6.5240 — Scribe Notes Fall 2026

Lecture 11: Monotonicity Testing

Date: Oct 20, 2026

Lecturer: Ronitt Rubinfeld Scribe: Liyan Chen

We continue our discussion of monotonicity testing. Last time, we talked about testing sortedness of a sequence: testing if a sequence $y_1, \ldots, y_n$ is close to sorted. We showed how to achieve query complexity $O(\log n/\epsilon)$; see also Ergün et al. [1]. This logarithmic dependence on n is optimal [2].

1 Boolean Cube

We say a function $f : \{0,1\}^n \to \{0,1\}$ on boolean cube is monotone, if $\forall x \prec y$, $f(x) \leq f(y)$. Here for $x, y \in \{0,1\}^n$, $x \prec y$ means $\forall i, x_i \leq y_i$.

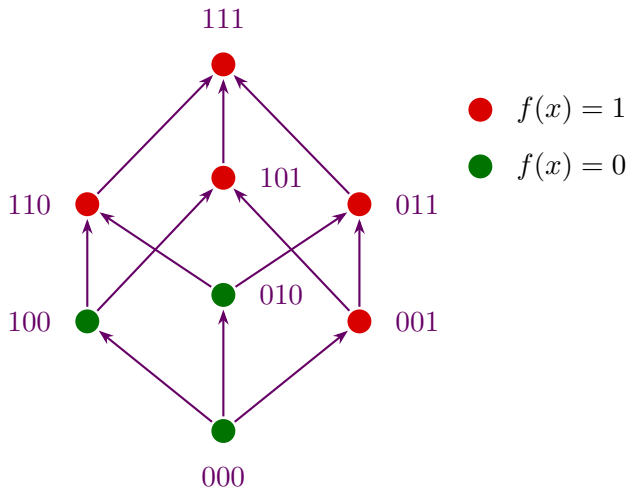


Figure 1: A monotone Boolean function on $\{0,1\}^3$: $f(x) = x_3 \vee (x_1 \wedge x_2)$. Arrows follow the coordinatewise order.

On the Boolean cube, we say an edge $x \to y$ is *violated* by f if $f(x) > f(y)$.

1.1 Property Testing

We consider property testers that make oracle queries to f. Today we show a property tester that makes $O(n/\epsilon)$ queries for distance parameter ϵ. Later Khot, Minzer, and Safra [3] give a nonadaptive tester with one-sided error using $\widetilde{O}(\sqrt{n}/\epsilon^2)$ queries.

1.2 Algorithm and Analysis

Algorithm 1: Edge tester for monotonicity on the Boolean cube

Input: Oracle access to $f : \{0,1\}^n \to \{0,1\}$, where $n \geq 1$; distance parameter $\varepsilon \in (0,1)$.

Output: Accept or Reject.

```
for t = 1, ..., ⌈2n/ε⌉ do
    Sample an upward edge (x, y) of the Boolean cube uniformly at random,
     independently of previous samples
    Query f(x) and f(y)
    if f(x) > f(y) then
        return Reject
return Accept
```

The completeness of the algorithm follows trivially. For analyze its soundness, we need to show that, if f is ϵ-far from monotone, a $\geq \epsilon/n$ fraction of edges are violated by f. In other words, let $V(f)$ be the number of edges violated by f, it suffices to show that, if $V(f) < \epsilon 2^{n-1}$, f can be made monotone by changing less than $\epsilon 2^n$ values. In fact, we show the following.

Lemma 1.1. *For any $f : \{0,1\}^n \to \{0,1\}$, there exists monotone $f' : \{0,1\}^n \to \{0,1\}$ of distance at most $2V(f)$ from f.*

To construct f', we repair violated edges one dimension at a time. Starting from $f' = f$, we process dimensions $1, \ldots, n$. In each dimension, we swap the current values of f' at the endpoints of every violated edge. Edges in a fixed dimension are disjoint, so their processing order does not matter. The subroutine Repair(i) updates the shared function f' in place.

Algorithm 2: Repairing f to construct a monotone function f'

Input: A function $f : \{0,1\}^n \to \{0,1\}$.

Output: A monotone function $f' : \{0,1\}^n \to \{0,1\}$.

```
Function Repair(i):
    foreach x ∈ {0,1}^n with x_i = 0 do
        y ← (x_1, ..., x_{i-1}, 1, x_{i+1}, ..., x_n)
        if f'(x) > f'(y) then
            (f'(x), f'(y)) ← (f'(y), f'(x))
f' ← f
for i = 1, ..., n do
    Repair(i)
return f'
```

To show this repairing procedure works, we show that repairing dimension i does not interfere with another dimension $j \neq i$. In particular, with V_j being the number of violated edges in dimension j, we show that Repair(i) does not increase V_j for $i \neq j$.

Lemma 1.2. *For any $i \neq j$, running* Repair(i) *does not increase V_j.*

Proof. WLOG we assume $i = 1, j = 2$. Since we are only consider edges in dimension $1, 2$, we can fix any $y \in \{0,1\}^{n-2}$, and only consider the square $\{x = z||y : z \in \{0,1\}^2\}$.

Case 1: No violated edges in dimension 1 In this case, Repair(1) does not do anything.

Case 2: Both edges in dimension 1 are violated In this case, for both edges f on endpoints are $1, 0$ respectively. Therefore the dimension 2 edges are not violated before and after Repair(1).

Case 3: One edge in dimension 1 is violated, and the other edge is (v, v) In this case, the number of violated edges in dimension 2 would not changed.

Case 4: One edge in dimension 1 is violated, and the other edge is $(0, 1)$ In this case, after swap number of violated edges in both dimension 1 and dimension 2 are decreased. □

Given the above lemma, f' is monotone since after running Repair(i) V_i is reduced to 0, and it is never increased later. The number of values changed from f to f' is at most

$$\begin{aligned}&\sum_{i=1}^{n} 2 \cdot (V_i \text{ when we run } \mathsf{Repair}(i))\\ \leq &\sum_{i=1}^{n} 2 \cdot (V_i \text{ in the beginning}) = \sum_{i=1}^{n} 2V_i\\ = &2V(f).\end{aligned}$$

2 Testing via "Implicit Learning"

Consider the following simple problem: given a string $y = y_1 \cdots y_n \in \{0,1\}^n$, test if it is monotone. We say y is ϵ-far from monotone if we must change at least ϵn bits to make it monotone.

An easy way to design property tester is by "implicit learning": we know that in the YES case, y has to be of the form $0^s||1^{n-s}$. Therefore, the tester can first learn an approximation $\tilde{y}$ of y with few queries, and then test if y is close to $\tilde{y}$. In this problem, the two steps proceeds as follows:

1. Query y at locations $i_1, \ldots, i_{2/\epsilon}$ for $i_j = j \cdot \epsilon n/2$, reject if not monotone. Let i_j be the leftmost index for which $y_{i_j} = 1$, so $y_{i_{j-1}} = 0$. (If $y_{i_1} = 1$ we set $i_0 = 0$, and if all the answers are 0 we set $i_{j-1} = n$ and $i_j = n + 1$.)
2. Take $O(1/\epsilon)$ (say $4/\epsilon$) new uniformly random samples $i \in [n]$ and reject if $y_i = 0$ but $i \geq i_j$, or $y_i = 1$ yet $i \leq i_{j-1}$. Accept if we never reject.

This tester makes $2/\epsilon + 4/\epsilon = O(1/\epsilon)$ queries (we ignore rounding throughout). The $\Omega(\log n)$ lower bound for sortedness [2] does not apply here, since the values are only 0 and 1.

Theorem 2.1. *If y is monotone, the tester always accepts. If y is ϵ-far from monotone, the tester rejects with probability at least 3/4.*

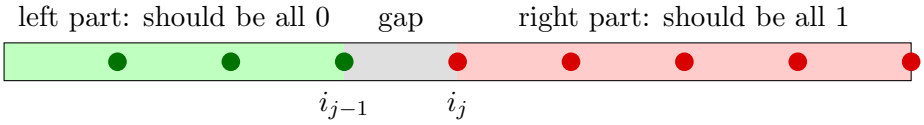


Figure 2: After Step 1, the query locations (dots: green for answer 0, red for answer 1) split $[n]$ into three parts. The gap has fewer than $\epsilon n/2$ indices.

Proof. Monotone case. Suppose $y = 0^s 1^{n-s}$. The answers in Step 1 are monotone, so Step 1 does not reject. Every index to the right of a 1 holds a 1, and every index to the left of a 0 holds a 0, so no sample in Step 2 can make us reject.

Far case. Suppose y is ϵ-far from monotone. If Step 1 rejects we are done, so assume it does not. Then Step 1 splits $[n]$ into three parts, as in Figure 2:

- the *left part*, $i \leq i_{j-1}$, which should be all 0;
- the *gap*, $i_{j-1} < i < i_j$, which we know nothing about. It has fewer than $\epsilon n/2$ indices, since neighboring query locations are $\epsilon n/2$ apart;
- the *right part*, $i \geq i_j$, which should be all 1.

Call an index *bad* if it is in the left part and holds a 1, or it is in the right part and holds a 0. Step 2 rejects exactly when it samples a bad index, so we want to show that there are many bad indices.

Here is the learned approximation: let $\tilde{y}$ be 0 on the left part and 1 on the gap and the right part. Then $\tilde{y}$ is monotone, and it differs from y only on the bad indices and possibly on the gap. Since y is ϵ-far from monotone,

$$\epsilon n \leq \text{dist}(y, \tilde{y}) \leq \#\{\text{bad indices}\} + \#\{\text{gap indices}\} < \#\{\text{bad indices}\} + \epsilon n/2,$$

so more than $\epsilon n/2$ indices are bad. Therefore, each sample in Step 2 is bad with probability more than $\epsilon/2$, independently, and

$$\Pr[\text{no sample is bad}] \leq (1 - \epsilon/2)^{4/\epsilon} \leq e^{-2} < 1/4. \qquad \square$$

3 Testing L1-distance

Given a function $f : [n] \to [0, 1] \subset \mathbb{R}$, we want our property tester to pass if f is monotone, and fail if f is ϵ-far in L_1 distance from any monotone function. Here, the distance between function f, g is defined as $\sum_{i=1}^{n} |f(i) - g(i)|$, and f is ϵ-far if this distance is at least ϵn for every monotone g.

A note on distance. In this problem, we consider L_1 distance instead of Hamming distance that we focus on before. For functions with discrete output $f : [n] \to \{0, 1\}$, the two distance measures are equivalent. For functions with real number output as here, Hamming distance is at least L_1 distance. So if f is ϵ-far in L_1 distance, it is also ϵ-far in Hamming distance. This means an L_1 tester has fewer functions to reject, so designing a tester for L_1 distance is an easier problem: any tester for Hamming distance also works for L_1 distance.

3.1 Testing Algorithm

Our goal is to design a tester for L_1 distance in $O(1/\epsilon)$ queries.

For comparison, testing real-valued functions with respect to Hamming distance needs $\Omega(\log n)$ queries [2]. The algorithm below queries the same locations as the tester in Section 2, and simply checks that the answers are monotone.

Algorithm 3: L_1 tester for monotonicity on $[n]$

Input: Oracle access to $f : [n] \to [0, 1]$; distance parameter $\varepsilon \in (0, 1)$.
Output: Accept or Reject.

$Q_1 \leftarrow \{j \cdot \varepsilon n/2 : j = 1, \ldots, 2/\varepsilon\}$
$Q_2 \leftarrow$ a set of $4/\varepsilon$ independent uniformly random samples from $[n]$
Query f on all of $Q \leftarrow Q_1 \cup Q_2$
if *there are* $x < y$ *in* Q *with* $f(x) > f(y)$ **then**
    return Reject
return Accept

Theorem 3.1. *Algorithm 3 makes $O(1/\epsilon)$ queries. If f is monotone, it always accepts. If f is ϵ-far from monotone in L_1 distance, it rejects with probability at least $3/4$.*

The idea of the proof is to reduce to Boolean monotonicity testing, by looking at f one threshold at a time. In the following sections we introduce the lemmas to prove the theorem.

3.2 Threshold Functions

Definition 3.2. For $t \in [0, 1]$, we define the threshold function $f_{(t)} : [n] \to \{0, 1\}$ that

$$f_{(t)}(x) = \begin{cases} 1, & \text{if } f(x) \geq t, \\ 0, & \text{otherwise.} \end{cases}$$

A simple observation is, for any function f,

$$f(x) = \int_0^{f(x)} 1 \mathsf{d}t = \int_0^1 f_{(t)}(x) \mathsf{d}t.$$

The last equality holds since $f_{(t)}(x) = 1$ exactly when $t \le f(x)$. So every f, monotone or not, is an average of Boolean functions; see Figure 3.

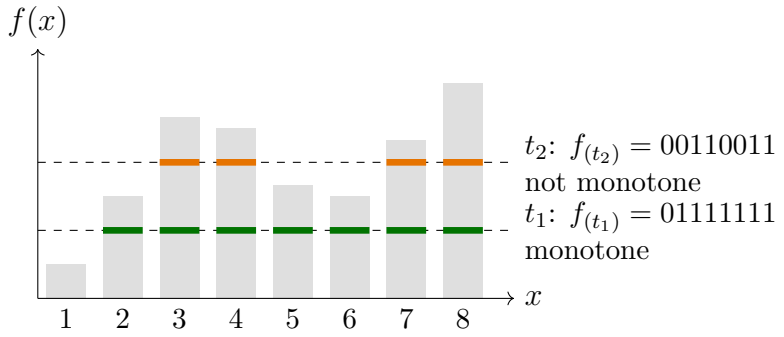


Figure 3: A function f on [8] (bars) that is not monotone, and two of its threshold functions. The thick segments on each dashed line mark where the threshold function is 1.

We define $L_1(f, \mathcal{M})$ as the minimum L_1 distance of f to monotone functions. For a Boolean function like $f_{(t)}$, we only compare to Boolean monotone functions, so $L_1(f_{(t)}, \mathcal{M})$ is simply the Hamming distance from $f_{(t)}$ to monotone. We will prove the following interesting lemma.

Lemma 3.3. $L_1(f, \mathcal{M}) = \int_0^1 L_1(f_{(t)}, \mathcal{M}) \mathsf{d}t$.

That is, we can express the distance to monotonicity in terms of the distances of the threshold functions. Think of each $f_{(t)}$ as a horizontal "row" of f: the cost of making f monotone is exactly the total cost of making each row monotone.

We use one more simple fact. For $a, b \in [0, 1]$, the bits $\mathbf{1}\{a \ge t\}$ and $\mathbf{1}\{b \ge t\}$ differ exactly when t is between a and b, so $|a - b| = \int_0^1 |\mathbf{1}\{a \ge t\} - \mathbf{1}\{b \ge t\}| \, \mathsf{d}t$. Writing $\|f - g\|_1 = \sum_x |f(x) - g(x)|$ and summing over all x, we get for any $f, g : [n] \to [0, 1]$ that

$$\|f - g\|_1 = \int_0^1 \left\|f_{(t)} - g_{(t)}\right\|_1 \mathsf{d}t. \tag{3.1}$$

Proof of Lemma 3.3. We show the two directions separately.

($\ge$) Making f monotone makes every row monotone. Let g be the closest monotone function to f, then each $g_{(t)}$ is monotone. So $\left\|f_{(t)} - g_{(t)}\right\|_1 \ge L_1(f_{(t)}, \mathcal{M})$ for every t, and by (3.1),

$$L_1(f, \mathcal{M}) = \|f - g\|_1 = \int_0^1 \left\|f_{(t)} - g_{(t)}\right\|_1 \mathsf{d}t \ge \int_0^1 L_1(f_{(t)}, \mathcal{M}) \mathsf{d}t.$$

(≤) Making every row monotone makes f monotone. For each t, let g_t be the closest monotone Boolean function to $f_{(t)}$, and let $g(x) = \int_0^1 g_t(x)\mathsf{d}t$. Then g is monotone, since it is an average of monotone functions. Using $f(x) = \int_0^1 f_{(t)}(x)\mathsf{d}t$,

$$\begin{aligned} L_1(f, \mathcal{M}) \leq \|f - g\|_1 &= \sum_x \left| \int_0^1 \left(f_{(t)}(x) - g_t(x)\right) \mathsf{d}t \right| \\ &\leq \sum_x \int_0^1 \left| f_{(t)}(x) - g_t(x) \right| \mathsf{d}t = \int_0^1 L_1(f_{(t)}, \mathcal{M})\mathsf{d}t. \end{aligned}$$

□

3.3 From Boolean Hamming Testers to L_1 Testers

Lemma 3.3 enables us to upgrade a tester for *Hamming distance* on Boolean functions into a tester for *L1 distance* on $[0,1]$-valued functions. Furthermore, the reduction preserves non-adaptivity.

Recall that a tester T is *nonadaptive* if it picks its whole query set $Q \subseteq [n]$ at random before seeing any answers, and it has *one-sided error* if it always passes monotone functions. (As usual, it must also reject ϵ-far functions with probability at least 3/4.) Such a T must pass whenever the answers are monotone on Q, since they could have come from a monotone function. And rejecting in all other cases can only help. So we can assume that T passes if and only if f is monotone on Q, that is, there are no $x < y$ in Q with $f(x) > f(y)$.

Lemma 3.4. *If T is a nonadaptive tester with one-sided error for monotonicity of $f : [n] \to \{0,1\}$ with respect to Hamming distance, then T (exactly the same test) is also a nonadaptive tester with one-sided error for monotonicity of $f : [n] \to [0,1]$ with respect to L_1 distance.*

Proof. Take any $f : [n] \to [0,1]$. If f is monotone, then it is monotone on every Q, so T always passes.

Now suppose $L_1(f, \mathcal{M}) \geq \epsilon n$. By Lemma 3.3,

$$\int_0^1 L_1(f_{(t)}, \mathcal{M})\mathsf{d}t = L_1(f, \mathcal{M}) \geq \epsilon n.$$

The left side is the average value of $L_1(f_{(t)}, \mathcal{M})$ over t, so there is some t^* with $L_1(f_{(t^*)}, \mathcal{M}) \geq \epsilon n$. Since $f_{(t^*)}$ is Boolean, this says that $f_{(t^*)}$ is ϵ-far from monotone in Hamming distance. So with probability at least 3/4 over the choice of Q, T rejects $f_{(t^*)}$, which means that Q contains $x < y$ with $f_{(t^*)}(x) = 1$ and $f_{(t^*)}(y) = 0$. By the definition of thresholds,

$$f(x) \geq t^* > f(y),$$

so the same pair x, y violates monotonicity of f itself, and T rejects f on the same Q. Hence T rejects f with probability at least 3/4. □

Notice that the tester never needs to know t^*; it only appears in the analysis.

3.4 Analysis of the Test Algorithm

To use Lemma 3.4, we use the Boolean tester from Section 2, with a change in the decision rule: query the same set Q, and reject whenever the function is not monotone on Q.

Proof of Theorem 3.1. The new Boolean test still always passes monotone strings. It also rejects whenever the old test does, since each rejection of the old test comes with a violated pair in Q: a violation among the locations of Step 1, or a sample $i > i_j$ with $y_{i_j} = 1 > y_i = 0$, or a sample $i < i_{j-1}$ with $y_i = 1 > y_{i_{j-1}} = 0$. So by Theorem 2.1, it rejects ϵ-far strings with probability at least 3/4.

Thus the new test is a nonadaptive tester with one-sided error for $y : [n] \to \{0, 1\}$ with respect to Hamming distance. Running the exact same test on $f : [n] \to [0, 1]$ is Algorithm 3, so Lemma 3.4 says it is a tester with respect to L_1 distance. It makes $2/\epsilon + 4/\epsilon = O(1/\epsilon)$ queries. □

References

[1] Funda Ergün, Sampath Kannan, S. Ravi Kumar, Ronitt Rubinfeld, and Mahesh Viswanathan. Spot-checkers. *Journal of Computer and System Sciences*, 60(3):717–751, 2000. doi:10.1006/jcss.1999.1692. Author manuscript.

[2] Eldar Fischer. On the strength of comparisons in property testing. *Information and Computation*, 189(1):107–116, 2004. doi:10.1016/j.ic.2003.09.003. ECCC preprint.

[3] Subhash Khot, Dor Minzer, and Muli Safra. On monotonicity testing and Boolean isoperimetric-type theorems. *SIAM Journal on Computing*, 47(6):2238–2276, 2018. doi:10.1137/16M1065872. ECCC preprint.

[4] Piotr Berman, Sofya Raskhodnikova, and Grigory Yaroslavtsev. L_p-testing. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 164–173, 2014. doi:10.1145/2591796.2591887. Author manuscript.