathbf{x} = \mathbf{y}$, then no such proof will convince Alice and Bob otherwise. This means that ¬EQUALITY has *nondeterministic communication complexity* at most $\log_2 n + 1$.

Coverings of $M(f)$ by monochromatic rectangles are closely related to the nondeterministic communication complexity of f. We first show how coverings lead to nondeterministic protocols. It's easiest to formally define such protocols after the proof.

Proposition 5.2 *Let $f : X \times Y \to \{0, 1\}$ be a Boolean function and $M(f)$ the corresponding matrix. If there is a cover of the 1-entries of $M(f)$ by t 1-rectangles, then there is a nondeterministic protocol that verifies $f(\mathbf{x}, \mathbf{y}) = 1$ with cost $\log_2 t$.*

---

are in S, so are $(\mathbf{x}_1, \mathbf{y}_2)$ and $(\mathbf{x}_2, \mathbf{y}_1)$. A rectangle is *monochromatic* (w.r.t. f) if it contains only 1-entries of $M(f)$ or only 0-entries of $M(f)$. In these cases, we call it a *1-rectangle* or a *0-rectangle*, respectively.

Proof: Let $R_1, \ldots, R_t$ denote a covering of the 1s of $M(f)$ by 1-rectangles. Alice and Bob can agree to this covering in advance of receiving their inputs. Now consider the following scenario:

1. A *prover* — a third party — sees both inputs $\mathbf{x}$ *and* $\mathbf{y}$. (This is the formal model used for nondeterministic protocols.)

2. The prover writes an index $i \in \{1, 2, \ldots, t\}$ — the name of a rectangle R_i — on a blackboard, in public view. Since R_i is a rectangle, it can be written as $R_i = A_i \times B_i$ with $A_i \subseteq X$, $B_i \subseteq Y$.

3. Alice accepts if and only if $\mathbf{x} \in A_i$.

4. Bob accepts if and only if $\mathbf{y} \in B_i$.

This protocol has the following properties:

1. If $f(\mathbf{x}, \mathbf{y}) = 1$, then there exists a proof such that Alice and Bob both accept. (Since $f(\mathbf{x}, \mathbf{y}) = 1$, $(\mathbf{x}, \mathbf{y}) \in R_i$ for some i, and Alice and Bob both accept if "i" is written on the blackboard.)

2. If $f(\mathbf{x}, \mathbf{y}) = 0$, there is no proof that both Alice and Bob accept. (Whatever index $i \in \{1, 2, \ldots, t\}$ is written on the blackboard, since $f(\mathbf{x}, \mathbf{y}) = 0$, either $\mathbf{x} \notin R_i$ or $y \notin R_i$, causing a rejection.)

3. The maximum length of a proof is $\log_2 t$. (A proof is just an index $i \in \{1, 2, \ldots, t\}$.)

These three properties imply, by definition, that the nondeterministic communication complexity of the function f and the output 1 is at most $\log_2 t$. ■

The proof of Proposition 5.2 introduces our formal model of nondeterministic communication complexity: Alice and Bob are given a "proof" or "advice string" by a prover, which can depend on both of their inputs; the communication cost is the worst-case length of the proof; and a protocol is said to compute an output $z \in \{0, 1\}$ of a function f if $f(\mathbf{x}, \mathbf{y}) = z$ if and only if there exists proof such that both Alice and Bob accept.

With nondeterministic communication complexity, we speak about both a function f and an output $z \in \{0, 1\}$. For example, if f is EQUALITY, then we saw that the nondeterministic communication complexity of f and the output 0 is at most $\log_2 n + 1$. Since it's not clear how to convince Alice and Bob that their inputs *are* equal without specifying at least one bit for each of the n coordinates, one might expect the nondeterministic communication complexity of f and the output 1 to be roughly n. (And it is, as we'll see.)

We've defined nondeterministic protocols so that Alice and Bob never speak, and only verify. This is without loss of generality, since given a protocol in which they do speak, one could modify it so that the prover writes on the blackboard everything that they would have

said. We encourage the reader to formalize an alternative definition of nondeterministic protocols without a prover and in which Alice and Bob speak nondeterministically, and to prove that this definition is equivalent to the one we've given above (see Exercises).

Next we prove the converse of Proposition 5.3.

Proposition 5.3 *If the nondeterministic communication complexity of the function f and the output 1 is c, then there is a covering of the 1s of $M(f)$ by 2^c 1-rectangles.*

Proof: Let $\mathcal{P}$ denote a nondeterministic communication protocol for f and the output 1 with communication cost (i.e., maximum proof length) at most c. For a proof ℓ, let $Z(\ell)$ denote the inputs $(\mathbf{x}, \mathbf{y})$ where both Alice and Bob accept the proof. We can write $Z(\ell) = A \times B$, where A is the set of inputs $\mathbf{x} \in X$ of Alice where she accepts the proof ℓ, and B is the set of inputs $\mathbf{y} \in Y$ of Bob where he accepts the proof. By the assumed correctness of $\mathcal{P}$, $f(\mathbf{x}, \mathbf{y}) = 1$ for every $(\mathbf{x}, \mathbf{y}) \in Z(\ell)$. That is, $Z(\ell)$ is a 1-rectangle.

By the first property of nondeterministic protocols, for every 1-input $(\mathbf{x}, \mathbf{y})$ there is a proof such that both Alice and Bob accept. That is, $\cup_\ell Z(\ell)$ is precisely the set of 1-inputs of f — a covering of the 1s of $M(f)$ by 1-rectangles. Since the communication cost of $\mathcal{P}$ is at most c, there are at most 2^c different proofs ℓ. ■

Proposition 5.3 implies that communication complexity lower bounds derived from covering lower bounds apply to nondeterministic protocols.

Corollary 5.4 *If every covering of the 1s of $M(f)$ by 1-rectangles uses at least t rectangles, then the nondeterministic communication complexity of f is at least $\log_2 t$.*

Thus our arguments in Lecture 4, while simple, were even more powerful than we realized — they prove that the nondeterministic communication complexity of Equality, Disjointness, and Greater-Than (all with output 1) is at least n. It's kind of amazing that these lower bounds can be proved with so little work.

5.3 Extended Formulations and Nondeterministic Communication Complexity

What does communication complexity have to do with extended formulations? To forge a connection, we need to show that an extended formulation with few inequalities is somehow useful for solving hard communication problems. While this course includes a number of clever connections between communication complexity and various computational models, this connection to extended formulations is perhaps the most surprising and ingenious one of them all. Superficially, extended formulations with few inequalities can be thought of as "compressed descriptions" of a polytope, and communication complexity is generally useful for ruling out compressed descriptions of various types. It is not at all obvious that this vague intuition can be turned into a formal connection, let alone one that is useful for proving non-trivial impossibility results.

5.3.1 Faces and Facets

We discuss briefly some preliminaries about polytopes. Let P be a polytope in variables $\mathbf{x} \in \mathbb{R}^n$. By definition, an *extended formulation* of P is a set of the form

$$Q = \{(\mathbf{x}, \mathbf{y}) : \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{y} \leq \mathbf{d}\},$$

where $\mathbf{x}$ and $\mathbf{y}$ are the original and auxiliary variables, respectively, such that

$$\{\mathbf{x} : \exists \mathbf{y} \text{ s.t. } (\mathbf{x}, \mathbf{y}) \in Q\} = P.$$

This is, projecting Q onto the original variables $\mathbf{x}$ yields the original polytope P. The extended formulation of the permutahedron described in Section 5.1.2 is a canonical example. The *size* of an extended formulation is the number of inequalities.[7]

Recall that $\mathbf{x} \in P$ is a *vertex* if it cannot be written as a non-trivial convex combination of other points in P. A *supporting hyperplane of* P is a vector $\mathbf{a} \in \mathbb{R}^n$ and scalar $b \in \mathbb{R}$ such that $\mathbf{a}\mathbf{x} = b$ for all $\mathbf{x} \in P$. Every supporting hyperplane $\mathbf{a}, b$ induces a *face* of P, defined as $\{\mathbf{x} \in P : \mathbf{a}\mathbf{x} = b\}$ — the intersection of the boundaries of P and of the the halfspace defined by the supporting hyperplane. (See Figure 5.2.) Note that a face is generally induced by many different supporting hyperplanes. The empty set is considered a face. Note also that faces are nested — in three dimensions, there are vertices, edges, and sides. In general, if f is a face of P, then the vertices of f are precisely the vertices of P that are contained in f.

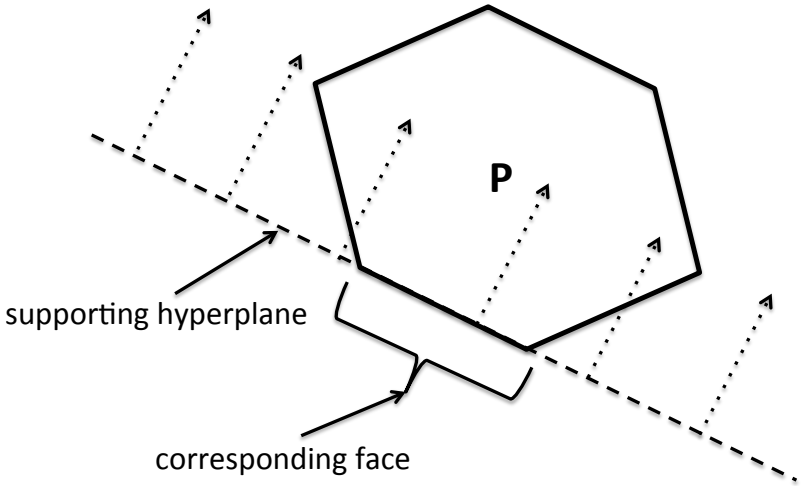


Figure 5.2 A supporting hyperplane of a polytope P and the corresponding face of the polytope.

[7] There is no need to keep track of the number of auxiliary variables — there is no point in having an extended formulation of this type with more variables than inequalities (see Exercises).

A *facet* of P is a maximal face — a face that is not strictly contained in any other face. Provided P has a non-empty interior, its facets are $(n-1)$-dimensional.

There are two different types of finite descriptions of a polytope, and it is useful to go back and forth between them. First, a polytope P equals the convex hull of its vertices. Second, P is the intersection of the halfspaces that define its facets.[8]

5.3.2 Yannakakis's Lemma

What good is a small extended formulation? We next make up a contrived communication problem for which small extended formulations are useful. For a polytope P, in the corresponding Face-Vertex(P) problem, Alice gets a face f of P (in the form of a supporting hyperplane $\mathbf{a}, b$) and Bob gets a vertex v of P. The function $FV(f, v)$ is defined as 1 if v does *not* belong to f, and 0 if $v \in f$. Equivalently, $FV(f, v) = 1$ if and only if $\mathbf{a}^\mathbf{T}\mathbf{v} < b$, where $\mathbf{a}, b$ is a supporting hyperplane that induces f. Polytopes in n dimensions generally have an exponential number of faces and vertices. Thus, trivial protocols for Face-Vertex(P), where one party reports their input to the other, can have communication cost $\Omega(n)$.

A key result is the following.

Lemma 5.5 (Yannakakis's Lemma (1991)) *If the polytope P admits an extended formulation Q with r inequalities, then the nondeterministic communication complexity of* Face-Vertex(P) *is at most* $\log_2 r$.

That is, if we can prove a linear lower bound on the nondeterministic communication complexity of the Face-Vertex(P) problem, then we have ruled out subexponential-size extended formulations of P.

Sections 5.3.3 and 5.3.4 give two different proof sketches of Lemma 5.5. These are roughly equivalent, with the first emphasizing the geometric aspects (following Lovász (1990)) and the second the algebraic aspects (following Yannakakis (1991)). In Section 5.4 we put Lemma 5.5 to use and prove strong lower bounds for a concrete polytope.

Remarkably, Yannakakis (1991) did not give any applications of his lemma — the lower bounds for extended formulations in Yannakakis (1991) are for "symmetric" formulations and proved via direct arguments. Lemma 5.5 was suggested by Yannakakis (1991) as a potentially useful tool for more general impossibility results, and finally in the past five years (beginning with Fiorini et al. (2015)) this prophecy has come to pass.

5.3.3 Proof Sketch of Lemma 5.5: A Geometric Argument

Suppose P admits an extended formulation $Q = \{(\mathbf{x}, \mathbf{y}) : \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{y} \leq \mathbf{d}\}$ with only r inequalities. Both P and Q are known to Alice and Bob before the protocol begins. A first

[8] Proofs of all of these statements are elementary but outside the scope of this lecture; see e.g. Ziegler (1995) for details.

idea is for Alice, who is given a face f of the original polytope P, to tell Bob the name of the "corresponding face" of Q. Bob can then check whether or not his "corresponding vertex" belongs to the named face or not, thereby computing the function.

Unfortunately, knowing that Q is defined by r inequalities only implies that it has at most r *facets* — it can have a very large number of faces. Thus Alice can no more afford to write down an arbitrary face of Q than a face of P.

We use a third-party prover to name a suitable facet of Q than enables Alice and Bob to compute the FACE-VERTEX(P) function; since Q has at most r facets, the protocol's communication cost is only $\log_2 r$, as desired.

Suppose the prover wants to convince Alice and Bob that Bob's vertex v of P does not belong to Alice's face f of P. If the prover can name a facet f^* of Q such that:

(i) there exists $\mathbf{y}_v$ such that $(v, \mathbf{y}_v) \notin f^*$; and

(ii) for every $(\mathbf{x}, \mathbf{y}) \in Q$ with $\mathbf{x} \in f$, $(\mathbf{x}, \mathbf{y}) \in f^*$;

then this facet f^* proves that $v \notin f$. Moreover, given f^*, Alice and Bob can verify (ii) and (i), respectively, without any communication.

All that remains to prove is that, when $v \notin f$, there exists a facet f^* of Q such that (i) and (ii) hold. First consider the inverse image of f in Q, $\tilde{f} = \{(\mathbf{x}, \mathbf{y}) \in Q : \mathbf{x} \in f\}$. Similarly, define $\tilde{v} = \{(v, \mathbf{y}) \in Q\}$. Since $v \notin f$, $\tilde{f}$ and $\tilde{v}$ are disjoint subsets of Q. It is not difficult to prove that $\tilde{f}$ and $\tilde{v}$, as inverse images of faces under a linear map, are faces of Q (exercise). An intuitive but non-trivial fact is that every face of a polytope is the intersection of the facets that contain it.[9] Thus, for every vertex v^* of Q that is contained in $\tilde{v}$ (and hence not in $\tilde{f}$) — and since $\tilde{v}$ is non-empty, there is at least one — we can choose a facet f^* of Q that contains $\tilde{f}$ (property (ii)) but excludes v^* (property (i)). This concludes the proof sketch of Lemma 5.5.

5.3.4 Proof Sketch of Lemma 5.5: An Algebraic Argument

The next proof sketch of Lemma 5.5 is a bit longer but introduces some of the most important concepts in the study of extended formulations.

The *slack matrix* of a polytope P has rows indexed by faces F and columns indexed by vertices V. We identify each face with a canonical supporting hyperplane $\mathbf{a}, b$. Entry S_{fv} of the slack matrix is defined as $b - \mathbf{a}^\mathbf{T}\mathbf{v}$, where $\mathbf{a}, b$ is the supporting hyperplane corresponding to the face f. Observe that all entries of S are nonnegative. Define the *support* $\text{supp}(S)$ of the slack matrix S as the $F \times V$ matrix with 1-entries wherever S has positive entries, and 0-entries wherever S has 0-entries. Observe that $\text{supp}(S)$ is a property only of the polytope P, independent of the choices of the supporting hyperplanes for the faces of P.

[9] This follows from Farkas's Lemma, or equivalently the Separating Hyperplane Theorem. See Ziegler (1995) for details.

Observe also that $\text{supp}(S)$ is precisely the answer matrix for the FACE-VERTEX(P) problem for the polytope P.

We next identify a sufficient condition for FACE-VERTEX(P) to have low nondeterministic communication complexity; later we explain why the existence of a small extended formulation implies this sufficient condition. Suppose the slack matrix S has *nonnegative rank* r, meaning it is possible to write $S = TU$ with T a $|F| \times r$ nonnegative matrix and U a $r \times |V|$ nonnegative matrix (Figure 5.3).[10] Equivalently, suppose we can write S as the sum of r outer products of nonnegative vectors (indexed by F and V):

$$S = \sum_{j=1}^{r} \alpha_j \cdot \beta_j^T, \tag{5.8}$$

where the α_j's correspond to the columns of T and the β_j's to the rows of U.

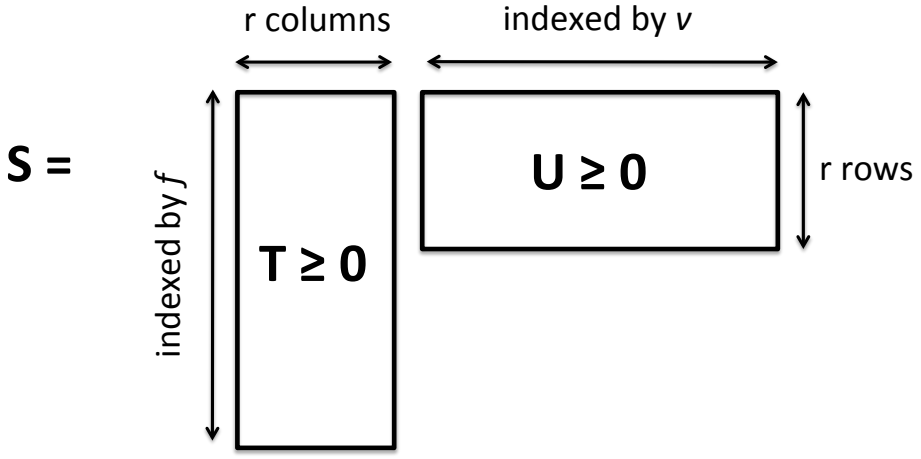


Figure 5.3 A rank-r factorization of the slack matrix S into nonnegative matrices T and U.

We claim that if the slack matrix S of a polytope P has nonnegative rank r, then there is a nondeterministic communication protocol for FACE-VERTEX(P) with cost at most $\log_2 r$. As usual, Alice and Bob can agree to the decomposition (5.8) in advance. A key observation is that, by inspection of (5.8), $S_{fv} > 0$ if and only if there exists some $j \in \{1, 2, \ldots, r\}$ with $\alpha_{fj}, \beta_{jv} > 0$. (We are using here that everything is nonnegative and so no cancellations are possible.) Equivalently, the supports of the outer products $\alpha_j \cdot \beta_j^T$ can be viewed as a covering of the 1-entries of $\text{supp}(S)$ by r 1-rectangles. Given this observation, the protocol for FACE-VERTEX(P) should be clear.

[10] This is called a *nonnegative matrix factorization*. It is the analog of the singular value decomposition (SVD), but with the extra constraint that the factors are nonnegative matrices. It obviously only makes sense to ask for such decompositions for nonnegative matrices (like S).

1. The prover announces an index $j \in \{1, 2, \ldots, r\}$.

2. Alice accepts if and only if the fth component of α_j is strictly positive.

3. Bob accepts if and only if the vth component of β_j is strictly positive.

The communication cost of the protocol is clearly $\log_2 r$. The key observation above implies that there is a proof (i.e., an index $j \in \{1, 2, \ldots, r\}$) accepted by both Alice and Bob if and only if Bob's vertex v does not belong to Alice's face f.

It remains to prove that, whenever a polytope P admits an extended formulation with a small number of inequalities, its slack matrix admits a low-rank nonnegative matrix factorization.[11] We'll show this by exhibiting nonnegative r-vectors λ_f (for all faces f of P) and μ_v (for all vertices v of P) such that $S_{fv} = \lambda_f^T \mu_v$ for all f and v. In terms of Figure 5.3, the λ_f's and μ_v's correspond to the rows of T and columns of U, respectively.

The next task is to understand better how an extended formulation $Q = \{(\mathbf{x}, \mathbf{y}) : \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{y} \leq \mathbf{d}\}$ must be related to the original polytope P. Given that projecting Q onto the variables $\mathbf{x}$ yields P, it must be that every supporting hyperplane of P is logically implied by the inequalities that define Q. To see one way how this can happen, suppose there is a non-negative r-vector $\lambda \in \mathbb{R}^r_+$ with the following properties:

(P1) $\lambda^T \mathbf{C} = \mathbf{a}^T$;

(P2) $\lambda^T \mathbf{D} = \mathbf{0}$;

(P3) $\lambda^T \mathbf{d} = b$.

(P1)–(P3) imply that, for every $(\mathbf{x}, \mathbf{y})$ in Q (and so with $\mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{y} \leq \mathbf{d}$), we have

$$\underbrace{\lambda^T \mathbf{C}}_{=\mathbf{a}^T} \mathbf{x} + \underbrace{\lambda^T \mathbf{D}}_{=\mathbf{0}} \mathbf{y} \leq \underbrace{\lambda^T \mathbf{d}}_{=b}$$

and hence $\mathbf{a}^\mathbf{T}\mathbf{x} \leq b$ (no matter what $\mathbf{y}$ is).

Nonnegative linear combinations λ of the constraints of Q that satisfy (P1)–(P3) are one way in which the constraints of Q imply constraints on the values of $\mathbf{x}$ in the projection of Q. A straightforward application of Farkas's Lemma (see e.g. Chvátal (1983)) implies that such nonnegative linear combinations are the *only way* in which the constraints of Q imply constraints on the projection of Q.[12] Put differently, whenever $\mathbf{a}^\mathbf{T}\mathbf{x} \leq b$ is a supporting hyperplane of P, there exists a nonnegative linear combination λ that proves it (i.e., that satisfies (P1)–(P3)). This clarifies what the extended formulation Q really accomplishes:

[11] The converse also holds, and might well be the easier direction to anticipate. See the Exercises for details.

[12] Farkas's Lemma is sometimes phrased as the Separating Hyperplane Theorem. It can also be thought of as the feasibility version of strong linear programming duality.

ranging over all $\lambda \in \mathbb{R}^r_+$ satisfying (P2) generates all of the supporting hyperplanes $\mathbf{a}, b$ of P (with $\mathbf{a}$ and b arising as $\lambda^T\mathbf{C}$ and $\lambda^T\mathbf{d}$, respectively).

To define the promised λ_f's and μ_v's, fix a face f of P with supporting hyperplane $\mathbf{a^T x} \leq b$. Since Q's projection does not include any points not in P, the constraints of Q imply this supporting hyperplane. By the previous paragraph, we can choose a nonnegative vector λ_f so that (P1)–(P3) hold.

Now fix a vertex v of P. Since Q's projection includes every point of P, there exists a choice of $\mathbf{y}_v$ such that $(v, \mathbf{y}_v) \in Q$. Define $\mu_v \in \mathbb{R}^r_+$ as the slack in Q's constraints at the point $(v, \mathbf{y}_v)$:

$$\mu_v = \mathbf{d} - \mathbf{C}\mathbf{v} - \mathbf{D}\mathbf{y}_v.$$

Since $(v, \mathbf{y}_v) \in Q$, μ_v is a nonnegative vector.

Finally, for every face f of P and vertex v of P, we have

$$\lambda_f^T \mu_v = \underbrace{\lambda_f^T \mathbf{d}}_{=b} - \underbrace{\lambda_f^T \mathbf{C}\mathbf{v}}_{=\mathbf{a}^T v} - \underbrace{\lambda_f^T \mathbf{D}\mathbf{y}_v}_{=0} = b - \mathbf{a}^T\mathbf{v} = S_{fv},$$

as desired. This completes the second proof of Lemma 5.5.

5.4 A Lower Bound for the Correlation Polytope

5.4.1 Overview

Lemma 5.5 reduces the task of proving lower bounds on the size of extended formulations of a polytope P to proving lower bounds on the nondeterministic communication complexity of Face-Vertex(P). The case study of the permutahedron (Section 5.1.2) serves as a cautionary tale here: the communication complexity of Face-Vertex(P) is surprisingly low for some complex-seeming polytopes, so proving strong lower bounds, when they exist, typically requires work and a detailed understanding of the particular polytope of interest.

Fiorini et al. (2015) were the first to use Yannakakis's Lemma to prove lower bounds on the size of extended formulations of interesting polytopes.[13] We follow the proof plan of Fiorini et al. (2015), which has two steps.

1. First, we exhibit a polytope that is tailor-made for proving a nondeterministic communication complexity lower bound on the corresponding Face-Vertex(P) problem, via a reduction from Disjointness. We'll prove this step in full.

2. Second, we extend the consequent lower bound on the size of extended formulations to other problems, such as the Traveling Salesman Problem (TSP), via reductions. These reductions are bread-and-butter NP-completeness-style reductions; see the Exercises for more details.

[13] This paper won the Best Paper Award at STOC '12.

This two-step plan does not seem sufficient to resolve the motivating problem mentioned in Section 5.1, the non-bipartite matching problem. For an NP-hard problem like TSP, we fully expect all extended formulations of the convex hull of the characteristic vectors of solutions to be exponential; otherwise, we could use linear programming to obtain a subexponential-time algorithm for the problem (an unlikely result). The non-bipartite matching problem is polynomial-time solvable, so it's less clear what to expect. Rothvoß (2014) proved that every extended formulation of the convex hull of the perfect matchings of the complete graph has exponential size.[14] The techniques in Rothvoß (2014) are more sophisticated variations of the tools covered in this lecture — a reader of these notes is well-positioned to learn them.

5.4.2 Preliminaries

We describe a polytope P for which it's relatively easy to prove nondeterministic communication complexity lower bounds for the corresponding Face-Vertex(P) problem. The polytope was studied earlier for other reasons (Pitowsky, 1991; de Wolf, 2003).

Given a 0-1 n-bit vector $\mathbf{x}$, we consider the corresponding (symmetric and rank-1) outer product $\mathbf{x}\mathbf{x}^T$. For example, if $\mathbf{x} = 10101$, then

$$\mathbf{x}\mathbf{x}^T = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

For a positive integer n, we define COR as the convex hull of all 2^n such vectors $\mathbf{x}\mathbf{x}^T$ (ranging over $\mathbf{x} \in \{0,1\}^n$). This is a polytope in $\mathbb{R}^{n^2}$, and its vertices are precisely the points $\mathbf{x}\mathbf{x}^T$ with $\mathbf{x} \in \{0,1\}^n$.

Our goal is to prove the following result.

Theorem 5.6 (Fiorini et al. 2015) *The nondeterministic communication complexity of* Face-Vertex(COR) *is* $\Omega(n)$.

This lower bound is clearly the best possible (up to a constant factor), since Bob can communicate his vertex to Alice using only n bits (by specifying the appropriate $\mathbf{x} \in \{0,1\}^n$).

Lemma 5.5 then implies that every extended formulation of the COR polytope requires $2^{\Omega(n)}$ inequalities, no matter how many auxiliary variables are added. Note the dimension d is $\Theta(n^2)$, so this lower bound has the form $2^{\Omega(\sqrt{d})}$.

Elementary reductions (see the Exercises) translate this extension complexity lower bound for the COR polytope to a lower bound of $2^{\Omega(\sqrt{n})}$ on the size of extended formulations of the convex hull of characteristic vectors of n-point traveling salesman tours.

[14] This paper won the Best Paper Award at STOC '14.

5.4.3 Some Faces of the Correlation Polytope

Next we establish a key connection between certain faces of the correlation polytope and inputs to DISJOINTNESS. Throughout, n is a fixed positive integer.

Lemma 5.7 (Fiorini et al. 2015) *For every subset $S \subseteq \{1, 2, \ldots, n\}$, there is a face f_S of COR such that: for every $R \subseteq \{1, 2, \ldots, n\}$ with characteristic vector $\mathbf{x}_R$ and corresponding vertex $\mathbf{v}_R = \mathbf{x}_R\mathbf{x}_R^T$ of COR,*

$$\mathbf{v}_R \in f_S \quad \text{if and only if} \quad |S \cap R| = 1.$$

That is, among the faces of COR are 2^n faces that encode the "unique intersection property" for each of the 2^n subsets S of $\{1, 2, \ldots, n\}$. Note that for a given S, the sets R with $|S \cap R|$ can be generated by (i) first picking a element of S; (ii) picking a subset of $\{1, 2, \ldots, n\} \setminus S$. Thus if $|S| = k$, there are $k2^{n-k}$ sets R with which it has a unique intersection.

Lemma 5.7 is kind of amazing, but also not too hard to prove.

Proof of Lemma 5.7: For every $S \subseteq \{1, 2, \ldots, n\}$, we need to exhibit a supporting hyperplane $\mathbf{a}^\mathbf{T}\mathbf{x} \leq b$ such that $\mathbf{a}^\mathbf{T}\mathbf{v}_R = b$ if and only if $|S \cap R| = 1$, where $\mathbf{v}_R$ denotes $\mathbf{x}_R\mathbf{x}_R^T$ and $\mathbf{x}_R$ the characteristic vector of $R \subseteq \{1, 2, \ldots, n\}$.

Fix $S \subseteq \{1, 2, \ldots, n\}$. We develop the appropriate supporting hyperplane, in variables $\mathbf{y} \in \mathbb{R}^{n^2}$, over several small steps.

1. For clarity, let's start in the wrong space, with variables $\mathbf{z} \in \mathbb{R}^n$ rather than $\mathbf{y} \in \mathbb{R}^{n^2}$. Here $\mathbf{z}$ is meant to encode the characteristic vector of a set $R \subseteq \{1, 2, \ldots, n\}$. One sensible inequality to start with is

$$\sum_{i \in S} z_i - 1 \geq 0. \tag{5.9}$$

   For example, if $S = \{1, 3\}$, then this constraint reads $z_1 + z_3 - 1 \geq 0$.

   The good news is that for 0-1 vectors $\mathbf{x}_R$, this inequality is satisfied with equality if and only if $|S \cap R| = 1$. The bad news is that it does not correspond to a supporting hyperplane: if S and R are disjoint, then $\mathbf{x}_R$ violates the inequality. How can we change the constraint so that it holds with equality for $\mathbf{x}_R$ with $|S \cap R| = 1$ and also valid for all R?

2. One crazy idea is to square the left-hand side of (5.9):

$$\left(\sum_{i \in S} z_i - 1\right)^2 \geq 0. \tag{5.10}$$

   For example, if $S = \{1, 3\}$, then the constraint reads (after expanding) $z_1^2 + z_3^2 + 2z_1z_3 - 2z_1 - 2z_3 + 1 \geq 0$.

The good news is that every 0-1 vector $\mathbf{x}_R$ satisfies this inequality, and equality holds if and only if $|S \cap R| = 1$. The bad news is that the constraint is non-linear and hence does not correspond to a supporting hyperplane.

3. The obvious next idea is to "linearize" the previous constraint. Wherever the constraint has a $\mathbf{z}_i^2$ or a $\mathbf{z}_i$, we replace it by a variable $\mathbf{y}_{ii}$ (note these partially cancel out). Wherever the constraint has a $2z_iz_j$ (and notice for $i \neq j$ these always come in pairs), we replace it by a $y_{ij} + y_{ji}$. Formally, the constraint now reads

$$-\sum_{i \in S} y_{ii} + \sum_{i \neq j \in S} y_{ij} + 1 \geq 0. \tag{5.11}$$

Note that the new variable set is $\mathbf{y} \in \mathbb{R}^{n^2}$. For example, if $S = \{1, 3\}$, then the new constraint reads $y_{13} + y_{31} - y_{11} - y_{33} \geq -1$.

A first observation is that, for $\mathbf{y}$'s that are 0-1, symmetric, and rank-1, with $\mathbf{y} = \mathbf{z}\mathbf{z}^T$ (hence $y_{ij} = z_i \cdot z_j$ for $i, j \in \{1, 2, \ldots, n\}$), the left-hand sides of (5.10) and (5.11) are the same by definition. Thus, for $\mathbf{y} = \mathbf{x}_R\mathbf{x}_R^T$ with $\mathbf{x} \in \{0, 1\}^n$, $\mathbf{y}$ satisfies the (linear) inequality (5.11), and equality holds if and only if $|S \cap R| = 1$.

We have shown that, for every $S \subseteq \{1, 2, \ldots, n\}$, the linear inequality (5.11) is satisfied by every vector $\mathbf{y} \in \mathbb{R}^{n^2}$ of the form $\mathbf{y} = \mathbf{x}_R\mathbf{x}_R^T$ with $\mathbf{x} \in \{0, 1\}^n$. Since COR is by definition the convex hull of such vectors, every point of COR satisfies (5.11). This inequality is therefore a supporting hyperplane, and the face it induces contains precisely those vertices of the form $\mathbf{x}_R\mathbf{x}_R^T$ with $|S \cap R| = 1$. This completes the proof. ■

5.4.4 Face-Vertex(COR) and Unique-Disjointness

In the Face-Vertex(COR) problem, Alice receives a face f of COR and Bob a vertex $\mathbf{v}$ of COR. In the 1-inputs, $\mathbf{v} \notin f$; in the 0-inputs, $\mathbf{v} \in f$. Let's make the problem only easier by restricting Alice's possible inputs to the 2^n faces (one per subset $S \subseteq \{1, 2, \ldots, n\}$) identified in Lemma 5.7. In the corresponding matrix M_U of this function, we can index the rows by subsets S. Since every vertex of COR has the form $\mathbf{y} = \mathbf{x}_R\mathbf{x}_R^T$ for $R \subseteq \{1, 2, \ldots, n\}$, we can index the columns of M_U by subsets R. By Lemma 5.7, the entry (S, R) of the matrix M_U is 1 if $|S \cap R| \neq 1$ and 0 of $|S \cap R| = 1$. That is, the 0-entries of M_U correspond to pairs (S, R) that intersect in a unique element.

There is clearly a strong connection between the matrix M_U above and the analogous matrix M_D for Disjointness. They differ on entries (S, R) with $|S \cap R| \geq 2$: these are 0-entries of M_D but 1-entries of M_U. In other words, M_U is the matrix corresponding to the communication problem ¬Unique-Intersection: do the inputs S and R fail to have a unique intersection?

The closely related Unique-Disjointness problem is a "promise" version of Disjointness. The task here is to distinguish between:

(1) inputs (S, R) of DISJOINTNESS with $|S \cap R| = 0$;

(0) inputs (S, R) of DISJOINTNESS with $|S \cap R| = 1$.

For inputs that fall into neither case (with $|S \cap R| > 1$), the protocol is off the hook — any output is considered correct. Since a protocol that solves UNIQUE-DISJOINTNESS has to do only less than one that solves ¬UNIQUE-INTERSECTION, communication complexity lower bounds for former problem apply immediate to the latter.

We summarize the discussion of this section in the following proposition.

Proposition 5.8 (Fiorini et al. 2015) *The nondeterministic communication complexity of* FACE-VERTEX(COR) *is at least that of* UNIQUE-DISJOINTNESS.

5.4.5 A Lower Bound for UNIQUE-DISJOINTNESS

The Goal

One final step remains in our proof of Theorem 5.6, and hence of our lower bound on the size of extended formulations of the correlation polytope.

Theorem 5.9 (Fiorini et al. 2015; Kaibel and Weltge 2015) *The nondeterministic communication complexity of* UNIQUE-DISJOINTNESS *is* $\Omega(n)$.

DISJOINTNESS Revisited

As a warm-up, we revisit the standard DISJOINTNESS problem. Recall that, in Lecture 4, we proved that the nondeterministic communication complexity of DISJOINTNESS is at least n by a fooling set argument (Corollary 4.8). Next we prove a slightly weaker lower bound, via an argument that generalizes to UNIQUE-DISJOINTNESS.

The first claim is that, of the $2^n \times 2^n = 4^n$ possible inputs of DISJOINTNESS, exactly 3^n of them are 1-inputs. The reason is that the following procedure, which makes n 3-way choices, generates every 1-input exactly once: independently for each coordinate $i = 1, 2, \ldots, n$, choose between the options (i) $x_i = y_i = 0$; (ii) $x_i = 1$ and $y_i = 0$; and (iii) $x_i = 0$ and $y_i = 1$.

The second claim is that every 1-rectangle — every subset A of rows of M_D and B of columns of M_D such that $A \times B$ contains only 1-inputs — has size at most 2^n. To prove this, let $R = A \times B$ be a 1-rectangle. We assert that, for every coordinate $i = 1, 2, \ldots, n$, either (i) $x_i = 0$ for all $\mathbf{x} \in A$ or (ii) $y_i = 0$ for all $\mathbf{y} \in B$. That is, every coordinate has, for at least one of the two parties, a "forced zero" in R. For if neither (i) nor (ii) hold for a coordinate i, then since R is a rectangle (and hence closed under "mix and match") we can choose $(\mathbf{x}, \mathbf{y}) \in R$ with $x_i = y_i = 1$; but this is a 0-input and R is a 1-rectangle. This assertion implies that the following procedure, which makes n 2-way choices, generates every 1-input of R (and possibly other inputs as well): independently for each coordinate

$i = 1, 2, \ldots, n$, set the forced zero ($x_i = 0$ in case (i) or $y_i = 0$ in case (ii)) and choose a bit for this coordinate in the other input.

These two claims imply that every covering of the 1-inputs by 1-rectangles requires at least $(3/2)^n$ rectangles. Proposition 5.3 then implies a lower bound of $\Omega(n)$ on the nondeterministic communication complexity of Disjointness.

Proof of Theorem 5.9

Recall that the 1-inputs $(\mathbf{x}, \mathbf{y})$ of Unique-Disjointness are the same as those of Disjointness (for each i, either $x_i = 0$, $y_i = 0$, or both). Thus, there are still exactly 3^n 1-inputs. The 0-inputs $(\mathbf{x}, \mathbf{y})$ of Unique-Disjointness are those with $x_i = y_i = 1$ in exactly one coordinate i. We call all other inputs, where the promise fails to hold, **-inputs*. By a 1-rectangle, we now mean a rectangle with no 0-inputs (*-inputs are fine). With this revised definition, it is again true that every nondeterministic communication protocol that solves Unique-Disjointness using c bits of communication induces a covering of the 1-inputs by at most 2^c 1-rectangles.

Lemma 5.10 *Every 1-rectangle of* Unique-Disjointness *contains at most* 2^n *1-inputs.*

As with the argument for Disjointness, Lemma 5.10 completes the proof of Theorem 5.9: since there are 3^n 1-inputs and at most 2^n per 1-rectangle, every covering by 1-rectangles requires at least $(3/2)^n$ rectangles. This implies that the nondeterministic communication complexity of Unique-Disjointness is $\Omega(n)$.

Why is the proof of Lemma 5.10 harder than in Section 5.4.5? We can no longer easily argue that, in a rectangle $R = A \times B$, for each coordinate i, either $x_i = 0$ for all $\mathbf{x} \in A$ or $y_i = 0$ for all $\mathbf{y} \in B$. Assuming the opposite no longer yields a contraction: exhibiting $\mathbf{x} \in A$ and $\mathbf{y} \in B$ with $x_i = y_i = 1$ does not necessarily contradict the fact that R is a 1-rectangle, since $(\mathbf{x}, \mathbf{y})$ might be a *-input.

Proof of Lemma 5.10: The proof is one of those slick inductions that you can't help but sit back and admire.

We claim, by induction on $k = 0, 1, 2, \ldots, n$, that if $R = A \times B$ is a 1-rectangle for which all $\mathbf{x} \in A$ and $\mathbf{y} \in B$ have 0s in their last $n - k$ coordinates, then the number of 1-inputs in R is at most 2^k. The lemma is equivalent to the case of $k = n$. The base case $k = 0$ holds, because in this case the only possible input in R is $(\mathbf{0}, \mathbf{0})$.

For the inductive step, fix a 1-rectangle $R = A \times B$ in which the last $n - k$ coordinates of all $\mathbf{x} \in A$ and all $\mathbf{y} \in B$ are 0. To simplify notation, from here on we ignore the last $n - k$ coordinates of all inputs (they play no role in the argument).

Intuitively, we need to somehow "zero out" the kth coordinate of all inputs in R so that we can apply the inductive hypothesis. This motivates focusing on the kth coordinate, and we'll often write inputs $\mathbf{x} \in A$ and $\mathbf{y} \in B$ as $\mathbf{x}'a$ and $\mathbf{y}'b$, respectively, with $\mathbf{x}', \mathbf{y}' \in \{0, 1\}^{k-1}$

and $a, b \in \{0, 1\}$. (Recall we're ignoring that last $n - k$ coordinates, which are now always zero.)

First observe that, whenever $(\mathbf{x}'a, \mathbf{y}'b)$ is a 1-input, we cannot have $a = b = 1$. Also:

(*) If $(\mathbf{x}'a, \mathbf{y}'b) \in R$ is a 1-input, then R cannot contain both the inputs $(\mathbf{x}'0, \mathbf{y}'1)$ and $(\mathbf{x}'1, \mathbf{y}'0)$.

For otherwise, R would also contain the 0-input $(\mathbf{x}'1, \mathbf{y}'1)$, contradicting that R is a 1-rectangle. (Since $(\mathbf{x}'a, \mathbf{y}'b)$ is a 1-input, the unique coordinate of $(\mathbf{x}'1, \mathbf{y}'1)$ with a 1 in both inputs is the kth coordinate.)

The plan for the rest of the proof is to define two sets S_1, S_2 of 1-inputs — not necessarily rectangles — such that:

(P1) the number of 1-inputs in S_1 and S_2 combined is at least that in R;

(P2) the inductive hypothesis applies to $\text{rect}(S_1)$ and $\text{rect}(S_2)$, where $\text{rect}(S)$ denotes the smallest rectangle containing a set S of inputs.[15]

If we can find sets S_1, S_2 with properties (P1),(P2), then we are done: by the inductive hypothesis, the $\text{rect}(S_i)$'s have at most 2^{k-1} 1-inputs each, the S_i's are only smaller, and hence (by (P1)) R has at most 2^k 1-inputs, as required.

We define the sets in two steps, focusing first on property (P1). Recall that every 1-input $(\mathbf{x}, \mathbf{y}) \in R$ has the form $(\mathbf{x}'1, \mathbf{y}'0)$, $(\mathbf{x}'0, \mathbf{y}'1)$, or $(\mathbf{x}'0, \mathbf{y}'0)$. We put all 1-inputs of the first type into a set S_1', and all 1-inputs of the second type into a set S_2'. When placing inputs of the third type, we want to avoid putting two inputs of the form $(\mathbf{x}'a, \mathbf{y}'b)$ with the same $\mathbf{x}'$ and $\mathbf{y}'$ into the same set (this would create problems in the inductive step). So, for an input $(\mathbf{x}'0, \mathbf{y}'0) \in R$, we put it in S_1' if and only if the input $(\mathbf{x}'1, \mathbf{y}'0)$ was not already put in S_1'; and we put it in S_2' if and only if the input $(\mathbf{x}'0, \mathbf{y}'1)$ was not already put in S_2'. Crucially, observation (*) implies that R cannot contain two 1-inputs of the form $(\mathbf{x}'1, \mathbf{y}'0)$ and $(\mathbf{x}'0, \mathbf{y}'1)$, so the 1-input $(\mathbf{x}'0, \mathbf{y}'0)$ is placed in at least one of the sets S_1', S_2'. (It is placed in both if R contains neither $(\mathbf{x}'1, \mathbf{y}'0)$ nor $(\mathbf{x}'0, \mathbf{y}'1)$.) By construction, the sets S_1' and S_2' satisfy property (P1).

We next make several observations about S_1' and S_2'. By construction:

(**) for each $i = 1, 2$ and $\mathbf{x}', \mathbf{y}' \in \{0, 1\}^{k-1}$, there is at most one input of S_i' of the form $(\mathbf{x}'a, \mathbf{y}'b)$.

Also, since S_1', S_2' are subsets of the rectangle R, $\text{rect}(S_1'), \text{rect}(S_2')$ are also subsets of R. Since R is a 1-rectangle, so are $\text{rect}(S_1'), \text{rect}(S_2')$. Also, since every input $(\mathbf{x}, \mathbf{y})$ of S_i' (and

[15] Equivalently, the closure of S under the "mix and match" operation on pairs of inputs. Formally, $\text{rect}(S) = X(S) \times Y(S)$, where $X(S) = \{\mathbf{x} : (\mathbf{x}, \mathbf{y}) \in S \text{ for some } \mathbf{y}\}$ and $Y(S) = \{\mathbf{y} : (\mathbf{x}, \mathbf{y}) \in S \text{ for some } \mathbf{x}\}$.

hence $\text{rect}(S_i')$) has $y_k = 0$ (for $i = 1$) or $x_k = 0$ (for $i = 2$), the kth coordinate contributes nothing to the intersection of any inputs of $\text{rect}(S_1')$ or $\text{rect}(S_2')$.

Now obtain S_i from S_i' (for $i = 1, 2$) by zeroing out the kth coordinate of all inputs. Since the S_i''s only contain 1-inputs, the S_i's only contain 1-inputs. Since property (**) implies that $|S_i| = |S_i'|$ for $i = 1, 2$, we conclude that property (P1) holds also for S_1, S_2.

Moving on to property (P2), since $\text{rect}(S_1'), \text{rect}(S_2')$ contain no 0-inputs and contain only inputs with no intersection in the kth coordinate, $\text{rect}(S_1), \text{rect}(S_2)$ contain no 0-inputs.[16] Finally, since all inputs of S_1, S_2 have zeroes in their final $n - k + 1$ coordinates, so do all inputs of $\text{rect}(S_1), \text{rect}(S_2)$. The inductive hypothesis applies to $\text{rect}(S_1)$ and $\text{rect}(S_2)$, so each of them has at most 2^{k-1} 1-inputs. This implies the inductive step and completes the proof. ■

[16]The concern is that zeroing out an input in the kth coordinate turns some *-input (with intersection size 2) into a 0-input (with intersection size 1); but since there were no intersections in the kth coordinate, anyways, this can't happen.

Lecture 6

Lower Bounds for Data Structures

6.1 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 Miltersen (1999) and Pătraşcu (2008) 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.

6.2 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, ℓ). Most commonly, the metric space is Euclidean space ($\mathbb{R}^d$ with the ℓ_2 norm). In these lectures, we'll focus on the Hamming cube, where $X = \{0,1\}^d$ and ℓ is Hamming distance. 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 ϵ 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.

6.3 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 6.4 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.

6.3.1 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 6.3.3 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 $\frac{n}{2} + \sqrt{n}$. In Theorem 2.5 of Section 2.7, we proved that this problem is hard for one-way communication protocols (via a clever reduction from Index); it is also hard for general communication protocols (Chakrabarti and Regev, 2012; Sherstov, 2012; Vidick, 2011). 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 ϵ-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 ϵ-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

$$\mathbf{Pr}_{\mathbf{r}_j}[\langle \mathbf{r}_j, \mathbf{x} \rangle \not\equiv \langle \mathbf{r}_j, \mathbf{y} \rangle \bmod 2] = \frac{1}{2}\left(\left(1 - \left(1 - \frac{1}{L}\right)^{\Delta}\right)\right) \tag{6.1}$$

for every $\mathbf{r}_j \in R$. In (6.1), the quantity inside the outer parentheses is exactly the probability that at least one of the Δ 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 (6.1) is an increasing function of Δ, as one would expect. Let t denote the probability in (6.1) when $\Delta = L$. We're interested in how much bigger this probability is when Δ 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 ϵ 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 6.1 *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 $\ell_H(\mathbf{x}, \mathbf{y}) \leq L$, then $\ell_H(h(\mathbf{x}), h(\mathbf{y})) \leq (t + \frac{1}{2}h(\epsilon)))s$.

(ii) If $\ell_H(\mathbf{x}, \mathbf{y}) \geq (1+\epsilon)L$*, 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 ϵ-Gap Hamming problem can be solved by a public-coin randomized protocol with two-sided error and communication cost $\Theta(\epsilon^{-2})$.

6.3.2 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 jth coordinate of $h_R(\mathbf{x})$ to $\langle \mathbf{r}_j, \mathbf{x} \rangle$ mod 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 $\ell_H(h_R(\mathbf{p}), \mathbf{b}) \leq (t + \frac{1}{2}h(\epsilon))s$, where t is defined as in the previous section (as the probability in (6.1) 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 6.1 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 $\ell_H(\mathbf{p}, \mathbf{q}) \leq L$) or is not in $h(\mathbf{q})$ (if $\ell_h(\mathbf{p}, \mathbf{q}) \geq (1+\epsilon)L$). 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. Indyk (2004)).

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

6.3.3 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 6.3.2, 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

[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 6.3.2 is not well defined, a standard data structure for set membership, such as a perfect hash table (Fredman et al., 1984), can be used.

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

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 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 6.3.2) 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 ϵ 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.

6.4 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 6.4.1, we introduce a model for proving lower bounds on time-space trade-offs in data structures — the *cell probe model*. In Section 6.4.2, we explain how to deduce lower bounds in the cell probe model from a certain type of communication complexity lower bound. Section 6.4.3 applies this machinery to the $(1 + \epsilon)$-approximate nearest neighbor problem, and proves a sense in which the data structure of Section 6.3 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.

6.4.1 The Cell Probe Model

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 (Yao, 1979) — in the paper "Should Tables Be Sorted?" (Yao, 1981).[4] The point

[4] Actually, what is now called the cell probe model is a bit stronger than the model proposed by Yao (1981).

of this model is to prove lower bounds for data structures that allow random access. To make the model as 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.

Some History

To explain the title of the paper by Yao (1981), 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 6.1(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.

(a) Sorted Array (b) Unsorted Array

Figure 6.1 Should tables be sorted? For $n = 3$ and $k = 2$, it is suboptimal to store the array elements in sorted order.

Suppose, on the other hand, our storage strategy is as shown in Figure 6.1(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]

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

[5]Yao (1981) 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 (Fredman et al., 1984) are ineligible. If the universe size is not too much larger than the array size, then there are better solutions (Fiat and Naor, 1993; Gabizon and Hassidim, 2010; Gabizon and Shaltiel, 2012), even when there is no auxiliary storage.

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 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 Yao (1981) mentioned above, the word size w is $\lceil \log_2 n \rceil$ — just enough to specify the name of an element. The goal in Yao (1981) 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 6.3.2 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.

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

[6] The model in Yao (1981) 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.

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 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 6.2 *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*

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

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 6.2 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]

6.4.2 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 6.2 above.

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.

[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.

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 (Theorem 2.4) — Bob must send almost his entire input to Alice 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 6.3 (Miltersen et al. 1998) *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 6.3 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 6.3 only for deterministic protocols. The lower bound for randomized protocols with two-sided error is very similar, just a little messier (see Miltersen et al. (1998)).

Conceptually, the proof of Theorem 6.3 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

[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.

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

this, with the height and width of the monochromatic rectangle parameterized by the communication used by Alice and Bob, respectively.

Lemma 6.4 (Richness Lemma (Miltersen et al., 1998)) *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]

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 6.4 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_η the inputs $\mathbf{x} \in A$ such Alice sends the bit η 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.

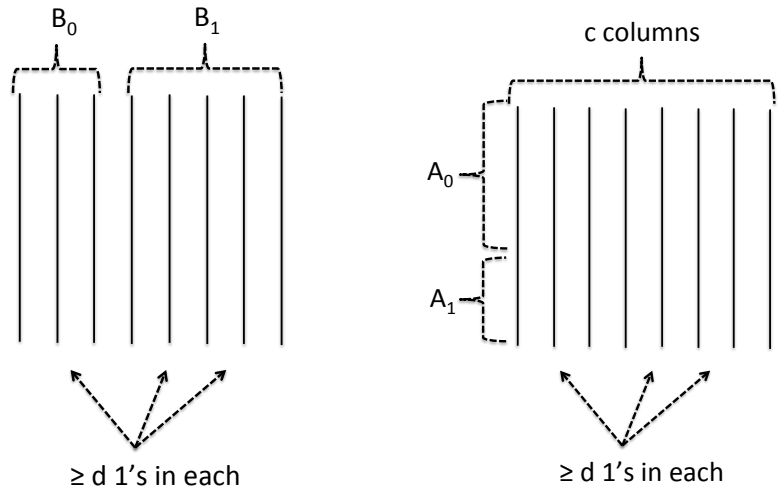
Proof of Lemma 6.4: We proceed by induction on the number of steps of the protocol. Suppose the protocol has generated the transcript-so-far $\mathbf{z}$ and that $A \times B$ is the rectangle of inputs consistent with this transcript. Suppose that at least c of the columns of B have at least d 1-inputs 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_η are the inputs $\mathbf{y} \in B$ such that (given the transcript-so-far $\mathbf{z}$) Bob sends the bit η. 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 6.2(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 6.2(b)).

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

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



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

Figure 6.2 Proof of the Richness Lemma (Lemma 6.4). 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.

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 α and β 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 6.4 and some simple calculations prove Theorem 6.3.

Proof of Theorem 6.3: We first observe that the matrix $M(\textsc{Index})$ 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 Lemma 6.4 proves that the matrix $M(\textsc{Index})$ 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 6.3 have any interesting implications? By Lemma 6.2, 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 6.3 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 (Fredman et al., 1984)). 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.

(k, ℓ)-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*

[12] Here $\delta \in (0, 1)$ is a constant and $n \geq N(\delta)$ is sufficiently large. Using the version of Theorem 6.3 with "$n^{1-2\delta}$" replaced by "$n^{1-c\delta}$" for an arbitrary constant $c > 1$, we can take δ arbitrarily close to 1.

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 6.3, 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 6.8 and Theorem 6.9 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 6.2 — 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 ϵ^{-2}. Formally, we define (k, ℓ)-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 ℓ (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 6.3 to show the following.

Theorem 6.5 (Andoni et al. 2006; Miltersen et al. 1998) *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:*

(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 6.3, we'll prove Theorem 6.5 for the special case of deterministic protocols. The theorem also holds for randomized protocols with two-sided error (Andoni et al., 2006), and we'll use this stronger result in Theorem 6.9 below. (Our upper bound in Section 6.3.2 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 Section 4.3.4).

Proof of Theorem 6.5: 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 6.4), there exists a

[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.

1-rectangle $A \times B$ where

$$|A| = \binom{n}{\epsilon^{-2}} \cdot 2^{-\delta\frac{\log n}{\epsilon^2}} \geq (\epsilon^2 n)^{\frac{1}{\epsilon^2}} \cdot n^{-\frac{\delta}{\epsilon^2}} = \epsilon^{\frac{2}{\epsilon^2}} n^{\frac{1}{\epsilon^2}(1-\delta)} \tag{6.2}$$

and

$$|B| = \underbrace{\binom{2n}{n}}_{\approx 2^{2n}/\sqrt{2n}} \cdot 2^{-\delta\frac{\log n}{\epsilon^2}} \cdot 2^{-n^{1-2\delta}} \geq 2^{2n-n^{1-3\delta/2}}, \tag{6.3}$$

where in (6.3) 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

$$|A| \leq \binom{s}{\epsilon^{-2}} \leq s^{1/\epsilon^2}. \tag{6.4}$$

Combining (6.2) and (6.4) implies that

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

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

$$|B| \leq 2^{2n-s} \leq 2^{2n-\epsilon^2 n^{1-\delta}}. \tag{6.5}$$

Inequalities (6.3) and (6.5) 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.

6.4.3 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 6.5, reduces to the Query-Database problem for the decision version of the $(1+\epsilon)$-nearest neighbor problem.

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 ith 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 6.4.1 — 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 6.3.2 — to the instances produced by the above reduction.

Precisely, we can use the following embedding lemma.

Lemma 6.6 (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 α;

(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 ϵ only.

Lemma 6.6 is an almost immediate consequence of Corollary 6.1 — 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 6.3.2 for a data structure — here we're using it for a lower bound!

Composing our initial reduction with Lemma 6.6 yields the following.

Corollary 6.7 *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 6.6 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 6.6. (Think of Lemma 6.6 as being invoked with a parameter ϵ' 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 6.4.2 translates our asymmetric communication complexity lower bound (via Lemma 6.2) to a data structure space lower bound.

Corollary 6.8 *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 $s = n^{\Omega(\epsilon^{-1})}$.*

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 6.7), Bob sends a sublinear number of bits. Theorem 6.5 then implies that Alice sends at least $\Omega(\epsilon^{-1} \log n)$ bits, and so (by Lemma 6.2) we have $t \log_2 s = \Omega(\epsilon^{-1} \log n)$. Since $t = O(1)$, this implies that $s = n^{\Omega(\epsilon^{-1})}$. ■

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

The culmination of this lecture is the following.

Theorem 6.9 *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 $s = n^{\Omega(\epsilon^{-2})}$.*

The proof is a refinement of the embedding arguments we used to prove Corollary 6.8. 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

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

[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 6.5 in its full generality.

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 ℓ_2 norm), which we then map (preserving distances) to high-dimensional space with the ℓ_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 6.6). 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 ϵ 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 ith standard basis vector.

First, suppose that S and T are disjoint. Then, the ℓ_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 ℓ_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 ℓ_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 6.5, following the proof of Corollary 6.8) 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 6.10 (Embedding Lemma #2) *For every $\delta > 0$ there exists a randomized function f from $\mathbb{R}^{2n}$ 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 6.10 says that one can re-represent a query $\mathbf{q}$ and a set P of n points in $\mathbb{R}^{2n}$ in a high-dimensional hypercube so that the nearest-neighbor distance — ℓ_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 $\mathbb{R}^D$ to $\mathbb{R}^{D'}$, where D' can depend on D and δ, such that, for every set $P \subseteq \mathbb{R}^D$ 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 $g : \mathbb{R}^{D'} \to \{0,1\}^{D''}$ such that, for every set $P \subseteq \mathbb{R}^{D'}$,

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

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

With Lemma 6.10 in hand, we can prove Theorem 6.9 by following the argument in Corollary 6.8.

Proof of Theorem 6.9: By Lemmas 6.6 and 6.10, 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 6.6 and 6.10).[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 6.5 then implies that Alice sends at least $\Omega(\epsilon^{-2} \log_2 n)$ bits, and so (by Lemma 6.2) we have $t \log_2 s = \Omega(\epsilon^{-2} \log_2 n)$. Since $t = O(1)$, this implies that $s = n^{\Omega(\epsilon^{-2})}$. ■

[15] Strictly speaking, we're using a generalization of Lemma 6.6 (with the same proof) where the query and point set can lie in a hypercube of arbitrarily large dimension, not just $2n$.

Lecture 7

Lower Bounds in Algorithmic Game Theory

7.1 Preamble

This lecture explains some applications of communication complexity to proving lower bounds in *algorithmic game theory (AGT)*, at the border of computer science and economics. In AGT, the natural description size of an object is often exponential in a parameter of interest, and the goal is to perform non-trivial computations in time polynomial in the parameter (i.e., logarithmic in the description size). As we know, communication complexity is a great tool for understanding when non-trivial computations require looking at most of the input.

7.2 The Welfare Maximization Problem

The focus of this lecture is the following optimization problem, which has been studied in AGT more than any other.

1. There are k players.

2. There is a set M of m items.

3. Each player i has a *valuation* $v_i : 2^M \to \mathbb{R}_+$. The number $v_i(T)$ indicates i's value, or willingness to pay, for the items $T \subseteq M$. The valuation is the private input of player i — i knows v_i but none of the other v_j's. We assume that $v_i(\emptyset) = 0$ and that the valuations are *monotone*, meaning $v_i(S) \leq v_i(T)$ whenever $S \subseteq T$. To avoid bit complexity issues, we'll also assume that all of the $v_i(T)$'s are integers with description length polynomial in k and m.

Note that we may have more than two players — more than just Alice and Bob. Also note that the description length of a player's valuation is exponential in the number of items m.

In the *welfare-maximization problem*, the goal is to partition the items M into sets $T_1, \ldots, T_k$ to maximize, at least approximately, the welfare

$$\sum_{i=1}^{k} v_i(T_i),$$

using communication polynomial in n and m. Note this amount of communication is logarithmic in the sizes of the private inputs.

The main motivation for this problem is combinatorial auctions. Already in the domain of government spectrum auctions, dozens of such auctions have raised hundreds of billions of dollars of revenue. They have also been used for other applications such as allocating take-off and landing slots at airports. For example, items could represent licenses for wireless spectrum — the right to use a certain frequency range in a certain geographic area. Players would then be wireless telecommunication companies. The value $v_i(S)$ would be the amount of profit company i expects to be able to extract from the licenses in S.

Designing good combinatorial auctions requires careful attention to "incentive issues," making the auctions as robust as possible to strategic behavior by the (self-interested) participants. Incentives won't play much of a role in this lecture. Our lower bounds for protocols in Section 7.4 apply even in the ideal case where players are fully cooperative. Our lower bounds for equilibria in Section 7.5 effectively apply no matter how incentive issues are resolved.

7.3 Multi-Party Communication Complexity

7.3.1 The Model

Welfare-maximization problems have an arbitrary number k of players, so lower bounds for them follow most naturally from lower bounds for *multi-party* communication protocols. The extension from two to many parties proceeds as one would expect, so we'll breeze through the relevant points without much fuss.

Suppose we want to compute a Boolean function $f : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \times \cdots \times \{0,1\}^{n_k} \to \{0,1\}$ that depends on the k inputs $\mathbf{x}_1, \ldots, \mathbf{x}_k$. We'll be interested in the *number-in-hand (NIH)* model, where player i only knows $\mathbf{x}_i$. What other model could there be, you ask? There's also the stronger *number-on-forehead (NOF)* model, where player i knows everything except $\mathbf{x}_i$. (Hence the name — imagine the players are sitting in a circle.) The NOF model is studied mostly for its connections to circuit complexity; it has few direct algorithmic applications, so we won't discuss it in this course. The NIH model is the natural one for our purposes and, happily, it's also much easier to prove strong lower bounds for it.

Deterministic protocols are defined as you would expect, with the protocol specifying whose turn it is speak (as a function of the protocol's transcript-so-far) and when the computation is complete. We'll use the *blackboard model*, where we think of the bits sent by each player as being written on a blackboard in public view.[1] Similarly, in a nondeterministic protocol, the prover writes a proof on the blackboard, and the protocol accepts the input if and only if all k players accept the proof.

[1] In the weaker *message-passing model*, players communicate by point-to-point messages rather than via broadcast.

7.3.2 The Multi-Disjointness Problem

We need a problem that is hard for multi-party communication protocols. An obvious idea is to use an analog of Disjointness. There is some ambiguity about how to define a version of Disjointness for three or more players. For example, suppose there are three players, and amongst the three possible pairings of them, two have disjoint sets while the third have intersecting sets. Should this count as a "yes" or "no" instance? We'll skirt this issue by worrying only about unambiguous inputs, that are either "totally disjoint" or "totally intersecting."

Formally, in the Multi-Disjointness problem, each of the k players i holds an input $\mathbf{x}_i \in \{0,1\}^n$. (Equivalently, a set $S_i \subseteq \{1,2,\ldots,n\}$.) The task is to correctly identify inputs that fall into one of the following two cases:

(1) "Totally disjoint," with $S_i \cap S_{i'} = \emptyset$ for every $i \neq i'$.

(0) "Totally intersecting," with $\cap_{i=1}^k S_i \neq \emptyset$.

When $k = 2$, this is just Disjointness. When $k > 2$, there are inputs that are neither 1-inputs nor 0-inputs. We let protocols off the hook on such ambiguous inputs — they can answer "1" or "0" with impunity.

In the next section, we'll prove the following communication complexity lower bound for Multi-Disjointness, credited to Jaikumar Radhakrishnan and Venkatesh Srinivasan in Nisan (2002).

Theorem 7.1 *The nondeterministic communication complexity of* Multi-Disjointness*, with k players with n-bit inputs, is $\Omega(n/k)$.*

The nondeterministic lower bound is for verifying a 1-input. (It is easy to verify a 0-input — the prover just suggests the index of an element r in $\cap_{i=1}^k S_i$, the validity of which is easily checked privately by each of the players.)

In our application in Section 7.4, we'll be interested in the case where k is much smaller than n, such as $k = \Theta(\log n)$. Intuition might suggest that the lower bound should be $\Omega(n)$ rather than $\Omega(n/k)$, but this is incorrect — a slightly non-trivial argument shows that Theorem 7.1 is tight for nondeterministic protocols (for all small enough k, like $k = O(\sqrt{n})$). See the Homework for details. This factor-k difference won't matter for our applications, however.

7.3.3 Proof of Theorem 7.1

The proof of Theorem 7.1 has three steps, all of which are generalizations of familiar arguments.

Step 1: *Every deterministic protocol with communication cost c induces a partition of $M(f)$ into at most 2^c monochromatic boxes.* By "$M(f)$," we mean the k-dimensional array in

which the ith dimension is indexed by the possible inputs of player i, and an array entry contains the value of the function f on the corresponding joint input. By a "box," we mean the k-dimensional generalization of a rectangle — a subset of inputs that can be written as a product $A_1 \times A_2 \times \cdots \times A_k$. By "monochromatic," we mean a box that does not contain both a 1-input and a 0-input. (Recall that for the Multi-Disjointness problem there are also wildcard ("*") inputs — a monochromatic box can contain any number of these.)

The proof of this step is the same as in the two-party case. We just run the protocol and keep track of the joint inputs that are consistent with the transcript. The box of all inputs is consistent with the empty transcript, and the box structure is preserved inductively: when player i speaks, it narrows down the remaining possibilities for the input $\mathbf{x}_i$, but has no effect on the possible values of the other inputs. Thus every transcript corresponds to a box, with these boxes partitioning $M(f)$. Since the protocol's output is constant over such a box and the protocol computes f, all of the boxes it induces are monochromatic with respect to $M(f)$.

Similarly, every nondeterministic protocol with communication cost c (for verifying 1-inputs) induces a cover of the 1-inputs of $M(f)$ by at most 2^c monochromatic boxes.

Step 2: *The number of* 1*-inputs in* $M(f)$ *is* $(k+1)^n$. This step and the next are easy generalizations of our second proof of our nondeterministic communication complexity lower bounds for Disjointness (from Section 5.4.5): first we lower bound the number of 1-inputs, then we upper bound the number of 1-inputs that can coexist in a single 1-box. In a 1-input $(\mathbf{x}_1, \ldots, \mathbf{x}_k)$, for every coordinate ℓ, at most one of the k inputs has a 1 in the ℓth coordinate. This yields $k+1$ options for each of the n coordinates, thereby generating a total of $(k+1)^n$ 1-inputs.

Step 3: *The number of* 1*-inputs in a monochromatic box is at most* k^n. Let $B = A_1 \times A_2 \times \cdots \times A_k$ be a 1-box. The key claim here is: for each coordinate $\ell = 1, \ldots, n$, there is a player $i \in \{1, \ldots, k\}$ such that, for every input $\mathbf{x}_i \in A_i$, the ℓth coordinate of $\mathbf{x}_i$ is 0. That is, to each coordinate we can associate an "ineligible player" that, in this box, never has a 1 in that coordinate. This is easily seen by contradiction: otherwise, there exists a coordinate ℓ such that, for every player i, there is an input $\mathbf{x}_i \in A_i$ with a 1 in the ℓth coordinate. As a box, this means that B contains the input $(\mathbf{x}_1, \ldots, \mathbf{x}_k)$. But this is a 0-input, contradicting the assumption that B is a 1-box.

The claim implies the stated upper bound. Every 1-input of B can be generated by choosing, for each coordinate ℓ, an assignment of at most one "1" in this coordinate to one of the $k-1$ eligible players for this coordinate. With only k choices per coordinate, there are at most k^n 1-inputs in the box B.

Conclusion: Steps 2 and 3 imply that covering of the 1s of the k-dimensional array of the Multi-Disjointness function requires at least $(1+\frac{1}{k})^n$ 1-boxes. By the discussion in Step 1, this implies a lower bound of $n \log_2(1+\frac{1}{k}) = \Theta(n/k)$ on the nondeterministic communication complexity of the Multi-Disjointness function (and output 1). This

concludes the proof of Theorem 7.1.

Remark 7.2 (Randomized Communication Complexity of Multi-Disjointness) Randomized protocols with two-sided error also require communication $\Omega(n/k)$ to solve Multi-Disjointness (Gronemeier, 2009; Chakrabarti et al., 2003).[2] This generalizes the $\Omega(n)$ lower bound that we stated (but did not prove) in Theorem 4.11, so naturally we're not going to prove this lower bound either. Extending the lower bound for Disjointness to Multi-Disjointness requires significant work, but it is a smaller step than proving from scratch a linear lower bound for Disjointness (Kalyanasundaram and Schnitger, 1992; Razborov, 1992). This is especially true if one settles for the weaker lower bound of $\Omega(n/k^4)$ (Alon et al., 1999), which is good enough for our purposes in this lecture.

7.4 Lower Bounds for Approximate Welfare Maximization

7.4.1 General Valuations

We now put Theorem 7.1 to work and prove that it is impossible to obtain a non-trivial approximation of the general welfare-maximization problem with a subexponential (in m) amount of communication. First, we observe that a k-approximation is trivial. The protocol is to give the full set of items M to the player with the largest $v_i(M)$. This protocol can clearly be implemented with a polynomial amount of communication. To prove the approximation guarantee, consider a partition $T_1, \ldots, T_k$ of M with the maximum-possible welfare W^*. There is a player i with $v_i(T_i) \geq W^*/k$. The welfare obtained by our simple protocol is at least $v_i(M)$; since we assume that valuations are monotone, this is at least $v_i(T_i) \geq W^*/k$.

To apply communication complexity, it is convenient to turn the optimization problem of welfare maximization into a decision problem. In the Welfare-Maximization(k) problem, the goal is to correctly identify inputs that fall into one of the following two cases:

(1) Every partition $(T_1, \ldots, T_k)$ of the items has welfare at most 1.

(0) There exists a partition $(T_1, \ldots, T_k)$ of the items with welfare at least k.

Clearly, communication lower bounds for Welfare-Maximization(k) apply more generally to the problem of obtaining a better-than-k-approximation of the maximum welfare.

We prove the following.

Theorem 7.3 (Nisan 2002) *The communication complexity of* Welfare-Maximization*(k) is* $\exp\{\Omega(m/k^2)\}$.

[2] There is also a far-from-obvious matching upper bound of $O(n/k)$ (Håstad and Wigderson, 2007; Chakrabarti et al., 2003).

Thus, if the number of items m is at least $k^{2+\epsilon}$ for some $\epsilon > 0$, then the communication complexity of the Welfare-Maximization(k) problem is exponential. Because the proof is a reduction from Multi-Disjointness, the lower bound applies to deterministic protocols, nondeterministic protocols (for the output 1), and randomized protocols with two-sided error.

The proof of Theorem 7.3 relies on Theorem 7.1 and a combinatorial gadget. We construct this gadget using the probabilistic method. As a thought experiment, consider t random partitions $P^1, \ldots, P^t$ of M, where t is a parameter to be defined later. By a random partition $P^j = (P_1^j, \ldots, P_k^j)$, we just mean that each of the m items is assigned to exactly one of the k players, independently and uniformly at random.

We are interested in the probability that two classes of different partitions intersect: for all $i \neq i'$ and $j \neq \ell$, since the probability that a given item is assigned to i in P^j and also to i' in P^ℓ is $\frac{1}{k^2}$, we have

$$\mathbf{Pr}\left[P_i^j \cap P_{i'}^\ell = \emptyset\right] = \left(1 - \frac{1}{k^2}\right)^m \leq e^{-m/k^2}.$$

Taking a Union Bound over the k choices for i and i' and the t choices for j and ℓ, we have

$$\mathbf{Pr}\left[\exists i \neq i', j \neq \ell \text{ s.t. } P_i^j \cap P_{i'}^\ell = \emptyset\right] \leq k^2 t^2 e^{-m/k^2}. \tag{7.1}$$

Call $P^1, \ldots, P^t$ an *intersecting family* if $P_i^j \cap P_{i'}^\ell \neq \emptyset$ whenever $i \neq i'$, $j \neq \ell$. By (7.1), the probability that our random experiment fails to produce an intersecting family is less than 1 provided $t < \frac{1}{k} e^{m/2k^2}$. The following lemma is immediate.

Lemma 7.4 *For every $m, k \geq 1$, there exists an intersecting family of partitions $P^1, \ldots, P^t$ with $t = \exp\{\Omega(m/k^2)\}$.*

A simple combination of Theorem 7.1 and Lemma 7.4 proves Theorem 7.3.

Proof of Theorem 7.3: The proof is a reduction from Multi-Disjointness. Fix k and m. (To be interesting, m should be significantly bigger than k^2.) Let $(S_1, \ldots, S_k)$ denote an input to Multi-Disjointness with t-bit inputs, where $t = \exp\{\Omega(m/k^2)\}$ is the same value as in Lemma 7.4. We can assume that the players have coordinated in advance on an intersecting family of t partitions of a set M of m items. Each player i uses this family and its input S_i to form the following valuation:

$$v_i(T) = \begin{cases} 1 & \text{if } T \supseteq P_i^j \text{ for some } j \in S_i \\ 0 & \text{otherwise.} \end{cases}$$

That is, player i is either happy (value 1) or unhappy (value 0), and is happy if and only if it receives all of the items in the corresponding class P_i^j of some partition P^j with index j belonging to its input to Multi-Disjointness. The valuations $v_1, \ldots, v_k$ define an input

to Welfare-Maximization(k). Forming this input requires no communication between the players.

Consider the case where the input to Multi-Disjointness is a 1-input, with $S_i \cap S_{i'} = \emptyset$ for every $i \neq i'$. We claim that the induced input to Welfare-Maximization(k) is a 1-input, with maximum welfare at most 1. To see this, consider a partition $(T_1, \ldots, T_k)$ in which some player i is happy (with $v_i(T_i) = 1$). For some $j \in S_i$, player i receives all the items in P_i^j. Since $j \notin S_{i'}$ for every $i' \neq i$, the only way to make a second player i' happy is to give it all the items in $P_{i'}^{\ell}$ in some other partition P^{ℓ} with $\ell \in S_{i'}$ (and hence $\ell \neq j$). Since $P^1, \ldots, P^t$ is an intersecting family, this is impossible — P_i^j and $P_{i'}^{\ell}$ overlap for every $\ell \neq j$.

When the input to Multi-Disjointness is a 0-input, with an element r in the mutual intersection $\cap_{i=1}^k S_i$, we claim that the induced input to Welfare-Maximization(k) is a 0-input, with maximum welfare at least k. This is easy to see: for $i = 1, 2, \ldots, k$, assign the items of P_i^r to player i. Since $r \in S_i$ for every i, this makes all k players happy.

This reduction shows that a (deterministic, nondeterministic, or randomized) protocol for Welfare-Maximization(k) yields one for Multi-Disjointness (with t-bit inputs) with the same communication. We conclude that the communication complexity of Welfare-Maximization(k) is $\Omega(t/k) = \exp\{\Omega(m/k^2)\}$. ■

7.4.2 Subadditive Valuations

To an algorithms person, Theorem 7.3 is depressing, as it rules out any non-trivial positive results. A natural idea is to seek positive results by imposing additional structure on players' valuations. Many such restrictions have been studied. We consider here the case of *subadditive* valuations, where each v_i satisfies $v_i(S \cup T) \leq v_i(S) + v_i(T)$ for every pair $S, T \subseteq M$.

Our reduction in Theorem 7.3 immediately yields a weaker inapproximability result for welfare maximization with subadditive valuations. Formally, define the Welfare-Maximization(2) problem as that of identifying inputs that fall into one of the following two cases:

(1) Every partition $(T_1, \ldots, T_k)$ of the items has welfare at most $k + 1$.

(0) There exists a partition $(T_1, \ldots, T_k)$ of the items with welfare at least $2k$.

Communication lower bounds for Welfare-Maximization(2) apply to the problem of obtaining a better-than-2-approximation of the maximum welfare.

Corollary 7.5 (Dobzinski et al. 2010) *The communication complexity of* Welfare-Maximization*(2) is* $\exp\{\Omega(m/k^2)\}$*, even when all players have subadditive valuations.*

Proof: Picking up where the reduction in the proof of Theorem 7.5 left off, every player i adds 1 to its valuation for every non-empty set of items. Thus, the previously 0-1 valuations become 0-1-2 valuations that are only 0 for the empty set. Such functions always satisfy the subadditivity condition ($v_i(S \cup T) \leq v_i(S) + v_i(T)$). 1-inputs and 0-inputs of Multi-Disjointness now become 1-inputs and 0-inputs of Welfare-Maximization(2), respectively. The communication complexity lower bound follows. ■

There is also a quite non-trivial matching upper bound of 2 for deterministic, polynomial-communication protocols (Feige, 2009).

7.5 Lower Bounds for Equilibria

The lower bounds of the previous section show that every protocol for the welfare-maximization problem that interacts with the players and then explicitly computes an allocation has either a bad approximation ratio or high communication cost. Over the past five years, many researchers have aimed to shift the work from the protocol to the players, by analyzing the equilibria of simple auctions. Can such equilibria bypass the communication complexity lower bounds proved in Section 7.4? The answer is not obvious, because equilibria are defined non-constructively, and not through a low-communication protocol.[3]

7.5.1 Game Theory

Next we give the world's briefest-ever game theory tutorial. See e.g. Shoham and Leyton-Brown (2010), or the instructor's CS364A lecture notes, for a more proper introduction. We'll be brief because the details of these concepts do not play a first-order role in the arguments below.

Games

A *(finite, normal-form) game* is specified by:

1. A finite set of $k \geq 2$ *players*.
2. For each player i, a finite *action set* A_i.
3. For each player i, a *utility function* $u_i(\mathbf{a})$ that maps an action profile $\mathbf{a} \in A_1 \times \cdots \times A_k$ to a real number. The utility of a player generally depends not only on its action, but also those chosen by the other players.

For example, in "Rock-Paper-Scissors (RPS)," there are two players, each with three actions. A natural choice of utility functions is depicted in Figure 7.1.

[3] This question was bothering your instructor back in CS364B (Winter '14) — hence, Theorem 7.9.

	Rock	Paper	Scissors
Rock	0,0	-1,1	1,-1
Paper	1,-1	0,0	-1,1
Scissors	-1,1	1,-1	0,0

Figure 7.1 Player utilities in Rock-Paper-Scissors. The pair of numbers in a matrix entry denote the utilities of the row and column players, respectively, in a given outcome.

For a more complex and relevant example of a game, consider *simultaneous first-price auctions (S1As)*. There are k players. An action a_i of a player i constitutes a bid b_{ij} on each item j of a set M of m items.[4] In a S1A, each item is sold separately in parallel using a "first-price auction" — the item is awarded to the highest bidder, and the price is whatever that player bid.[5] To specify the utility functions, we assume that each player i has a valuation v_i as in Section 7.2. We define

$$u_i(\mathbf{a}) = \underbrace{v_i(S_i)}_{\text{value of items won}} - \underbrace{\sum_{j \in S_i} b_{ij}}_{\text{price paid for them}},$$

where S_i denotes the items on which i is the highest bidder (given the bids of $\mathbf{a}$).[6] Note that the utility of a bidder depends both on its own action and those of the other bidders. Having specified the players, their actions, and their utility functions, we see that an S1A is an example of a game.

Equilibria

Given a game, how should one reason about it? The standard approach is to define some notion of "equilibrium" and then study the equilibrium outcomes. There are many useful notions of equilibria (see e.g. the instructor's CS364A notes); for simplicity, we'll stick here with the most common notion, (mixed) Nash equilibria.[7]

A *mixed strategy* for a player i is a probability distribution over its actions — for example, the uniform distribution over Rock/Paper/Scissors. A *Nash equilibrium* is a collection $\sigma_1, \ldots, \sigma_k$ of mixed strategies, one per player, so that each player is performing a "best response" to the others. To explain, adopt the perspective of player i. We think

[4] To keep the game finite, let's agree that each bid has to be an integer between 0 and some known upper bound B.

[5] You may have also heard of the *Vickrey* or *second-price* auction, where the winner does not pay their own bid, but rather the highest bid by someone else (the second-highest overall). We'll stick with S1As for simplicity, but similar results are known for simultaneous second-price auctions, as well.

[6] Break ties in an arbitrary but consistent way.

[7] For the auction settings we study, "Bayes-Nash equilibria" are more relevant. These generalize Nash equilibria, so our lower bounds immediately apply to them.

of i as knowing the mixed strategies σ_{-i} used by the other $k-1$ players (but not their coin flips). Thus, player i can compute the expected payoff of each action $a_i \in A_i$, where the expectation assumes that the other $k-1$ players randomly and independently select actions from their mixed strategies. Every action that maximizes i's expected utility is a *best response* to σ_{-i}. Similarly, every probability distribution over best responses is again a best response (and these exhaust the best responses). For example, in Rock-Paper-Scissors, both players playing the uniform distribution yields a Nash equilibrium. (Every action of a player has expected utility 0 w.r.t. the mixed strategy of the other player, so everything is a best response.)

Nash proved the following.

Theorem 7.6 (Nash 1950) *In every finite game, there is at least one Nash equilibrium.*

Theorem 7.6 can be derived from, and is essentially equivalent to, Brouwer's Fixed-Point Theorem. Note that a game can have a large number of Nash equilibria— if you're trying to meet a friend in New York City, with actions equal to intersections, then every intersection corresponds to a Nash equilibrium.

An *ϵ-Nash equilibrium* is the relaxation of a Nash equilibrium in which no player can increase its expected utility by more than ϵ by switching to a different strategy. Note that the set of ϵ-Nash equilibria is nondecreasing with ϵ. Such approximate Nash equilibria seem crucial to the lower bound in Theorem 7.9, below.

The Price of Anarchy

So how good are the equilibria of various games, such as S1As? To answer this question, we use an analog of the approximation ratio, adapted for equilibria. Given a game (like an S1A) and a nonnegative maximization objective function on the outcomes (like welfare), the *price of anarchy (POA)* (Koutsoupias and Papadimitriou, 1999) is defined as the ratio between the objective function value of an optimal solution, and that of the worst equilibrium.[8] If the equilibrium involves randomization, as with mixed strategies, then we consider its expected objective function value.

The POA of a game and a maximization objective function is always at least 1. It is common to identify "good performance" of a system with strategic participants as having a POA close to 1.[9]

For example, the equilibria of S1As are surprisingly good in fairly general settings.

[8] Recall that games generally have multiple equilibria. Ideally, we'd like an approximation guarantee that applies to *all* equilibria — this is the point of the POA.

[9] An important issue, outside the scope of these notes, is the plausibility of a system reaching an equilibrium. A natural solution is to relax the notion of equilibrium enough so that it become "relatively easy" to reach an equilibrium. See e.g. the instructor's CS364A notes for much more on this point.

Theorem 7.7 (Feldman et al. 2013) *In every S1A with subadditive bidder valuations, the POA is at most 2.*

Theorem 7.7 is non-trivial and we won't prove it here (see the paper or the instructor's CS364B notes for a proof). This result is particularly impressive because achieving an approximation factor of 2 for the welfare-maximization problem with subadditive bidder valuations by any means (other than brute-force search) is not easy (see Feige (2009)).

A recent result shows that the analysis of Feldman et al. (2013) is tight.

Theorem 7.8 (Christodoulou et al. 2013) *The worst-case POA of S1As with subadditive bidder valuations is at least 2.*

The proof of Theorem 7.8 is an ingenious explicit construction — the authors exhibit a choice of subadditive bidder valuations and a Nash equilibrium of the corresponding S1A so that the welfare of this equilibrium is only half of the maximum possible. One reason that proving results like Theorem 7.8 is challenging is that it can be difficult to solve for a (bad) equilibrium of a complex game like a S1A.

7.5.2 Price-of-Anarchy Lower Bounds from Communication Complexity

Theorem 7.7 motivates an obvious question: can we do better? Theorem 7.8 implies that the analysis in Feldman et al. (2013) cannot be improved, but can we reduce the POA by considering a different auction? Ideally, the auction would still be "reasonably simple" in some sense. Alternatively, perhaps no "simple" auction could be better than S1As? If this is the case, it's not clear how to prove it directly — proving lower bounds via explicit constructions auction-by-auction does not seem feasible.

Perhaps it's a clue that the POA upper bound of 2 for S1As (Theorem 7.7) gets stuck at the same threshold for which there is a lower bound for protocols that use polynomial communication (Theorem 7.5). It's not clear, however, that a lower bound for low-communication protocols has anything to do with equilibria. In the spirit of the other reductions that we've seen in this course, can we extract a low-communication protocol from an equilibrium?

Theorem 7.9 (Roughgarden 2014) *Fix a class $\mathcal{V}$ of possible bidder valuations. Suppose there exists no nondeterministic protocol with subexponential (in m) communication for the 1-inputs of the following promise version of the welfare-maximization problem with bidder valuations in $\mathcal{V}$:*

(1) Every allocation has welfare at most W^*/α.

(0) There exists an allocation with welfare at least W^*.

Let ϵ be bounded below by some inverse polynomial function of n and m. Then, for every auction with sub-doubly-exponential (in m) actions per player, the worst-case POA of ϵ-Nash equilibria with bidder valuations in $\mathcal{V}$ is at least α.

Theorem 7.9 says that lower bounds for nondeterministic protocols carry over to all "sufficiently simple" auctions, where "simplicity" is measured by the number of actions available to each player. These POA lower bounds follow from communication complexity lower bounds, and do not require any new explicit constructions.

To get a feel for the simplicity constraint, note that S1As with integral bids between 0 and B have $(B+1)^m$ actions per player — singly exponential in m. On the other hand, in a "direct-revelation" auction, where each bidder is allowed to submit a bid on each bundle $S \subseteq M$ of items, each player has a doubly-exponential (in m) number of actions.[10]

The POA lower bound promised by Theorem 7.9 is only for ϵ-Nash equilibrium; since the POA is a worst-case measure and the set of ϵ-Nash equilibria is nondecreasing with ϵ, this is weaker than a lower bound for exact Nash equilibria. It is an open question whether or not Theorem 7.9 holds also for the POA of exact Nash equilibria. Arguably, Theorem 7.9 is good enough for all practical purposes — a POA upper bound that holds for exact Nash equilibria and does not hold (at least approximately) for ϵ-Nash equilibria with very small ϵ is too brittle to be meaningful.

Theorem 7.9 has a number of interesting corollaries. First, since S1As have only a singly-exponential (in m) number of actions per player, Theorem 7.9 applies to them. Thus, combining it with Theorem 7.5 recovers the POA lower bound of Theorem 7.8 — modulo the exact vs. approximate Nash equilibria issue — and shows the optimality of the upper bound in Theorem 7.7 without an explicit construction. More interestingly, this POA lower bound of 2 (for subadditive bidder valuations) applies not only to S1As, but more generally to all auctions in which each player has a sub-doubly-exponential number of actions. Thus, S1As are in fact *optimal* among the class of all such auctions when bidders have subadditive valuations (w.r.t. the worst-case POA of ϵ-Nash equilibria).

We can also combine Theorem 7.9 with Theorem 7.3 to prove that no "simple" auction gives a non-trivial (better than k-) approximation for general bidder valuation. Thus with general valuations, complexity is essential to any auction format that offers good equilibrium guarantees.

7.5.3 Proof of Theorem 7.9

Presumably, the proof of Theorem 7.9 extracts a low-communication protocol from a good POA bound. The hypothesis of Theorem 7.9 offers the clue that we should be looking to construct a nondeterministic protocol. So what could we use an all-powerful prover for? We'll see that a good role for the prover is to suggest a Nash equilibrium to the players.

Unfortunately, it's too expensive for the prover to even write down the description of a Nash equilibrium, even in S1As. Recall that a mixed strategy is a distribution over actions, and that each player has an exponential (in m) number of actions available in a S1A. Specifying a Nash equilibrium thus requires an exponential number of probabilities.

[10] Equilibria can achieve the optimal welfare in direct-revelation mechanisms, so the bound in Theorem 7.9 on the number of actions is necessary. See the Exercises for further details.

To circumvent this issue, we resort to ϵ-Nash equilibria, which are guaranteed to exist even if we restrict ourselves to distributions with small descriptions.

Lemma 7.10 (Lipton et al. 2003) *For every $\epsilon > 0$ and every game with k players with action sets $A_1, \ldots, A_k$, there exists an ϵ-Nash equilibrium with description length polynomial in k, $\log(\max_{i=1}^{k} |A_i|)$, and $\frac{1}{\epsilon}$.*

We give the high-level idea of the proof of Lemma 7.10; see the Exercises for details.

1. Let $(\sigma_1, \ldots, \sigma_k)$ be a Nash equilibrium. (One exists, by Nash's Theorem.)

2. Run T independent trials of the following experiment: draw actions $a_1^t \sim \sigma_1, \ldots, a_k^t \sim \sigma_k$ for the k players independently, according to their mixed strategies in the Nash equilibrium.

3. For each i, define $\hat{\sigma}_i$ as the empirical distribution of the a_i^t's. (With the probability of a_i in $\hat{\sigma}_i$ equal to the fraction of trials in which i played a_i.)

4. Use Chernoff bounds to prove that, if T is at least a sufficiently large polynomial in k, $\log(\max_{i=1}^{k} |A_i|)$, and $\frac{1}{\epsilon}$, then with high probability $(\hat{\sigma}_1, \ldots, \hat{\sigma}_k)$ is an ϵ-Nash equilibrium. Note that the natural description length of $(\hat{\sigma}_1, \ldots, \hat{\sigma}_k)$ — for example, just by listing all of the sampled actions — is polynomial in n, $\log(\max_{i=1}^{k} |A_i|)$, and $\frac{1}{\epsilon}$.

The intuition is that, for T sufficiently large, expectations with respect to σ_i and with respect to $\hat{\sigma}_i$ should be roughly the same. Since there are $|A_i|$ relevant expectations per player (the expected utility of each of its actions) and Chernoff bounds give deviation probabilities that have an inverse exponential form, we might expect a $\log |A_i|$ dependence to show up in the number of trials.

We now proceed to the proof of Theorem 7.9.

Proof of Theorem 7.9: Fix an auction with at most A actions per player, and a value for $\epsilon = \Omega(1/\text{poly}(k, m))$. Assume that, no matter what the bidder valuations $v_1, \ldots, v_k \in \mathcal{V}$ are, the POA of ϵ-Nash equilibria of the auction is at most $\rho < \alpha$. We will show that A must be doubly-exponential in m.

Consider the following nondeterministic protocol for computing a 1-input of the welfare-maximization problem — for convincing the k players that every allocation has welfare at most W^*/α. See also Figure 7.2. The prover writes on a publicly visible blackboard an ϵ-Nash equilibrium $(\sigma_1, \ldots, \sigma_k)$ of the auction, with description length polynomial in k, $\log A$, and $\frac{1}{\epsilon} = O(\text{poly}(k, m))$ as guaranteed by Lemma 7.10. The prover also writes down the expected welfare contribution $\mathbf{E}[v_i(S)]$ of each bidder i in this equilibrium.

Given this advice, each player i verifies that σ_i is indeed an ϵ-best response to the other σ_j's and that its expected welfare is as claimed when all players play the mixed strategies $\sigma_1, \ldots, \sigma_k$. Crucially, player i is fully equipped to perform both of these checks without any

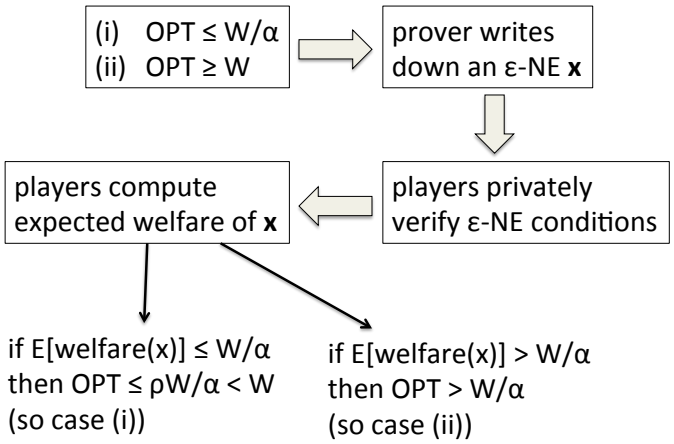


Figure 7.2 Proof of Theorem 7.9. How to extract a low-communication nondeterministic protocol from a good price-of-anarchy bound.

communication — it knows its valuation v_i (and hence its utility in each outcome of the game) and the mixed strategies used by all players, and this is all that is needed to verify the ϵ-Nash equilibrium conditions that apply to it and to compute its expected contribution to the welfare.[11] Player i accepts if and only if the prover's advice passes these two tests, and if the expected welfare of the equilibrium is at most W^*/α.

For the protocol correctness, consider first the case of a 1-input, where every allocation has welfare at most W^*/α. If the prover writes down the description of an arbitrary ϵ-Nash equilibrium and the appropriate expected contributions to the social welfare, then all of the players will accept (the expected welfare is obviously at most W^*/α). We also need to argue that, for the case of a 0-input — where some allocation has welfare at least W^* — there is no proof that causes all of the players to accept. We can assume that the prover writes down an ϵ-Nash equilibrium and its correct expected welfare W, since otherwise at least one player will reject. Since the maximum-possible welfare is at least W^* and (by assumption) the POA of ϵ-Nash equilibria is at most $\rho < \alpha$, the expected welfare of the given ϵ-Nash equilibrium must satisfy $W \geq W^*/\rho > W/\alpha$. Since the players will reject such a proof, we conclude that the protocol is correct. Our assumption then implies that the protocol has communication cost exponential in m. Since the cost of the protocol is polynomial in k, m, and $\log A$, A must be doubly exponential in m. ■

Conceptually, the proof of Theorem 7.9 argues that, when the POA of ϵ-Nash equilibria is small, every ϵ-Nash equilibrium provides a privately verifiable proof of a good upper bound

[11] These computations may take a super-polynomial amount of time, but they do not contribute to the protocol's cost.

on the maximum-possible welfare. When such upper bounds require large communication, the equilibrium description length (and hence the number of actions) must be large.

7.5.4 An Open Question

While Theorems 7.5, 7.7, and 7.9 pin down the best-possible POA achievable by simple auctions with subadditive bidder valuations, there are still open questions for other valuation classes. For example, a valuation v_i is *submodular* if it satisfies

$$v_i(T \cup \{j\}) - v_i(T) \leq v_i(S \cup \{j\}) - v_i(S)$$

for every $S \subseteq T \subset M$ and $j \notin T$. This is a "diminishing returns" condition for set functions. Every submodular function is also subadditive, so welfare-maximization with the former valuations is only easier than with the latter.

The worst-case POA of S1As is exactly $\frac{e}{e-1} \approx 1.58$ when bidders have submodular valuations. The upper bound was proved in Syrgkanis and Tardos (2013), the lower bound in Christodoulou et al. (2013). It is an open question whether or not there is a simple auction with a smaller worst-case POA. The best lower bound known — for nondeterministic protocols and hence, by Theorem 7.9, for the POA of ϵ-Nash equilibria of simple auctions — is $\frac{2e}{2e-1} \approx 1.23$. Intriguingly, there is an upper bound (slightly) better than $\frac{e}{e-1}$ for polynomial-communication protocols (Feige and Vondrák, 2010) — can this better upper bound also be realized as the POA of a simple auction? What is the best-possible approximation guarantee, either for polynomial-communication protocols or for the POA of simple auctions?

Lecture 8

Lower Bounds in Property Testing

8.1 Property Testing

We begin in this section with a brief introduction to the field of property testing. Section 8.2 explains the famous example of "linearity testing." Section 8.3 gives upper bounds for the canonical problem of "monotonicity testing," and Section 8.4 shows how to derive property testing lower bounds from communication complexity lower bounds.[1] These lower bounds will follow from our existing communication complexity toolbox (specifically, DISJOINTNESS); no new results are required.

Let D and R be a finite domain and range, respectively. In this lecture, D will always be $\{0,1\}^n$, while R might or might not be $\{0,1\}$. A *property* is simply a set $\mathcal{P}$ of functions from D to R. Examples we have in mind include:

1. Linearity, where $\mathcal{P}$ is the set of linear functions (with R a field and D a vector space over R).

2. Monotonicity, where $\mathcal{P}$ is the set of monotone functions (with D and R being partially ordered sets).

3. Various graph properties, like bipartiteness (with functions corresponding to characteristic vectors of edge sets, with respect to a fixed vertex set).

4. And so on. The property testing literature is vast. See Ron (2010) for a starting point.

In the standard property testing model, one has "black-box access" to a function $f : D \to R$. That is, one can only learn about f by supplying an argument $x \in D$ and receiving the function's output $f(x) \in R$. The goal is to test membership in $\mathcal{P}$ by querying f as few times as possible. Since the goal is to use a small number of queries (much smaller than $|D|$), there is no hope of testing membership exactly. For example, suppose you derive f from your favorite monotone function by changing its value at a single point to introduce a non-monotonicity. There is little hope of detecting this monotonicity violation with a small number of queries. We therefore consider a relaxed "promise" version of the membership problem.

[1] Somewhat amazingly, this connection was only discovered in 2011 (Blais et al., 2012), even though the connection is simple and property testing is a relatively mature field.

Formally, we say that a function f is ϵ-*far* from the property $\mathcal{P}$ if, for every $g \in \mathcal{P}$, f and g differ in at least $\epsilon|D|$ entries. Viewing functions as vectors indexed by D with coordinates in R, this definition says that f has distance at least $\epsilon|D|$ from its nearest neighbor in $\mathcal{P}$ (under the Hamming metric). Equivalently, repairing f so that it belongs to $\mathcal{P}$ would require changing at least an ϵ fraction of its values. A function f is ϵ-*close* to $\mathcal{P}$ if it is not ϵ-far — if it can be turned into a function in $\mathcal{P}$ by modifying its values on strictly less than $\epsilon|D|$ entries.

The property testing goal is to query a function f a small number of times and then decide if:

1. $f \in \mathcal{P}$; or

2. f is ϵ-far from $\mathcal{P}$.

If neither of these two conditions applies to f, then the tester is off the hook — any declaration is treated as correct.

A *tester* specifies a sequence of queries to the unknown function f, and a declaration of either "$\in \mathcal{P}$" or "ϵ-far from $\mathcal{P}$" at its conclusion. Interesting property testing results almost always require randomization. Thus, we allow the tester to be randomized, and allow it to err with probability at most 1/3. As with communication protocols, testers come in various flavors. *One-sided error* means that functions in $\mathcal{P}$ are accepted with probability 1, with no false negative allowed. Testers with *two-sided error* are allowed both false positives and false negatives (with probability at most 1/3, on every input that satisfies the promise). Testers can be *non-adaptive*, meaning that they flip all their coins and specify all their queries up front, or *adaptive*, with queries chosen as a function of the answers to previously asked queries. For upper bounds, we prefer the weakest model of non-adaptive testers with 1-sided error. Often (though not always) in property testing, neither adaptivity nor two-sided error leads to more efficient testers. Lower bounds can be much more difficult to prove for adaptive testers with two-sided error, however.

For a given choice of a class of testers, the *query complexity* of a property $\mathcal{P}$ is the minimum (over testers) worst-case (over inputs) number of queries used by a tester that solves the testing problem for $\mathcal{P}$. The best-case scenario is that the query complexity of a property is a function of ϵ only; sometimes it depends on the size of D or R as well.

8.2 Example: The BLR Linearity Test

The unofficial beginning of the field of property testing is Blum et al. (1993). (For the official beginning, see Rubinfeld and Sudan (1996) and Goldreich et al. (1998).) The setting is $D = \{0,1\}^n$ and $R = \{0,1\}$, and the property is the set of *linear functions*, meaning

functions f such that $f(\mathbf{x}+\mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y})$ (over $\mathbb{F}_2$) for all $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$.[2] The *BLR linearity test* is the following:

1. Repeat $t = \Theta(\frac{1}{\epsilon})$ times:

   a) Pick $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$ uniformly at random.

   b) If $f(\mathbf{x}+\mathbf{y}) \neq f(\mathbf{x}) + f(\mathbf{y})$ (over $\mathbb{F}_2$), then REJECT.

2. ACCEPT.

It is clear that if f is linear, then the BLR linearity test accepts it with probability 1. That is, the test has one-sided error. The test is also non-adaptive — the t random choices of $\mathbf{x}$ and $\mathbf{y}$ can all be made up front. The non-trivial statement is that only functions that are close to linear pass the test with large probability.

Theorem 8.1 (Blum et al. 1993) *If the BLR linearity test accepts a function f with probability greater than $\frac{1}{3}$, then f is ϵ-close to the set of linear functions.*

The modern and slick proof of Theorem 8.1 uses Fourier analysis — indeed, the elegance of this proof serves as convincing motivation for the more general study of Boolean functions from a Fourier-analytic perspective. See Chapter 1 of Donnell (2014) for a good exposition. There are also more direct proofs of Theorem 8.1, as in Blum et al. (1993). None of these proofs are overly long, but we'll spend our time on monotonicity testing instead. We mention the BLR test for the following reasons:

1. If you only remember one property testing result, Theorem 8.1 and the BLR linearity test would be a good one.

2. The BLR test is the thin end of the wedge in constructions of probabilistically checkable proofs (PCPs). Recall that a language is in NP if membership can be efficiently verified — for example, verifying an alleged satisfying assignment to a SAT formula is easy to do in polynomial time. The point of a PCP is to rewrite such a proof of membership so that it can be probabilistically verified after reading only a constant number of bits. The BLR test does exactly this for the special case of linearity testing — for proofs where "correctness" is equated with being the truth table of a linear function. The BLR test effectively means that one can assume without loss of generality that a proof encodes a linear function — the BLR test can be used as a preprocessing step to reject alleged proofs that are not close to a linear function. Subsequent testing steps can then focus on whether or not the encoded linear function is close to a subset of linear functions of interest.

[2] Equivalently, these are the functions that can be written as $f(\mathbf{x}) = \sum_{i=1}^n a_i x_i$ for some $a_1, \ldots, a_n \in \{0,1\}$.

3. Theorem 8.1 highlights a consistent theme in property testing — establishing connections between "global" and "local" properties of a function. Saying that a function f is ϵ-far from a property $\mathcal{P}$ refers to the entire domain D and in this sense asserts a "global violation" of the property. Property testers work well when there are ubiquitous "local violations" of the property. Theorem 8.1 proves that, for the property of linearity, a global violation necessarily implies lots of local violations. We give a full proof of such a "global to local" statement for monotonicity testing in the next section.

8.3 Monotonicity Testing: Upper Bounds

The problem of monotonicity testing was introduced in Goldreich et al. (2000) and is one of the central problems in the field. We discuss the Boolean case, where there have been several breakthroughs in just the past few months, in Sections 8.3.1 and 8.3.2. We discuss the case of larger ranges, where communication complexity has been used to prove strong lower bounds, in Section 8.3.3.

8.3.1 The Boolean Case

In this section, we take $D = \{0,1\}^n$ and $R = \{0,1\}$. For $b \in \{0,1\}$ and $\mathbf{x}_{-i} \in \{0,1\}^{n-1}$, we use the notation $(b, \mathbf{x}_{-i})$ to denote a vector of $\{0,1\}^n$ in which the ith bit is b and the other $n-1$ bits are $\mathbf{x}_{-i}$. A function $f : \{0,1\}^n \to \{0,1\}$ is *monotone* if flipping a coordinate of an input from 0 to 1 can only increase the function's output:

$$f(0, \mathbf{x}_{-i}) \leq f(1, \mathbf{x}_{-i})$$

for every $i \in \{1, 2, \ldots, n\}$ and $\mathbf{x}_{-i} \in \{0,1\}^{n-1}$.

It will be useful to visualize the domain $\{0,1\}^n$ as the n-dimensional hypercube; see also Figure 8.1. This graph has 2^n vertices and $n2^{n-1}$ edges. An edge can be uniquely specified by a coordinate i and vector $\mathbf{x}_{-i} \in \{0,1\}^{n-1}$ — the edge's endpoints are then $(0, \mathbf{x}_{-i})$ and $(1, \mathbf{x}_{-i})$. By the *ith slice* of the hypercube, we mean the 2^{n-1} edges for which the endpoints differ (only) in the ith coordinate. The n slices form a partition of the edge set of the hypercube, and each slice is a perfect matching of the hypercube's vertices. A function $\{0,1\}^n \to \{0,1\}$ can be visualized as a binary labeling of the vertices of the hypercube.

We consider the following *edge tester*, which picks random edges of the hypercube and rejects if it ever finds a monotonicity violation across one of the chosen edges.

1. Repeat t times:
   a) Pick $i \in \{1, 2, \ldots, n\}$ and $\mathbf{x}_{-i} \in \{0,1\}^{n-1}$ uniformly at random.
   b) If $f(0, \mathbf{x}_{-i}) > f(1, \mathbf{x}_{-i})$ then REJECT.
2. ACCEPT.

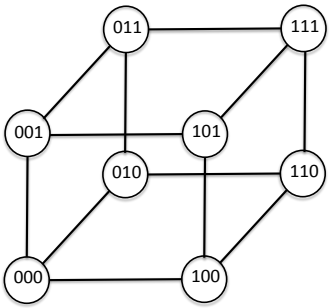


Figure 8.1 $\{0,1\}^n$ can be visualized as an n-dimensional hypercube.

Like the BLR test, it is clear that the edge tester has 1-sided error (no false negatives) and is non-adaptive. The non-trivial part is to understand the probability of rejecting a function that is ϵ-far from monotone — how many trials t are necessary and sufficient for a rejection probability of at least 2/3? Conceptually, how pervasive are the local failures of monotonicity for a function that is ϵ-far from monotone?

The bad news is that, in contrast to the BLR linearity test, taking t to be a constant (depending only on ϵ) is not good enough. The good news is that we can take t to be only logarithmic in the size of the domain.

Theorem 8.2 (Goldreich et al. 2000) *For $t = \Theta(\frac{n}{\epsilon})$, the edge tester rejects every function that is ϵ-far from monotone with probability at least 2/3.*

Proof: A simple calculation shows that it is enough to prove that a single random trial of the edge test rejects a function that is ϵ-far from monotone with probability at least $\frac{\epsilon}{n}$.

Fix an arbitrary function f. There are two quantities that we need to relate to each other — the rejection probability of f, and the distance between f and the set of monotone functions. We do this by relating both quantities to the sizes of the following sets: for $i = 1, 2, \ldots, n$, define

$$|A_i| = \{\mathbf{x}_{-i} \in \{0,1\}^{n-1} \,:\, f(0, \mathbf{x}_{-i}) > f(1, \mathbf{x}_{-i})\}. \tag{8.1}$$

That is, A_i is the edges of the ith slice of the hypercube across which f violates monotonicity. By the definition of the edge tester, the probability that a single trial rejects f is exactly

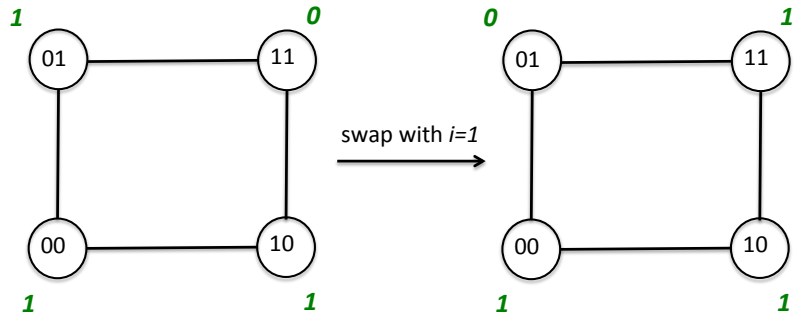
$$\underbrace{\sum_{i=1}^{n} |A_i|}_{\#\text{ of violations}} \quad / \quad \underbrace{n2^{n-1}}_{\#\text{ of edges}} . \tag{8.2}$$

Next, we upper bound the distance between f and the set of monotone functions, implying that the only way in which the $|A_i|$'s (and hence the rejection probability) can be

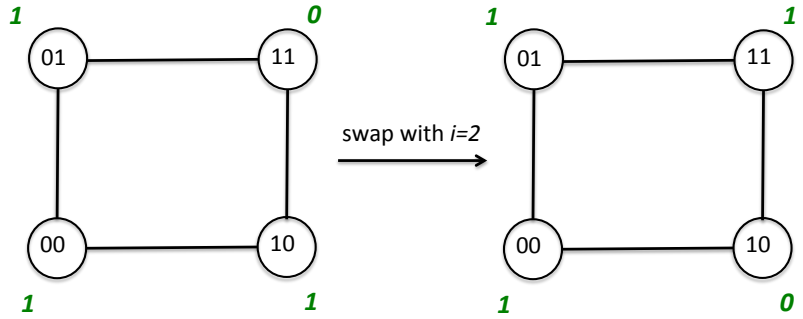
small is if f is close to a monotone function. To upper bound the distance, all we need to do is exhibit a single monotone function close to f. Our plan is to transform f into a monotone function, coordinate by coordinate, tracking the number of changes that we make along the way. The next claim controls what happens when we "monotonize" a single coordinate.

Key Claim: Let $i \in \{1, 2, \ldots, n\}$ be a coordinate. Obtain f' from f by, for each violated edge $((0, \mathbf{x}_{-i}), (1, \mathbf{x}_{-i})) \in A_i$ of the ith slice, swapping the values of f on its endpoints (Figure 8.2). That is, set $f'(0, \mathbf{x}_{-i}) = 0$ and $f'(1, \mathbf{x}_{-i}) = 1$. (This operation is well defined because the edges of A_i are disjoint.) For every coordinate $j = 1, 2, \ldots, n$, f' has no more monotonicity violations in the jth slice than does f.

Proof of Key Claim: The claim is clearly true for $j = i$: by construction, the swapping operation fixes all of the monotonicity violations in the ith slice, without introducing any new violations in the ith slice. The interesting case is when $j \neq i$, since new monotonicity violations can be introduced (cf., Figure 8.2). The claim asserts that the overall number of violations cannot increase (cf., Figure 8.2).



(a) Fixing the first slice



(b) Fixing the second slice

Figure 8.2 Swapping values to eliminate the monotonicity violations in the ith slice.

We partition the edges of the jth slice into edge pairs as follows. We use $\mathbf{x}^0_{-j}$ to denote an assignment to the $n-1$ coordinates other than j in which the ith coordinate is 0, and $\mathbf{x}^1_{-j}$ the corresponding assignment in which the ith coordinate is flipped to 1. For a choice of $\mathbf{x}^0_{-j}$, we can consider the "square" formed by the vertices $(0, \mathbf{x}^0_{-j})$, $(0, \mathbf{x}^1_{-j})$, $(1, \mathbf{x}^0_{-j})$, and $(1, \mathbf{x}^1_{-j})$; see Figure 8.3. The edges $((0, \mathbf{x}^0_{-j}), (1, \mathbf{x}^0_{-j}))$ and $((0, \mathbf{x}^1_{-j}), (1, \mathbf{x}^1_{-j}))$ belong to the jth slice, and ranging over the 2^{n-2} choices for $\mathbf{x}^0_{-j}$ — one binary choice per coordinate other than i and j — generates each such edge exactly once.

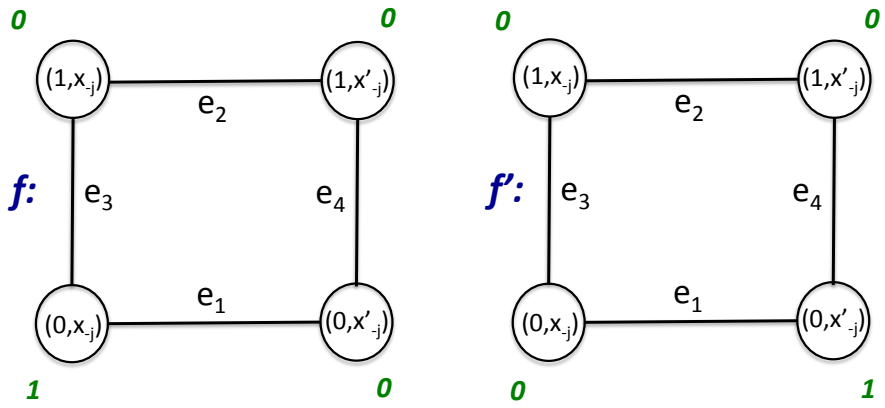


Figure 8.3 The number of monotonicity violations on edges e_3 and e_4 is at least as large under f as under f'.

Fix a choice of $\mathbf{x}^0_{-j}$, and label the edges of the corresponding square e_1, e_2, e_3, e_4 as in Figure 8.3. A simple case analysis shows that the number of monotonicity violations on edges e_3 and e_4 is at least as large under f as under f'. If neither e_1 nor e_2 was violated under f, then f' agrees with f on this square and the total number of monotonicity violations is obviously the same. If both e_1 and e_2 were violated under f, then values were swapped along both these edges; hence e_3 (respectively, e_4) is violated under f' if and only if e_4 (respectively, e_3) was violated under f. Next, suppose the endpoints of e_1 had their values swapped, while the endpoints of e_2 did not. This implies that $f(0, \mathbf{x}^0_{-j}) = 1$ and $f(0, \mathbf{x}^1_{-j}) = 0$, and hence $f'(0, \mathbf{x}^0_{-j}) = 0$ and $f(0, \mathbf{x}^1_{-j}) = 1$. If the endpoints $(1, \mathbf{x}^0_{-j})$ and $(1, \mathbf{x}^1_{-j})$ of e_2 have the values 0 and 1 (under both f and f'), then the number of monotonicity violations on e_3 and e_4 drops from 1 to 0. The same is true if their values are 0 and 0. If their values are 1 and 1, then the monotonicity violation on edge e_4 under f moves to one on edge e_3 under f', but the number of violations remains the same. The final set of cases, when the endpoints of e_2 have their values swapped while the endpoints of e_1 do not, is similar.[3]

[3]Suppose we corrected only one endpoint of an edge to fix a monotonicity violation, rather than swapping the endpoint values. Would the proof still go through?

Summing over all such squares — all choices of $\mathbf{x}^0_{-j}$ — we conclude that the number of monotonicity violations in the jth slice can only decrease. ■

Now consider turning a function f into a monotone function g by doing a single pass through the coordinates, fixing all monotonicity violations in a given coordinate via swaps as in the Key Claim. This process terminates with a monotone function: immediately after coordinate i is treated, there are no monotonicity violations in the ith slice by construction; and by the Key Claim, fixing future coordinates does not break this property. The Key Claim also implies that, in the iteration where this procedure processes the ith coordinate, the number of monotonicity violations that need fixing is at most the number $|A_i|$ of monotonicity violations in this slice under the original function f. Since the procedure makes two modifications to f for each monotonicity violation that it fixes (the two endpoints of an edge), we conclude that f can be made monotone by changing at most $2\sum_{i=1}^{n}|A_i|$ of its values. If f is ϵ-far from monotone, then $2\sum_{i=1}^{n}|A_i| \geq \epsilon 2^n$. Plugging this into (8.2), we find that a single trial of the edge tester rejects such an f with probability at least

$$\frac{\frac{1}{2}\epsilon 2^n}{n2^{n-1}} = \frac{\epsilon}{n},$$

as claimed. ■

8.3.2 Recent Progress for the Boolean Case

An obvious question is whether or not we can improve over the query upper bound in Theorem 8.2. The analysis in Theorem 8.2 of the edge tester is tight up to a constant factor (see Exercises), so an improvement would have to come from a different tester. There was no progress on this problem for 15 years, but recently there has been a series of breakthroughs on the problem. Chakrabarty and Seshadhri (2013) gave the first improved upper bounds, of $\tilde{O}(n^{7/8}/\epsilon^{3/2})$.[4] A year later, Chen et al. (2014) gave an upper bound of $\tilde{O}(n^{5/6}/\epsilon^4)$. Just a couple of months ago, Khot et al. (2015) gave a bound of $\tilde{O}(\sqrt{n}/\epsilon^2)$. All of these improved upper bounds are for *path testers*. The idea is to sample a random monotone path from the hypercube (checking for a violation on its endpoints), rather than a random edge. One way to do this is: pick a random point $\mathbf{x} \in \{0,1\}^n$; pick a random number z between 0 and the number of zeroes of $\mathbf{x}$ (from some distribution); and obtain $\mathbf{y}$ from $\mathbf{x}$ by choosing at random z of $\mathbf{x}$'s 0-coordinates and flipping them to 1. Given that a function that is ϵ-far from monotone must have lots of violated edges (by Theorem 8.2), it is plausible that path testers, which aspire to check many edges at once, could be more effective than edge testers. The issue is that just because a path contains one or more violated edges does not imply that the path's endpoints will reveal a monotonicity violation. Analyzing path testers seems substantially more complicated than the edge tester (Chakrabarty and Seshadhri, 2013;

[4]The notation $\tilde{O}(\cdot)$ suppresses logarithmic factors.

Chen et al., 2014; Khot et al., 2015). Note that path testers are non-adaptive and have 1-sided error.

There have also been recent breakthroughs on the lower bound side. It has been known for some time that all non-adaptive testers with 1-sided error require $\Omega(\sqrt{n})$ queries (Fischer et al., 2002); see also the Exercises. For non-adaptive testers with two-sided error, Chen et al. (2014) proved a lower bound of $\tilde{\Omega}(n^{1/5})$ and Chen et al. (2015) improve this to $\Omega(n^{(1/2)-c})$ for every constant $c > 0$. Because the gap in query complexity between adaptive and non-adaptive testers can only be exponential (see Exercises), these lower bounds also imply that adaptive testers (with two-sided error) require $\Omega(\log n)$ queries. The gap between $\tilde{O}(\sqrt{n})$ and $\Omega(\log n)$ for adaptive testers remains open; most researchers think that adaptivity cannot help and that the upper bound is the correct answer.

An interesting open question is whether or not communication complexity is useful for proving interesting lower bounds for the monotonicity testing of Boolean functions.[5] We'll see in Section 8.4 that it is useful for proving lower bounds in the case where the range is relatively large.

8.3.3 Larger Ranges

In this section we study monotonicity testing with the usual domain $D = \{0,1\}^n$ but with a range R that is an arbitrary finite, totally ordered set. Some of our analysis for the Boolean case continues to apply. For example, the edge tester continues to be a well-defined tester with 1-sided error. Returning to the proof of Theorem 8.2, we can again define each A_i as the set of monotonicity violations — meaning $f(0, \mathbf{x}_{-i}) > f(1, \mathbf{x}_{-i})$ — along edges in the ith slice. The rejection probability again equals the quantity in (8.2).

We need to revisit the major step of the proof of Theorem 8.2, which for Boolean functions gives an upper bound of $2\sum_{i=1}^{n} |A_i|$ on the distance from a function f to the set of monotone functions. One idea is to again do a single pass through the coordinates, swapping the function values of the endpoints of the edges in the current slice that have monotonicity violations. In contrast to the Boolean case, this idea does not always result in a monotone function (see Exercises).

We can extend the argument to general finite ranges R by doing multiple passes over the coordinates. The simplest approach uses one pass over the coordinates, fixing all monotonicity violations that involve a vertex $\mathbf{x}$ with $f(\mathbf{x}) = 0$; a second pass, fixing all monotonicity violations that involve a vertex $\mathbf{x}$ with $f(\mathbf{x}) = 1$; and so on. Formalizing this argument yields a bound of $2|R|\sum_{i=1}^{n} |A_i|$ on the distance between f and the set of monotone functions, which gives a query bound of $O(n|R|/\epsilon)$ Goldreich et al. (2000).

[5] At the very least, some of the techniques we've learned in previous lectures are useful. The arguments in Chen et al. (2014) and Chen et al. (2015) use an analog of Yao's Lemma (Lemma 2.3)) to switch from randomized to distributional lower bounds. The hard part is then to come up with a distribution over both monotone functions and functions ϵ-far from monotone such that no deterministic tester can reliably distinguish between the two cases using few queries to the function.

A divide-and-conquer approach gives a better upped bound Dodis et al. (1999). Assume without loss of generality (relabeling if necessary) that $R = \{0, 1, \ldots, r-1\}$, and also (by padding) that $r = 2^k$ for a positive integer k. The first pass over the coordinates fixes all monotonicity violations that involve values that differ in their most significant bit — one value that is less than $\frac{r}{2}$ and one value that is at least $\frac{r}{2}$. The second pass fixes all monotonicity violations involving two values that differ in their second-highest-order bit. And so on. The Exercises ask you to prove that this idea can be made precise and show that the distance between f and the set of monotone functions is at most $2 \log_2 |R| \sum_{i=1}^{n} |A_i|$. This implies an upper bound of $O(\frac{n}{\epsilon} \log |R|)$ on the number of queries used by the edge tester for the case of general finite ranges. The next section shows a lower bound of $\Omega(n/\epsilon)$ when $|R| = \Omega(\sqrt{n})$; in these cases, this upper bound is the best possible, up to the $\log R$ factor.[6]

8.4 Monotonicity Testing: Lower Bounds

8.4.1 Lower Bound for General Ranges

This section uses communication complexity to prove a lower bound on the query complexity of testing monotonicity for sufficiently large ranges.

Theorem 8.3 (Blais et al. (2012)) *For large enough ranges R and $\epsilon = \frac{1}{8}$, every (adaptive) monotonicity tester with two-sided error uses $\Omega(n)$ queries.*

Note that Theorem 8.3 separates the case of a general range R from the case of a Boolean range, where $\tilde{O}(\sqrt{n})$ queries are enough Khot et al. (2015). With the right communication complexity tools, Theorem 8.3 is not very hard to prove. Simultaneously with Blais et al. (2012), Briët et al. Briët et al. (2012) gave a non-trivial proof from scratch of a similar lower bound, but it applies only to non-adaptive testers with 1-sided error. Communication complexity techniques naturally lead to lower bounds for adaptive testers with two-sided error.

As always, the first thing to try is a reduction from Disjointness, with the query complexity somehow translating to the communication cost. At first this might seem weird — there's only one "player" in property testing, so where do Alice and Bob come from? But as we've seen over and over again, starting with our applications to streaming lower bounds, it can be useful to invent two parties just for the sake of standing on the shoulders of communication complexity lower bounds. To implement this, we need to show how a low-query tester for monotonicity leads to a low-communication protocol for Disjointness.

It's convenient to reduce from a "promise" version of Disjointness that is just as hard as the general case. In the Unique-Disjointness problem, the goal is to distinguish between

[6] It is an open question to reduce the dependence on $|R|$. Since we can assume that $|R| \leq 2^n$ (why?), any sub-quadratic upper bound $o(n^2)$ would constitute an improvement.

inputs where Alice and Bob have sets A and B with $A \cap B = \emptyset$, and inputs where $|A \cap B| = 1$. On inputs that satisfy neither property, any output is considered correct. The Unique-Disjointness problem showed up a couple of times in previous lectures; let's review them. At the conclusion of our lecture on the extension complexity of polytopes, we proved that the nondeterministic communication complexity of the problem is $\Omega(n)$ using a covering argument with a clever inductive proof (Theorem 5.9). In our boot camp (Section 4.3.4), we discussed the high-level approach of Razborov's proof that every randomized protocol for Disjointness with two-sided error requires $\Omega(n)$ communication. Since the hard probability distribution in this proof makes use only of inputs with intersection size 0 or 1, the lower bound applies also to the Unique-Disjointness problem.

Key to the proof of Theorem 8.3 is the following lemma.

Lemma 8.4 *Fix sets $A, B \subseteq U = \{1, 2, \ldots, n\}$. Define the function $h_{AB} : 2^U \to \mathbb{Z}$ by*

$$h_{AB}(S) = 2|S| + (-1)^{|S \cap A|} + (-1)^{|S \cap B|}. \tag{8.3}$$

Then:

(i) If $A \cap B = \emptyset$, then h is monotone.

(ii) If $|A \cap B| = 1$, then h is $\frac{1}{8}$-far from monotone.

We'll prove the lemma shortly; let's first see how to use it to prove Theorem 8.3. Let Q be a tester that distinguishes between monotone functions from $\{0,1\}^n$ to R and functions that are $\frac{1}{8}$-far from monotone. We proceed to construct a (public-coin randomized) protocol for the Unique-Disjointness problem.

Suppose Alice and Bob have sets $A, B \subseteq \{1, 2, \ldots, n\}$. The idea is for both parties to run local copies of the tester Q to test the function h_{AB}, communicating with each other as needed to carry out these simulations. In more detail, Alice and Bob first use the public coins to agree on a random string to be used with the tester Q. Given this shared random string, Q is deterministic. Alice and Bob then simulate local copies of Q query-by-query:

1. Until Q halts:

   a) Let $S \subseteq \{1, 2, \ldots, n\}$ be the next query that Q asks about the function h_{AB}.[7]

   b) Alice sends $(-1)^{|S \cap A|}$ to Bob.

   c) Bob sends $(-1)^{|S \cap B|}$ to Alice.

   d) Both Alice and Bob evaluate the function h_{AB} at S, and give the result to their respective local copies of Q.

[7] As usual, we're not distinguishing between subsets of $\{1, 2, \ldots, n\}$ and their characteristic vectors.

2. Alice (or Bob) declares "disjoint" if Q accepts the function h_{AB}, and "not disjoint" otherwise.

We first observe that the protocol is well defined. Since Alice and Bob use the same random string and simulate Q in lockstep, both parties know the (same) relevant query S to h_{AB} in every iteration, and thus are positioned to send the relevant bits ($(-1)^{|S\cap A|}$ and $(-1)^{|S\cap B|}$) to each other. Given these bits, they are able to evaluate h_{AB} at the point S (even though Alice doesn't know B and Bob doesn't know A).

The communication cost of this protocol is twice the number of queries used by the tester Q, and it doesn't matter if Q is adaptive or not. Correctness of the protocol follows immediately from Lemma 8.4, with the error of the protocol the same as that of the tester Q. Because every randomized protocol (with two-sided error) for Unique-Disjointness has communication complexity $\Omega(n)$, we conclude that every (possibly adaptive) tester Q with two-sided error requires $\Omega(n)$ queries for monotonicity testing. This completes the proof of Theorem 8.3.

Proof of Lemma 8.4: For part (i), assume that $A \cap B = \emptyset$ and consider any set $S \subseteq \{1, 2, \ldots, n\}$ and $i \notin S$. Because A and B are disjoint, i does not belong to at least one of A or B. Recalling (8.3), in the expression $h_{AB}(S \cup \{i\}) - h_{AB}(S)$, the difference between the first terms is 2, the difference in either the second terms (if $i \notin A$) or in the third terms (if $i \notin B$) is zero, and the difference in the remaining terms is at least -2. Thus, $h_{AB}(S \cup \{i\}) - h_{AB}(S) \geq 0$ for all S and $i \notin S$, and h_{AB} is monotone.

For part (ii), let $A \cap B = \{i\}$. For all $S \subseteq \{1, 2, \ldots, n\} \setminus \{i\}$ such that $|S \cap A|$ and $|S \cap B|$ are both even, $h_{AB}(S \cup \{i\}) - h_{AB}(S) = -2$. If we choose such an S uniformly at random, then $\mathbf{Pr}[|S \cap A| \text{ is even}]$ is 1 (if $A = \{i\}$) or $\frac{1}{2}$ (if A has additional elements, using the Principle of Deferred Decisions). Similarly, $\mathbf{Pr}[|S \cap B| \text{ is even}] \geq \frac{1}{2}$. Since no potential element of $S \subseteq \{1, 2, \ldots, n\} \setminus \{i\}$ is a member of both A and B, these two events are independent and hence $\mathbf{Pr}[|S \cap A|, |S \cap B| \text{ are both even}] \geq \frac{1}{4}$. Thus, for at least $\frac{1}{4} \cdot 2^{n-1} = 2^n/8$ choices of S, $h_{AB}(S \cup \{i\}) < h_{AB}(S)$. Since all of these monotonicity violations involve different values of h_{AB} — in the language of the proof of Theorem 8.2, they are all edges of the ith slice of the hypercube — fixing all of them requires changing h_{AB} at $2^n/8$ values. We conclude that h_{AB} is $\frac{1}{8}$-far from a monotone function. ■

8.4.2 Extension to Smaller Ranges

Recalling the definition (8.3) of the function h_{AB}, we see that the proof of Theorem 8.3 establishes a query complexity lower bound of $\Omega(n)$ provided the range R has size $\Omega(n)$. It is not difficult to extend the lower bound to ranges of size $\Omega(\sqrt{n})$. The trick is to consider a "truncated" version of h_{AB}, call it h'_{AB}, where values of h_{AB} less than $n - c\sqrt{n}$ are rounded up to $n - c\sqrt{n}$ and values more than $n + c\sqrt{n}$ are rounded down to $n + c\sqrt{n}$. (Here c is a sufficiently large constant.) The range of h'_{AB} has size $\Theta(\sqrt{n})$ for all $A, B \subseteq \{1, 2, \ldots, n\}$.

We claim that Lemma 8.4 still holds for h'_{AB}, with the "$\frac{1}{8}$" in case (ii) replaced by "$\frac{1}{16}$;" the new version of Theorem 8.3 then follows. Checking that case (i) in Lemma 8.4 still holds is easy: truncating a monotone function yields another monotone function. For case (ii), it is enough to show that h_{AB} and h'_{AB} differ in at most a $\frac{1}{16}$ fraction of their entries; since Hamming distance satisfies the triangle inequality, this implies that h'_{AB} must be $\frac{1}{16}$-far from the set of monotone functions. Finally, consider choosing $S \subseteq \{1, 2, \ldots, n\}$ uniformly at random: up to an ignorable additive term in $\{-2, -1, 0, 1, 2\}$, the value of h_{AB} lies in $n \pm c\sqrt{n}$ with probability at least $\frac{15}{16}$, provided c is a sufficiently large constant (by Chebyshev's inequality). This implies that h_{AB} and h'_{AB} agree on all but a $\frac{1}{16}$ fraction of the domain, completing the proof.

For even smaller ranges R, the argument above can be augmented by a padding argument to prove a query complexity lower bound of $\Omega(|R|^2)$; see the Exercises.

8.5 A General Approach

It should be clear from the proof of Theorem 8.3 that its method of deriving property testing lower bounds from communication complexity lower bounds is general, and not particular to the problem of testing monotonicity. The general template for deriving lower bounds for testing a property $\mathcal{P}$ is:

1. Map inputs $(\mathbf{x}, \mathbf{y})$ of a communication problem Π with communication complexity at least c to a function $h_{(\mathbf{x},\mathbf{y})}$ such that:

   a) 1-inputs $(\mathbf{x}, \mathbf{y})$ of Π map to functions $h_{(\mathbf{x},\mathbf{y})}$ that belong to $\mathcal{P}$;

   b) 0-inputs $(\mathbf{x}, \mathbf{y})$ of Π map to functions $h_{(\mathbf{x},\mathbf{y})}$ that are ϵ-far from $\mathcal{P}$.

2. Devise a communication protocol for evaluating $h_{(\mathbf{x},\mathbf{y})}$ that has cost d. (In the proof of Theorem 8.3, $d = 2$.)

Via the simulation argument in the proof of Theorem 8.3, instantiating this template yields a query complexity lower bound of c/d for testing the property $\mathcal{P}$.[8]

There are a number of other applications of this template to various property testing problems, such as testing if a function admits a small representation (as a sparse polynomial, as a small decision tree, etc.). See Blais et al. (2012); Goldreich (2013) for several examples.

A large chunk of the property testing literature is about testing graph properties Goldreich et al. (1998). An interesting open question is if communication complexity can be used to prove strong lower bounds for such problems.

[8] There is an analogous argument that uses one-way communication complexity lower bounds to derive query complexity lower bounds for *non-adaptive* testers; see the Exercises.

Bibliography

Alon, N., Matias, Y., and Szegedy, M. (1999). The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147.

Alon, N. and Spencer, J. H. (2008). *The Probabilistic Method*. Wiley. Third Edition.

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

Arora, S. and Lund, C. (1997). Hardness of approximations. In Hochbaum, D. S., editor, *Approximation Algorithms for NP-Hard Problems*, chapter 10, pages 399–446. PWS Publishing Company.

Babai, L., Frankl, P., and Simon, J. (1986). Complexity classes in communication complexity. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 337–347.

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

Bar-Yossef, Z., Jayram, T. S., Kumar, R., and Sivakumar, D. (2002a). An information statistics approach to data stream and communication complexity. In *Proceedings of the 43rd Symposium on Foundations of Computer Science (FOCS)*, pages 209–218.

Bar-Yossef, Z., Jayram, T. S., Kumar, R., Sivakumar, D., and Trevisan, L. (2002b). Counting distinct elements in a data stream. In *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques (RANDOM)*, pages 1–10.

Blais, E., Brody, J., and Matulef, K. (2012). Property testing lower bounds via communication complexity. *Computational Complexity*, 21(2):311–358.

Blum, M., Luby, M., and Rubinfeld, R. (1993). Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595.

Briët, J., Chakraborty, S., García-Soriano, D., and Matsliah, A. (2012). Monotonicity testing and shortest-path routing on the cube. *Combinatorica*, 32(1):35–53.

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

Chakrabarti, A., Khot, S., and Sun, X. (2003). Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *Proceedings of the 18th Annual IEEE Conference on Computational Complexity*, pages 107–117.

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

Chakrabarty, D. and Seshadhri, C. (2013). A $o(n)$ monotonicity tester for boolean functions over the hypercube. In *Proceedings of the 45th ACM Symposium on Theory of Computing (STOC)*, pages 411–418.

Charikar, M., Chen, K., and Farach-Colton, M. (2004). Finding frequent items in data streams. *Theoretical Computer Science*, 312(1):3–15.

Chattopadhyay, A. and Pitassi, T. (2010). The story of set disjointness. *SIGACT News*, 41(3):59–85.

Chen, X., De, A., Servedio, R. A., and Tan, L. (2015). Boolean function monotonicity testing requires (almost) $n^{1/2}$ non-adaptive queries. In *Proceedings of the 47th ACM Symposium on Theory of Computing (STOC)*, pages 519–528.

Chen, X., Servedio, R. A., and Tan, L. (2014). New algorithms and lower bounds for monotonicity testing. In *Proceedings of the 55th Symposium on Foundations of Computer Science (FOCS)*, pages 286–295.

Christodoulou, G., Kovacs, A., Sgoutitsa, A., and Tang, B. (2013). Tight bounds for the price of anarchy of simultaneous first price auctions. arXiv:1312.2371.

Chvátal, V. (1983). *Linear Programming*. Freeman.

Cormode, G. and Muthukrishnan, S. (2005). An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75.

de Wolf, R. (2003). Nondeterministic quantum query and quantum communication complexities. *SIAM Journal on Computing*, 32(3):681–699.

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

Dobzinski, S., Nisan, N., and Schapira, M. (2010). Approximation algorithms for combinatorial auctions with complement-free bidders. *Math. Oper. Res.*, 35(1):1–13.

Dodis, Y., Goldreich, O., Lehman, E., Raskhodnikova, S., Ron, D., and Samorodnitsky, A. (1999). Improved testing algorithms for monotonicity. In *Proceedings of APPROX-RANDOM*, pages 97–108.

Donnell, R. O. (2014). *Analysis of Boolean Functions*. Cambridge.

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

Edmonds, J. (1965). Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards, Series B*, 69B:125–130.

Feige, U. (2009). On maximizing welfare where the utility functions are subadditive. *SIAM Journal on Computing*, 39(1):122–142.

Feige, U. and Vondrák, J. (2010). The submodular welfare problem with demand queries. *Theory of Computing*, 6(1):247–290.

Feldman, M., Fu, H., Gravin, N., and Lucier, B. (2013). Simultaneous auctions are (almost) efficient. In *45th ACM Symposium on Theory of Computing (STOC)*, pages 201–210.

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

Fiorini, S., Massar, S., Pokutta, S., Tiwary, H. R., and de Wolf, R. (2015). Exponential lower bounds for polytopes in combinatorial optimization. *Journal of the ACM*, 62(2). Article 17.

Fischer, E., Lehman, E., Newman, I., Raskhodnikova, S., Rubinfeld, R., and Samorodnitsky, A. (2002). Monotonicity testing over general poset domains. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 474–483.

Flajolet, P. and Martin, G. N. (1983). Probabilistic counting. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 76–82.

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

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

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

Goemans, M. X. (2014). Smallest compact formulation for the permutahedron. *Mathematical Programming, Series A*. To appear.

Goldreich, O. (2013). On the communication complexity methodology for proving lower bounds on the query complexity of property testing. *Electronic Colloquium on Computational Complexity (ECCC)*, 20.

Goldreich, O., Goldwasser, S., Lehman, E., Ron, D., and Samorodnitsky, A. (2000). Testing monotonicity. *Combinatorica*, 20(3):301–337.

Goldreich, O., Goldwasser, S., and Ron, D. (1998). Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750.

Gronemeier, A. (2009). Asymptotically optimal lower bounds on the NIH-multi-party information complexity of the AND-function and disjointness. In *Proceedings of the 26th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 505–516.

Grötschel, M., Lovász, L., and Schrijver, A. (1988). *Geometric Algorithms and Combinatorial Optimization*. Springer. Second Edition, 1993.

Håstad, J. and Wigderson, A. (2007). The randomized communication complexity of set disjointness. *Theory of Computing*, 3(11):211–219.

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

Indyk, P. and Woodruff, D. P. (2005). Optimal approximations of the frequency moments of data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 202–208.

Jayram, T. S., Kumar, R., and Sivakumar, D. (2008). The one-way communication complexity of hamming distance. *Theory of Computing*, 4:129–135.

Jukna, S. (2012). *Boolean Function Complexity: Advances and Frontiers*. Springer.

Kaibel, V. and Weltge, S. (2015). A short proof that the extension complexity of the correlation polytope grows exponentially. *Discrete & Computational Geometry*, 53(2):397–401.

Kalyanasundaram, B. and Schnitger, G. (1992). The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5(4):545–557.

Khachiyan, L. G. (1979). A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20(1):191–194.

Khot, S., Minzer, D., and Safra, M. (2015). On monotonicity testing and Boolean isoperimetric type theorems. In *Proceedings of the 56th Symposium on Foundations of Computer Science (FOCS)*. To appear.

Koutsoupias, E. and Papadimitriou, C. H. (1999). Worst-case equilibria. In *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1563 of *Lecture Notes in Computer Science*, pages 404–413.

Kremer, I., Nisan, N., and D, R. (1999). On randomized one-round communication complexity. *Computational Complexity*, 8(1):21–49.

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97.

Kushilevitz, E. and Nisan, N. (1996). *Communication Complexity*. Cambridge.

Lee, T. and Shraibman, A. (2009). Lower bounds in communication complexity. *Foundations and Trends in Theoretical Computer Science*, 3(4):263–399.

Lipton, R. J., Markakis, E., and Mehta, A. (2003). Playing large games using simple strategies. In *Proceedings of the 4th ACM Conference on Electronic Commerce (EC)*, pages 36–41.

Lovász, L. (1990). Communication complexity: A survey. In Korte, B. H., editor, *Paths, Flows, and VLSI Layout*, pages 235–265. Springer-Verlag.

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

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

Moitra, A. (2014). Algorithmic aspects of machine learning. Lecture notes.

Nash, J. F. (1950). Equilibrium points in N-person games. *Proceedings of the National Academy of Science*, 36(1):48–49.

Newman, I. (1991). Private vs. common random bits in communication complexity. *Information Processing Letters*, 39:67–71.

Nisan, N. (2002). The communication complexity of approximate set packing. In *Proceedings of the 29th Annual International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 2380 of *Lecture Notes in Computer Science*, pages 868–875.

Padberg, M. and Rao, M. R. (1982). Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research*, 7:67–80.

Pitowsky, I. (1991). Correlation polytopes: Their geometry and complexity. *Mathematical Programming*, 50:395–414.

Pătraşcu, M. (2008). *Lower Bound Techniques for Data Structures*. PhD thesis, Massachusetts Institute of Technology.

Pătraşcu, M. (2009). Communication complexity I, II, III, IV. WebDiarios de Motocicleta blog posts dated May 23, 2008; March 25, 2009; April 3, 2009; April 8, 2009.

Pulleyblank, W. and Edmonds, J. (1974). Facets of 1-matching polyhedra. In *Proceedings of the Working Seminar on Hypergraphs*, pages 214–242. Springer.

Razborov, A. A. (1992). On the distributional complexity of disjointness. *Theoretical Computer Science*, 106(2):385–390.

Razborov, A. A. (2011). Communication complexity. In Schleicher, D. and Lackmann, M., editors, *An Invitation to Mathematics: From Competitions to Research*, pages 97–117. Springer.

Ron, D. (2010). Property testing: A learning theory perspective. *Foundations and Trends in Theoretical Computer Science*, 5(2):73–205–402.

Rothvoß, T. (2014). The matching polytope has exponential extension complexity. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 263–272.

Roughgarden, T. (2014). Barriers to near-optimal equilibria. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 71–80.

Rubinfeld, R. and Sudan, M. (1996). Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271.

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

Shoham, Y. and Leyton-Brown, K. (2010). *Multiagent Systems*. Cambridge.

Syrgkanis, V. and Tardos, É. (2013). Composable and efficient mechanisms. In *Proceedings of the 45th ACM Symposium on Theory of Computing (STOC)*, pages 211–220.

Vidick, T. (2011). 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.

Woodruff, D. P. (2004). Optimal space lower bounds for all frequency moments. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 167–175.

Woodruff, D. P. (2007). *Efficient and Private Distance Approximation in the Communication and Streaming Models*. PhD thesis, MIT.

Yannakakis, M. (1991). Expressing combinatorial optimization problems by linear programs. *Journal of Computer and System Sciences*, 43(3):441–466.

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

Yao, A. C.-C. (1979). 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.

Yao, A. C.-C. (1983). Lower bounds by probabilistic arguments. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 420–428.

Ziegler, G. M. (1995). *Lectures on Polytopes*. Springer.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de