CS369E: Communication Complexity (for Algorithm Designers) Lecture #8: Lower Bounds in Property Testing[*]

Tim Roughgarden[†]

March 12, 2015

1 Property Testing

We begin in this section with a brief introduction to the field of property testing. Section 2 explains the famous example of "linearity testing." Section 3 gives upper bounds for the canonical problem of "monotonicity testing," and Section 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 [14] for a starting point.

[*]©2015, Tim Roughgarden.

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

[1]Somewhat amazingly, this connection was only discovered in 2011 [1], even though the connection is simple and property testing is a relatively mature field.

In the standard property testing model, one has "black-box access" to a function $f : D \rightarrow R$. 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.

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.

2 Example: The BLR Linearity Test

The unofficial beginning of the field of property testing is [2]. (For the official beginning, see [12, 15].) 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 2.1 ([2]) *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 2.1 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 [8, Chapter 1] for a good exposition. There are also more direct proofs of Theorem 2.1, as in [2]. 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 2.1 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 2.1 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 2.1 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.

3 Monotonicity Testing: Upper Bounds

The problem of monotonicity testing was introduced in [11] 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 3.1 and 3.2. We discuss the case of larger ranges, where communication complexity has been used to prove strong lower bounds, in Section 3.3.

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:

$$f(0, \mathbf{x}_{-i}) \le f(1, \mathbf{x}_{-i})$$

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 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.

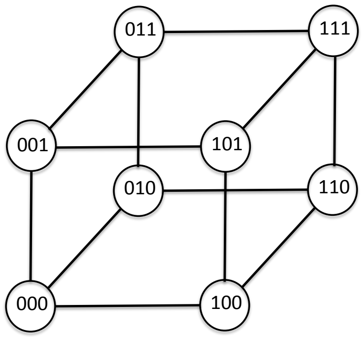


Figure 1: $\{0,1\}^n$ can be visualized as an n-dimensional hypercube.

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 3.1 ([11]) *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 $\frac{\epsilon}{n}$.

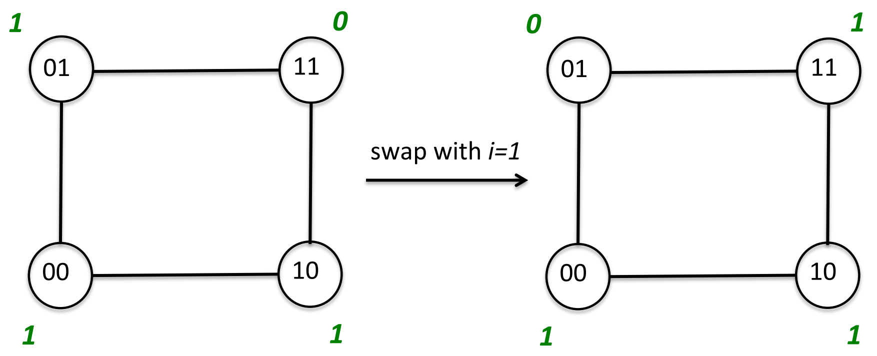
Fix an arbitrary function f. There are two quantities that we need to relate to each other — the rejection probability of f, and the distance between f and the set of monotone functions. We do this by relating both quantities to the sizes of the following sets: for $i = 1, 2, \ldots, n$, define

$$|A_i| = \{\mathbf{x}_{-i} \in \{0,1\}^{n-1} \,:\, f(0, \mathbf{x}_{-i}) > f(1, \mathbf{x}_{-i})\}. \tag{1}$$

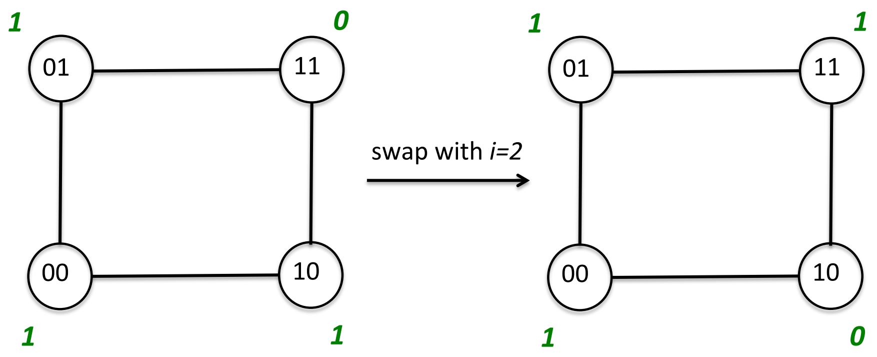
That is, A_i is the edges of the ith slice of the hypercube across which f violates monotonicity. By the definition of the edge tester, the probability that a single trial rejects f is exactly

$$\underbrace{\sum_{i=1}^{n} |A_i|}_{\#\text{ of violations}} \;\Big/\; \underbrace{n2^{n-1}}_{\#\text{ of edges}}. \tag{2}$$

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,



(a) Fixing the first slice



(b) Fixing the second slice

Figure 2: Swapping values to eliminate the monotonicity violations in the ith slice.

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 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 2). The claim asserts that the overall number of violations cannot increase (cf., Figure 2).

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 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

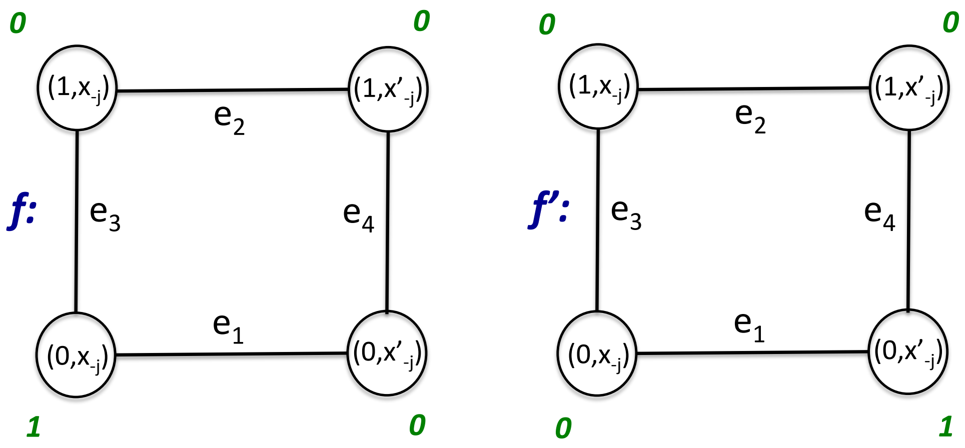


Figure 3: The number of monotonicity violations on edges e_3 and e_4 is at least as large under f as under f'.

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.

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 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]

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

[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?

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 (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. ■

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 3.1. The analysis in Theorem 3.1 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 [4] gave the first improved upper bounds, of $\tilde{O}(n^{7/8}/\epsilon^{3/2})$.[4] A year later, Chen et al. [6] gave an upper bound of $\tilde{O}(n^{5/6}/\epsilon^4)$. Just a couple of months ago, Knot et al. [13] 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 3.1), 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 [4, 6, 13]. 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 [9]; see also the Exercises. For non-adaptive testers with two-sided error, Chen et al. [6] proved a lower bound of $\tilde{\Omega}(n^{1/5})$ and Chen et al. [5] 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.

[4]The notation $\tilde{O}(\cdot)$ suppresses logarithmic factors.

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 4 that it is useful for proving lower bounds in the case where the range is relatively large.

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 3.1, 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 (2).

We need to revisit the major step of the proof of Theorem 3.1, 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 $O(n|R|/\epsilon)$ [11].

A divide-and-conquer approach gives a better upped bound [7]. 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]

[5]At the very least, some of the techniques we've learned in previous lectures are useful. The arguments in [6, 5] use an analog of Yao's Lemma (Lecture #4) 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.

[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.

4 Monotonicity Testing: Lower Bounds

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 4.1 ([1]) *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 4.1 separates the case of a general range R from the case of a Boolean range, where $\tilde{O}(\sqrt{n})$ queries are enough [13]. With the right communication complexity tools, Theorem 4.1 is not very hard to prove. Simultaneously with [1], Briët et al. [3] 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 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 (Lecture #5), we proved that the nondeterministic communication complexity of the problem is $\Omega(n)$ using a covering argument with a clever inductive proof. In our boot camp (Lecture #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 4.1 is the following lemma.

Lemma 4.2 *Fix sets $A, B \subseteq U = \{1, 2, \ldots, n\}$. Define the function $h_{AB} : 2^U \to \mathbb{Z}$ by*

$$h_{AB}(S) = 2|S| + (-1)^{|S \cap A|} + (-1)^{|S \cap B|}. \qquad (3)$$

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 4.1. 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.

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 4.2, 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 4.1.

Proof of Lemma 4.2: 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 (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.

[7]As usual, we're not distinguishing between subsets of $\{1, 2, \ldots, n\}$ and their characteristic vectors.

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 3.1, 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. ■

4.2 Extension to Smaller Ranges

Recalling the definition (3) of the function h_{AB}, we see that the proof of Theorem 4.1 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 4.2 still holds for h'_{AB}, with the "$\frac{1}{8}$" in case (ii) replaced by "$\frac{1}{16}$;" the new version of Theorem 4.1 then follows. Checking that case (i) in Lemma 4.2 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.

5 A General Approach

It should be clear from the proof of Theorem 4.1 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 4.1, $d = 2$.)

Via the simulation argument in the proof of Theorem 4.1, 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 [1, 10] for several examples.

A large chunk of the property testing literature is about testing graph properties [12]. An interesting open question is if communication complexity can be used to prove strong lower bounds for such problems.

References

[1] E. Blais, J. Brody, and K. Matulef. Property testing lower bounds via communication complexity. *Computational Complexity*, 21(2):311–358, 2012.

[2] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, 1993.

[3] J. Briët, S. Chakraborty, D. García-Soriano, and A. Matsliah. Monotonicity testing and shortest-path routing on the cube. *Combinatorica*, 32(1):35–53, 2012.

[4] D. Chakrabarty and C. Seshadhri. 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, 2013.

[5] X. Chen, A. De, R. A. Servedio, and L.-Y. Tan. 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, 2015.

[6] X. Chen, R. A. Servedio, and L.-Y. Tan. New algorithms and lower bounds for monotonicity testing. In *Proceedings of the 55th Symposium on Foundations of Computer Science (FOCS)*, pages 286–295, 2014.

[7] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron, and A. Samorodnitsky. Improved testing algorithms for monotonicity. In *Proceedings of APPROX-RANDOM*, pages 97–108, 1999.

[8] R. O' Donnell. *Analysis of Boolean Functions.* Cambridge, 2014.

[8] There is an analogous argument that uses 1-way communication complexity lower bounds to derive query complexity lower bounds for *non-adaptive* testers; see the Exercises.

[9] E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld, and A. Samorodnitsky. Monotonicity testing over general poset domains. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 474–483, 2002.

[10] O. Goldreich. On the communication complexity methodology for proving lower bounds on the query complexity of property testing. *Electronic Colloquium on Computational Complexity (ECCC)*, 20, 2013.

[11] O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samorodnitsky. Testing monotonicity. *Combinatorica*, 20(3):301–337, 2000.

[12] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998.

[13] S. Khot, D. Minzer, and M. Safra. On monotonicity testing and Boolean isoperimetric type theorems. In *Proceedings of the 56th Symposium on Foundations of Computer Science (FOCS)*, 2015. To appear.

[14] D. Ron. Property testing: A learning theory perspective. *Foundations and Trends in Theoretical Computer Science*, 5(2):73–205–402, 2010.

[15] R. Rubinfeld and M. Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.