Sublinear Algorithms	March 15, 2012
Lecture 17	
Lecturer: Sofya Raskhodnikova	*Scribe(s): Youngtae Youn*

The lecture 2 covered a monotonicity tester for $f : \{0,1\}^n \to \{0,1\}$ with query complexity $2n/\epsilon$. In this lecture, we present another monotonicity tester for slightly more general functions $f : \{0,1\}^n \to \mathcal{R}$ whose query complexity is $\frac{2n \log |\mathcal{R}|}{\epsilon}$ [2].

1 A Monotonicity Tester for $f : \{0,1\}^n \to \{0,1\}$

A monotonicity tester for a function $f : \{0,1\}^n \to \{0,1\}$ works as follows.

1. Pick t edges uniformly at random from $\mathcal{H}_n$.
2. If any of these edges are violated, reject. Otherwise, accept.

If f is monotone and has no violated edges, the tester surely accepts the input. Still, it remains to be seen how many edges the tester must sample to reject with probability at least $2/3$ the functions which are ϵ-far from monotone. First of all, the number of samples t is related to the probability that a tester catches a witness as follows.

Lemma 1 (Witness Lemma)**.** *If a tester catches a witness with probability p, then $t = 2/p$ iterations catches a witness with probability at least 2/3.*

Now we need to compute the probability p that a tester catches a violated edges via random sampling. This relation is usually stated as a contrapositive:

Lemma 2 (Repair Lemma)**.** *If f has v violated edges, then at most $\kappa(v)$ node labels must be changed to make f monotone.*

Assume that we obtain $\kappa(v)$ for an n-dimensional hypercube $\mathcal{H}_n$. Note that $\mathcal{H}_n$ has 2^n vertices and $n2^{n-1}$ edges. As f is ϵ-far from monotone, $\kappa(v) = \epsilon \cdot 2^n$. Meanwhile, the probability p that a tester catches a witness is $\frac{v}{n2^{n-1}}$ as v edges are violated out of $n2^{n-1}$ edges. Then p can be written in $\kappa(v)$ as follows.

$$p = \frac{v}{n2^{n-1}} = \frac{v}{n\frac{\kappa(v)}{\epsilon}\frac{1}{2}} = \frac{2v\epsilon}{n\kappa(v)}$$

Finally, we compute the number of samples t:

$$t = 2/p = \frac{n\kappa(v)}{v\epsilon}. \tag{1}$$

Note that this analysis is entirely based on the fact that f is defined on a hypercube $\mathcal{H}_n$.

Computing $\kappa(v)$. Now the problem boils down to compute $\kappa(v)$, the number of node labels that must be changed to repair v violated edges such that f is monotone. In case of $f : \{0,1\}^n \to \{0,1\}$, we have seen in the lecture 2 that $\kappa(v) = 2v$ which leads to a monotonicity tester with query complexity $t = \frac{n \cdot 2v}{v\epsilon} = 2n/\epsilon$.[1]

The repair procedure is simple – repair each violated edge in one dimension at a time by swapping its end points. As each violated edge contributes two end points, $2v$ labels will be modified. And we have seen that swapping in the dimension i does not increase the number of violated edges for all other dimensions.

[1] A careful analysis shows that $\kappa(v)$ can be improved to v so that the query complexity is n/ϵ.

2 A Monotonicity Tester for $f : \{0,1\}^n \to \mathcal{R}$

Why New Fixing Procedure? Even though the function range is generalized from $\{0,1\}$ to $\mathcal{R}$, the idea behind the construction of a tester remains the same – pick a number of edges uniformly at random and check whether any of them are violated. The problem is that the previous repair procedure does *not* work. For example, consider the following function on $\mathcal{H}_2$ in Fig 1a where violated edges are drawn by dashed arrows.

(a) before fixing (b) after fixing

Figure 1: The previous repair procedure does not work.

The function has two violated edges $1 \to 0$ and $2 \to 0$. Suppose that we try to fix violation in the horizontal dimension by swapping the end points of the violated edge $1 \to 0$. Unfortunately, it induces another violation $1 \to 0$ in the vertical dimension, as shown in Fig 1b. And the correctness of the previous fixing mechanism is based on the fact that fixing a violated edge in one dimension does not increase the number of violations in other dimensions. Hence, we need to come up with a new repair mechanism and compute $\kappa(v)$ in this mechanism.

A New Fixing Procedure. Without loss of generality, we can assume that $\mathcal{R} = \{0, 1, ..., 2^s - 1\}$. We will show that one can repair v violated edges by changing $2vs = 2v \log |\mathcal{R}|$ labels.[2] As $\kappa(v) = vs = 2v \log |\mathcal{R}|$, the query complexity of the tester is $\frac{2n}{\epsilon} \log |\mathcal{R}|$ by (1).

Theorem 3. *If $f : \{0,1\}^n \to \mathcal{R} = \{0, 1, ..., 2^s - 1\}$ has v violated edges, then at most $2vs$ node labels must be changed to make f monotone.*

We will prove it by the induction on s. The base case is trivial. If $s = 1$, the range $\mathcal{R}$ reduces to $\{0,1\}$ and we have seen in the previous lecture that changing at most $2v$ node labels by swapping the endpoints of each violated edge is enough to make f monotone. To handle the induction step, we devise a divide-and-conquer type repair mechanism. And this mechanism is based on the operator `clear`.

Operator `Clear`. A function f on a hypercube $\mathcal{H}_n$ has $n \cdot 2^{n-1}$ edges. Until now we have paid attention to the dimensions of each edge on a hypercube. Instead, we change our view and pay attention to the values of end points in each edge. As the range is restricted to $[0, 2^s - 1]$, we can pack all the $n \cdot 2^{n-1}$ directed edges on an interval $[0, 2^s - 1]$. For convenience, we only pack violated edges on the interval and show how to make the function monotone by changing at most $2vs$ labels.

Suppose we are given an interval $[a, b]$. Let us call a the *left barrier* and b the *right barrier*. Then the operator `clear`(a, b) guarantees to modify the function such that:

1. no new violated edges are introduced,
2. any violated edges $x \to y$ with x and y between the barriers are fixed,
3. each violated edge $x \to y$ with $x > b$ and $y \leq b$ is replaced with $x \to b$, and similarly
4. each violated edge $x \to y$ with $y < a$ and $x \in [a, b]$ is replaced with $a \to y$.

[2] Throughout the note, the base of log is 2.

Observe that `clear`(a, b) fixes the violated edges whose both end points are in $[a, b]$ while it replaces other violated edges with less *harmful* violated edges. A violated edge becomes less harmful if its length shrinks such that it does not span across the barriers. While doing so, this operator introduces no new violated edges.

Divide-and-Conquer. Assume that the function has w violated edges whose both end points are inside the barriers $[a, b]$. Mathematical induction will show that the `clear`(a, b) operator fixes w violations by modifying $2w \cdot \log(b - a + 1)$ node labels. A divide-and-conquer type approach will fix the v violated edges in the range of $[0, 2^s - 1]$ as follows.

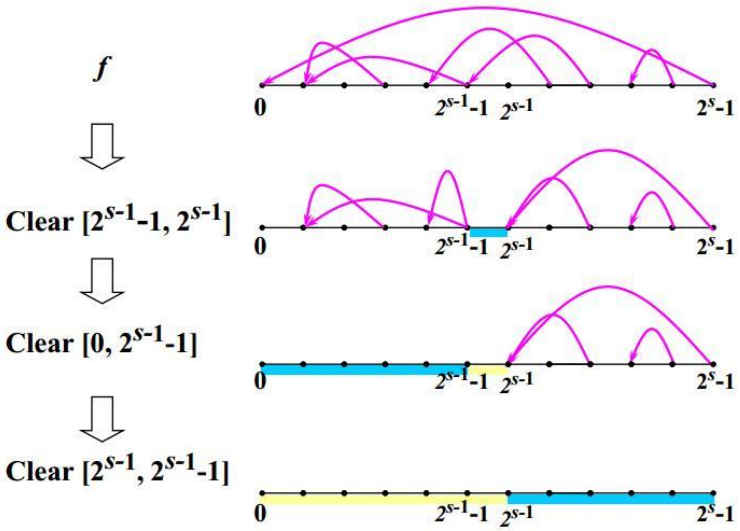

Figure 2: A divide-and-conquer approach to fix violated edges

1. `clear`$(2^{s-1}-1, 2^{s-1})$ corresponds to a base case. It fixes the violated edges in the interval $[2^{s-1}-1, 2^{s-1}]$ by changing at most $2v \cdot \log(2^{s-1} - (2^{s-1} - 1) + 1) = 2v \cdot \log 2 = 2v$ node labels based on the previous fixing procedure since the range is restricted to have width 1.

   Meanwhile, the original v violated edges are modified so that they do not span beyond the barriers. Suppose that there are ℓ and r modified violated edges left to the barrier $2^{s-1} - 1$ and right to the barrier 2^{s-1}, respectively. As no new violation is introduced, we have $\ell + r \leq v$.

2. `clear`$(0, 2^{s-1} - 1)$ deals with ℓ violated edges in the range which shrinks by half. By the induction hypothesis, `clear`$(0, 2^{s-1} - 1)$ fixes ℓ violations by changing at most $2\ell \cdot \log(2^{s-1} - 1 + 1) = 2\ell(s - 1)$ node labels.

3. `clear`$(2^{s-1}, 2^s - 1)$ deals with r violated edges and the similar reasoning shows that it fixes r violations by changing at most $2r(s - 1)$ node labels.

Summing up all the number of node label changing, one can easily get the upper bound

$$2v + 2\ell(s - 1) + 2r(s - 1) = 2v + 2(\ell + r)(s - 1) = 2vs,$$

which was required. [3]

[3] This upper bound can be improved to vs by a factor of 2 as we did in the previous tester for $f : \{0, 1\}^n \to \{0, 1\}$.

How to Implement `Clear`. At the heart of `clear`(a, b) operator lies `squash`(f, a, b) operator which modifies the function f such that no violated edges span across the barrier a and b.

$$\texttt{squash}(f, a, b) = \begin{cases} a & \text{if } f(x) \leq a \\ b & \text{if } f(x) \geq b \\ f(x) & \text{otherwise.} \end{cases}$$

The `clear` operator is implemented by applying `squash` operator, repairing the function into the closest monotone function with the squashed range, and applying `unsquash` operator as shown in Fig 3. Now the

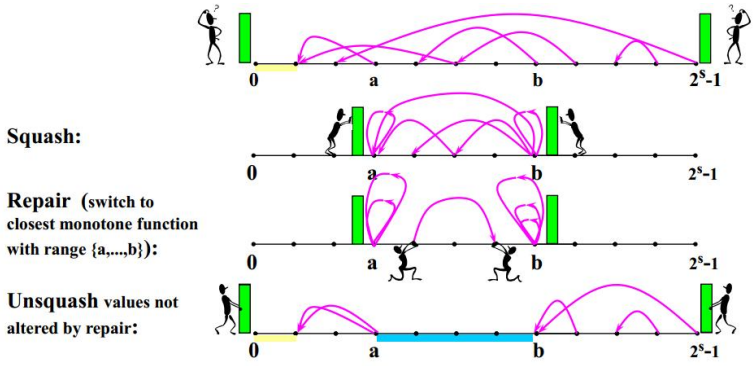

Figure 3: How to implement `clear`(a, b)

problem boils down to showing that this implementation satisfies all the requirements in page 2.

1. *No new violated edge is introduced.*
   There are four kinds of non-violated edges: (a) none of its end point is inside the barriers, (b) only the left end point is inside the barriers, (c) only the right end point is inside the barriers, and (d) both of its end points are inside the barriers.

   Any edge with both of its end points outside of the barrier remains intact. For the second kind, consider an edge $x \to y$ with $x \in [a, b]$ and $y > b$. Even though the labels of x changes to $x' \in [a, b]$ during the `clear` operation, $x' \to y$ is still not violated as $x' \leq y$. The same reasoning would show that the other two cases (c) and (d) do not induce new violated edges.

2. *Any violated edges with both of their end points inside the barriers are fixed.*
   This is guaranteed by the `Repair` procedure in Fig 3.

3. *Each violated edge $x \to y$ with $x > b$ and $y \in [a, b]$ is replaced with $x \to b$.*

   The `squash` operator changes a violated edge $x \to y$ into a less harmful violated edge $x \to b$.

4. *Each violated edge $x \to y$ with $y < a$ and $x \in [a, b]$ is replaced with $y \to a$.*

   The similar reasoning would show that this requirement is satisfied.

We have shown the correctness of the `clear` operator and analyze the number of label changing in the previous page, which showed that $\kappa(v) = vs$ for function $f : \{0, 1\}^n \to \mathcal{R} = \{0, ..., 2^s - 1\}$.

3 Open Problem

A recent work shows that testing $f : \{0, 1\}^n \to \mathcal{R}$ for monotonicity requires $\Omega(\min(n, |\mathcal{R}|^2)$ queries [1]. It is an open question whether the general upper bound of $O(\frac{n}{\epsilon} \log |\mathcal{R}|)$ can be improved such that it is independent of $|\mathcal{R}|$ (or at least has a better dependence on $|\mathcal{R}|$). Since one can assume that $|\mathcal{R}| \leq 2^n$, any upper bound of $o(n^2/\epsilon)$ would be an improvement.

References

[1] Eric Blais, Joshua Brody and Kevin Matulef. *Property Testing Lower Bounds via Communication Complexity*, CCC 2011.

[2] Yevgeniy Dodis, Oded Goldreich, Eric Lehman, Sofya Raskhodnikova, Dana Ron, and Alex Samorodnitsky. *Improved Testing Algorithms for Monotonicity*, RANDOM-APPROX 1999.