CS369E: Communication Complexity (for Algorithm Designers) Lecture #2: Lower Bounds for One-Way Communication: Disjointness, Index, and Gap-Hamming*

Tim Roughgarden†

January 15, 2015

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 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.

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

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

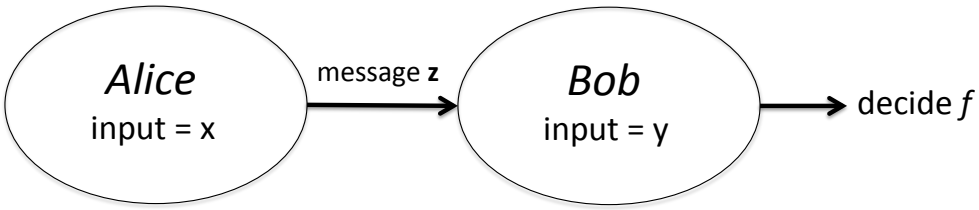

Figure 1: A one-way communication protocol. Alice sends a message to Bob that depends only on her input; Bob makes a decision based on his input and Alice's message.

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.[1]

Theorem 1.1 *The one-way communication complexity of the Fproblem 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 1.1 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.

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

In today's lecture we'll prove Theorem 1.1, 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 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 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

[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.

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 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

[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.

[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 oflog $\frac{1}{\delta}$.

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 1.1 as follows.

Theorem 2.1 *Every public-coin randomized 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?

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 3.1 (Yao [5]) *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*

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

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 3.1, 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 3.1 extremely useful, but it is easy to prove.

Proof of Lemma 3.1: 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 3.1 also holds — whatever the true randomized communication complexity of a problem, there exists a bad distribution D over inputs that proves it [5]. 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.

4 The Index Problem

We prove Theorem 2.1 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 4.1 ([2]) *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 4.1 easily implies Theorem 2.1.

Proof of Theorem 2.1: 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 4.1. While some computations are required, the proof is conceptually pretty straightforward.

Proof of Theorem 4.1: 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 3.1, 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, 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 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

$$\mathbf{Pr}_i[P \text{ is incorrect} \,|\, \mathbf{x}, \mathbf{z}] = \frac{d_H(\mathbf{x}, \mathbf{a}(\mathbf{z}))}{n}, \tag{1}$$

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 (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 $\mathbf{a} \in A$

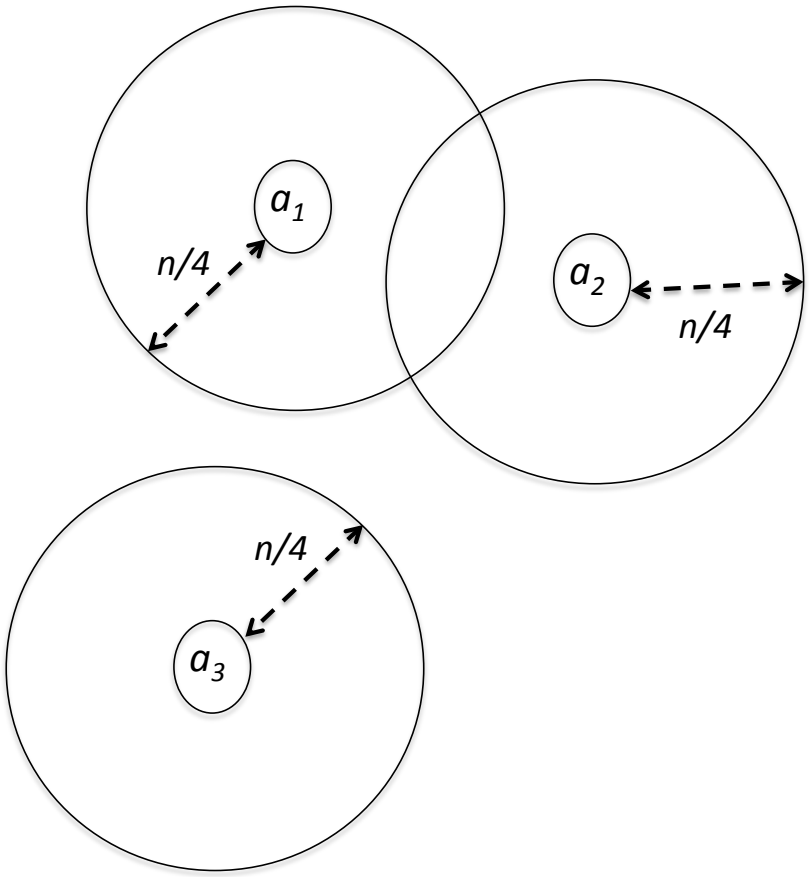

Figure 2: Balls of radius $n/4$ in the Hamming metric, centered at the answer vectors used by the protocol P.

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. 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]

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 (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

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

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) 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). ■

5 Where We're Going

Theorem 4.1 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 4.1), reduced INDEX to DISJOINTNESS to extend the lower bound to the latter problem (Theorem 2.1), and reduced DISJOINTNESS to various streaming computations (last lecture). See also Figure 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.

$$\underbrace{\text{INDEX}}_{\text{Theorem 4.1}} \xrightarrow{\text{Theorem 2.1}} \text{DISJOINTNESS} \xrightarrow{\text{Lecture \#1}} \text{STREAMING}$$

Figure 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 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.

$$\underbrace{\text{INDEX}}_{\text{Theorem 4.1}} \xrightarrow{\text{Theorem 7.1}} \text{GAP-HAMMING} \xrightarrow{\text{Section 6.2}} \text{STREAMING}$$

Figure 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.

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.

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 Fis 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 F? 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.

6.2 Reducing Gap-Hamming to F_0 Estimation

A $(1 \pm \frac{1}{\sqrt{n}})$-approximation of F_0 is insufficient to solve F— 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 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.

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 $d_H(\mathbf{x}, \mathbf{y}) < t - c\sqrt{n}$

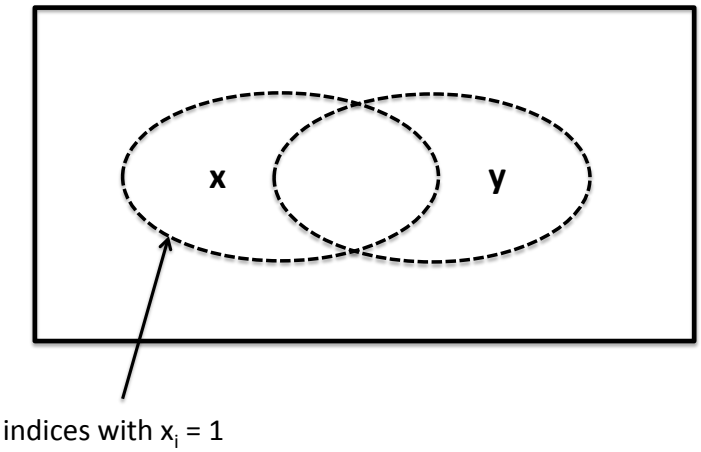

Figure 5: The Hamming distance between two bit vectors equals the size of the symmetric difference of the corresponding subsets of 1-coordinates.

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.

7 Lower Bound on the 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 7.1 ([1, 3, 4]) *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]

[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.

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 7.1 with our reduction from GAP-HAMMING to estimating F_∞, we've proved the following.

Theorem 7.2 *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.

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 7.2 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 7.2 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 7.1 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.

References

[1] T. S. Jayram, R. Kumar, and D. Sivakumar. The one-way communication complexity of hamming distance. *Theory of Computing*, 4:129–135, 2008.

[2] I. Kremer, N. Nisan, and Ron D. On randomized one-round communication complexity. *Computational Complexity*, 8(1):21–49, 1999.

[3] D. P. Woodruff. Optimal space lower bounds for all frequency moments. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 167–175, 2004.

[4] D. P. Woodruff. *Efficient and Private Distance Approximation in the Communication and Streaming Models*. PhD thesis, MIT, 2007.

[5] A. C.-C. Yao. Lower bounds by probabilistic arguments. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 420–428, 1983.