Sublinear Algorithms	March 20, 2012
Lecture 16	
Lecturer: Sofya Raskhodnikova	*Scribe(s): Fang Song*

Last time we did monotonicity test for functions on a hypercube. In this lecture, we extend the ideas and techniques to give efficient testers of monotonicity for a couple of other classes of functions [1].

1 Testing Monotonicity of Functions on Hyper-grids

Consider a function $f : \{1, \ldots, n\}^d \to R$, where the domain is a hyper-grid with alphabet size n. As usual we call it monotone if increasing one coordinate of x doesn't decrease $f(x)$. We omit a formal definition.

Example 1. *We give a couple of simple examples.*

- $d = 1$ *(See Fig. 1 below): we can arrange the elements in domain* $[n]$ *on a line, and the arrows only go to the right indicating that the function values are non-decreasing. Note that testing monotonicity in this case corresponds to testing if a list of numbers is sorted.*

Figure 1: A monotone function f on a line with alphabet size n.

- $d = 2$ *(See Fig. 2 below): we can arrange the elements in* $[n]^2$ *as a* $n \times n$ *grid, and the arrows only go upward or rightward indicating that the function values are non-decreasing. Similarly, in this case, monotonicity testing boils down to testing if rows and columns of a matrix is sorted.*

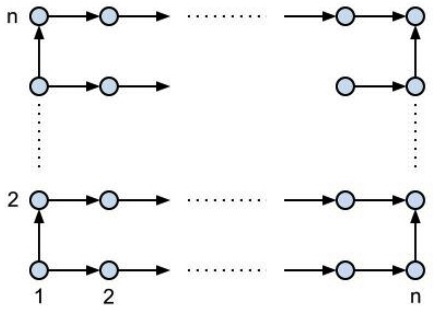

Figure 2: A monotone function f on a 2-d grid with alphabet size n.

- *Functions on a hypercube* $\{0, 1\}^d$ *can be seen as a special case with alphabet size* $n = 2$.

The main theorem we are going to prove is the following:

Theorem 1. *Testing monotonicity of* $f : [n]^d \to R$ *can be done with* $\mathcal{O}(\frac{d}{\varepsilon} \cdot \log n \cdot \log |R|)$ *queries.*

1.1 Proof of Main Theorem

The proof of the main theorem will follow in two steps. We first analyze the case of Boolean functions (i.e., $R = \{0, 1\}$) and then we apply the *range reduction* technique to obtain the claim for general range R. Our focus here will be the first step, and the second step is straightforward from last lecture and is left as an exercise. We restate Theorem 1 in the Boolean case.

Proposition 2. *Testing monotonicity of $f : [n]^d \to \{0, 1\}$ can be done with $\mathcal{O}(\frac{d}{\varepsilon} \cdot \log n)$ queries.*

Proof. From prior lecture (Lecture 2: Testing sortedness of a list.), we know how to test monotonicity of boolean functions on a line, namely the special case where $d = 1$. Here we will essentially show a reduction of higher dimension $d > 1$ case to the $d = 1$ case. Note that for a hyper-grid $[n]^d$, we can imagine n orthogonal axes. Lines in certain dimension i just mean all the lines parallel to axis i. It is easy to calculate that there are n^{d-1} lines in each dimension, and hence dn^{d-1} in total. Our tester operates like below:

Tester $\mathcal{T}$ for function $f : [n]^d \to \{0, 1\}$

1. Pick a line uniformly at random.
2. Run a monotonicity tester for the line:
   - Pick an edge from a 2-TC-spanner for the line and reject if the edge is violated.

Remark Note that for step 2 above, we have a $\mathcal{O}(\frac{1}{\varepsilon})$ tester in the Boolean case. However, it is not known how to perform a range reduction for this more efficient test. So we work with the test for which we can do range reduction.

For each line, we consider the sparest 2-TC-spanner (with $\mathcal{O}(n \log n)$ edges). We call these edges *special*. In order to analyze the query complexity of the tester, one crucial question we need to answer is:

How many special edges are violated if the function is ε-far from monotone?

We know how to give a lower bound on the number of violated edges in the case of a line (i.e. $d = 1$). Here we adopt the same idea as to show a "contrapositive" statement, namely we only need to make few changes to *repair* a function if it is close to monotone.

Repair Operation for function $f : [n]^d \to \{0, 1\}$

1. For $i = 1, \ldots, d$ \\Repair each dimension one by one
   - $Sort(i)$: sort values on all lines in that dimension.

To see why this fixes the monotonicity, it suffices to argue that sorting dimension i is "independent" of sorting dimension j. Namely, we show

Claim 3. *$Sort(i)$ does not increase number of violated special edeges along dimension j.*

Proof. Consider two (horizontal) lines h_1 and h_2 along dimension i, and take two (vertical) lines v_1 and v_2 along the j-dimension (see Fig. 3). From prior lecture, we know that fixing the four intersecting points along dimension i doesn't introduce more violations along lines v_1 and v_2. This fact suffices to imply the claim here, because we can break the sorting operation into a sequence of pairwise *swap* operations, e.g., by using *Bubble Sort*. □

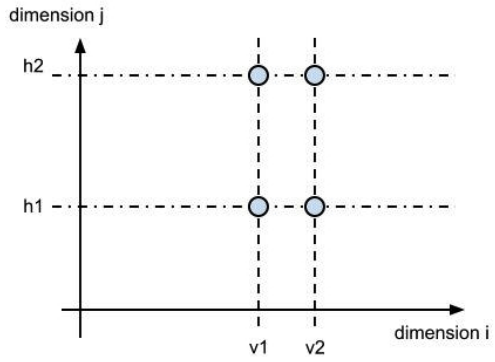

Figure 3: Correctness of repair operation: dimension i doesn't affect dimension j

Let V_f be the total number of violated special edges, then the number of changes that are necessary to repair the function follows from earlier argument (Lecture 2, Lemma 5), which is $\leq 2 \cdot V_f$. This immediately implies that if f is ε-far from monotone, i.e., at least εn^d changes are necessary to make it monotone, then there are at least $\frac{1}{2}\varepsilon n^d$ violated special edges*. Therefore, going back to step 2 of tester $\mathcal{T}$, we know that the probability of sampling a violated edge will be at least $\frac{\frac{1}{2}\varepsilon n^d}{dn^{d-1} \cdot n \log n} = \Omega(\frac{\varepsilon}{d \log n})^\dagger$. Then *witness lemma* tells us that $\mathcal{T}$ will reject an f ε-far from monotone with probability at least $2/3$ within $\mathcal{O}(\frac{d \log n}{\varepsilon})$ queries.

Finally, we conclude that with query complexity $\mathcal{O}(\frac{d}{\varepsilon} \cdot \log n)$, our tester $\mathcal{T}$ accepts a function $f : [n]^d \to \{0, 1\}$ if f is monotone; and rejects f with probability at least $2/3$ if f is ε-far from monotone. □

2 Testing Monotonicity on Arbitrary Poset Domains

In this setting, we study functions whose domain is the verticies in a *directed acyclic graph* (DAG) G [2]. We say a function $f : V(G) \to R$ is monotone $f(u) \leq f(v)$ for all edges $(u, v) \in G$.
[Discussion in class: is it possible to embed DAGs into hyper-grids so everything reduces to the previous setting? Answer: it is possible to embed, but it is not clear how to get a good tester out of it. The reason is that in general, a hypergrid of d dimensions might be needed to embed d nodes. As a result, if we use the previous tester directly, the query complexity will be linear.]

The motivation of studying such functions comes from the observation that any property over $\{0, 1\}$ alphabet can be described as a k-CNF formula, for some k, on n variables. Then we want to understand how many queries are needed to test a property based on the complexity of the formulas that describe the property. Interestingly, the special case of $k = 2$, i.e., testing 2-CNF properties, is known to be equivalent to testing monotonicity on general DAGs.

There is a straightforward $\mathcal{O}(\sqrt{n})$ upper bound on the query complexity of testing monotonicity on these functions, but so far we only know a very loose lower bound $\Omega(n^{\frac{1}{\log \log n}})$ for *non-adaptive* testers. Details to come in next lecture.

References

[1] Yevgeniy Dodis, Oded Goldreich, Eric Lehman, Sofya Raskhodnikova and Dana Ron and Alex Samorodnitsky *Improved Testing Algorithms for Monotonicity.* RANDOM: 97-108 (1999)

*Because otherwise, we can fix f with less than εn^d changes, contradiction the definition of being ε-far from monotone

†$n \log n$ comes from the total number of edges of the (sparsest) 2-TC-spanner on a line L_n, and there are in total dn^{d-1} lines in the hyper-grid.

[2] Eldar Fischer, Eric Lehman, Ilan Newman, Sofya Raskhodnikova, Ronitt Rubinfeld and Alex Samorodnitsky *Monotonicity Testing Over General Poset Domains.* Proceedings of the 34th ACM STOC, 474–483, 2002.