6.889 Sublinear Time Algorithms	March 13, 2019
Lecture 11	
Lecturer: Ronitt Rubinfeld	*Scribe: Lillian Zhang*

1 Outline

The following topics were covered in class:

- Monotonicity testing of $f : [n] \to \mathbb{Z}$ in Hamming distance
- Monotonicity testing of $f : [n] \to [0, 1]$ in L_1 distance

2 Monotonicity testing in Hamming distance

Problem: Given a list $y_1, \cdots, y_n$ of integers, our goal is to output whether or not the list is sorted:

- If the list is sorted, that is $y_1 \leq y_2 \leq \cdots \leq y_n$, then output PASS.
- If the list is ϵ-far from sorted in Hamming distance, that is $> \epsilon n$ elements need to be changed to make the list sorted, then output FAIL with probability $\geq 3/4$.

It may be easier to consider counting the number of elements that need to be deleted to make the list sorted rather than the number of elements that need to be replaced, though they are equivalent because for every element that we would delete, we can instead change its value to that of the closest correctly sorted element, and finish with a sorted list.

An easy case of this problem is when $y_i \in \{0, 1\}$ $\forall i$, which can be tested in poly$(1/\epsilon)$ time.

2.1 (Linear) proposed algorithms

We propose two and analyze 'first instinct' algorithms, using some of the intuition they give to propose a faster algorithm.

2.1.1 A first attempt

A naive way to test for monotonicity would be to check if all neighbors are consistently nondecreasing, but we will give a 'bad input' example that shows that such a tester cannot be sublinear.

Algorithm (neighbour test): Pick random i from $[n]$ and test if $y_i \leq y_{i+1}$.

Bad input: $1, 1, \cdots, 1, 1, 0, 0, \cdots, 0, 0$.

This list is $n/2$-far from monotone, but only one choice of i fails the neighbour test, so we need linear queries to find the i using this algorithm.

2.1.2 A second attempt

The most straightforward way to perform property testing is to query random pairs, but we will again give a 'bad input' example that shows that such a tester cannot be sublinear.

Algorithm (random pair test): Pick random i, j from $[n]$ such that $i < j$ and test if $y_i \leq y_j$.

Bad input: $n/4$ groups of 4 decreasing elements: $4, 3, 2, 1,\ 8, 7, 6, 5,\ 12, 11, 10, 9,\ \cdots$.

The longest monotone sequence has size $n/4$, so this list is $3n/4$-far from monotone. However, we only realize that it is not monotone when we pick two elements from the same group. The probability of this is $\frac{6 \cdot n/4}{n(n-1)/2} = \frac{3}{n-1}$, so any sublinear number of samples gives us a less than constant chance of success using this algorithm.

2.2 An $O(\log n\ /\epsilon)$ query algorithm

We will give an $O(\log n\ /\epsilon)$ query algorithm for this problem, assuming that all elements in the list are distinct. We will later show that distinctness is not necessary.

Some intuition for the $\log n$ factor is that pairs of every distance need to be checked: Our first bad input shows the difficulties of an input whose non-monotone pairs are far apart, while our second bad input shows the difficulties of an input whose non-monotone pairs are nearby. A naive, though sublinear, algorithm would be to check pairs for every distance 2^i for all integer $i < \log n$. The following algorithm, involving binary search, is more elegant.

Algorithm 1: Hamming Distance Monotonicity Tester

Input : List $y_1, y_2, \cdots, y_n$, and value ϵ
Output: pass or fail

1. **for** $O(1/\epsilon)$ *random indices* i *in* $[1, n]$ **do**
2. Set $z \leftarrow y_i$;
3. Do binary search on $y_1, y_2, \cdots, y_n$ for z;
4. **if** *see inconsistency (left is bigger, right is smaller, or* z *not found)* **then**
5. Fail and halt
6. Pass

2.2.1 Correctness

If the list is sorted, then binary search will always succeed, by distinctness, so it will always be passed.

If the list is ϵ-far from sorted, we want to show that the probability of failure is $\geq 1/4$. To do so, we will show the contrapositive, that when the probability of failure is $\leq 1/4$ then the list is ϵ-close to monotone.

Definition 1 *The index i is called 'good' if binary search for x_i is successful.*

A restatement of the test: Pick $O(1/\epsilon)$ i's randomly and pass if all are good.

So if the test is likely to pass, then $\geq 1 - \epsilon$ fraction of i's are good. (Otherwise, if $< 1 - \epsilon$ fraction of i's are good and we sample $O(1/\epsilon)$, the probability of only sampling good i's is $< (1 - \epsilon)^{O(1/\epsilon)} < e^{-O(1)}$, very low).

The main observation: Values at 'good' indices form an increasing subsequence.

Proof of observation: Consider $i < j$ both good. Let k be their least common ancestor in the binary search tree. Then at x_k, a search for x_i goes left while a search for x_j goes right. Thus $x_i < x_k < x_j$.

We have proved that if the test is likely to pass, then $\geq 1 - \epsilon$ fraction of i's are good, and these indices form an increasing subsequence, so the sequence is ϵ-close to monotone.

2.2.2 Runtime

The runtime is $O(\frac{\log n}{\epsilon})$, since there are $O(1/\epsilon)$ rounds and each round takes $\log n$ queries due to the binary search.

2.2.3 Distinctness is unnecessary

Using an old trick from parallel computation, we can create distinctness. If a list is not distinct, we can create distinctness 'virtually' (at runtime) by appending the index i to each element x_i. That is, the list is (virtually) transformed as follows:

$$x_1, x_2, \cdots, x_n \to (x_1, 1), (x_2, 2), \cdots, (x_n, n).$$

Now, all the elements are distinct, and further the ties were broken without a change in order. (If $x_i \leq x_{i+1}$ then $(x_i, i) < (x_{i+1,i+1})$). With this trick, the above algorithm can be applied to arbitrary lists.

3 Monotonicity testing in L_1 distance

Problem: Given a function $f : [n] \to [0, 1]$, our goal is to output whether or not the f is monotone with respect to L_1 distance:

- If f is monotone, that is $f(x_1) \leq f(x_2)$ for all $x_1 < x_2$, then output PASS.
- If the f is ϵ-far in L_1 distance from any monotone function, then output FAIL with probability $\geq 3/4$.

Recall that the L_1 *distance* of functions f and g is $||f - g||_1 = \sum_i |f(i) - g(i)|$. Let's see how it compares to Hamming distance, the distance metric used in the previous section, under different ranges.

L_1 and Hamming distance comparison:

- when $f : [n] \to \{0, 1\}$, Hamming Distance = L_1 distance
- when $f : [n] \to [0, 1]$, Hamming Distance $\geq L_1$ distance
- when $f : [n] \to [0, d]$, Hamming Distance $\times d \geq L_1$ distance

3.1 An $O(1/\epsilon)$ query algorithm

We will give an $O(1/\epsilon)$ query algorithm for testing if f is monotone with respect to L_1 distance. (Even though testing with respect to Hamming distance takes $O(\log n\ /\epsilon)$ queries, from the previous section!!!)

Idea for algorithm: Reduce to boolean function monotonicity testing, which has an $O(1/\epsilon)$ query algorithm!

Definition 2 *For $t \in [0, 1]$, 'threshold function'*

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

For example, an arbitrary function f (in black) and the threshold functions $f_{(t_1)}, f_{(t_2)}, f_{(t_3)}$ (in red, green, and blue) are depicted on the following page. Observe that for f monotone, $f_{(t)}$ is monotone for all t, and for f not monotone, there is some t for which $f_{(t)}$ not monotone.

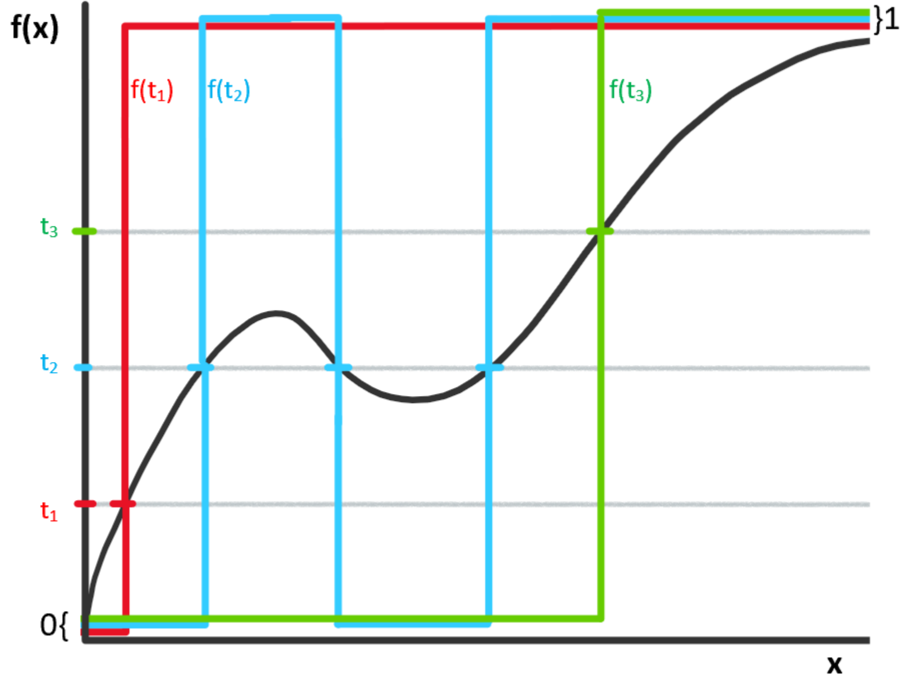

Figure 1: Function f (in black) and the threshold functions $f_{(t_1)}, f_{(t_2)}, f_{(t_3)}$ (in red, green, and blue).

f can be expressed as a sum of functions mapping to $\{0,1\}$: For *any* function f such that $f(0) = 0$,

$$f(x) = \int_0^{f(x)} 1\ dt + \int_{f(x)}^1 0\ dt = \int_0^{f(x)} f_{(t)}(x)\ dt + \int_{f(x)}^1 f_{(t)}(x)\ dt = \int_0^1 f_{(t)}(x)\ dt$$

Definition 3 *Let $L_1(f, M) \equiv L_1$ distance of f to closest monotone function.*

Main lemma: $L_1(f, M) = \int_0^1 L_1(f_{(t)}, M)\ dt$.

That is, we can express the distance to monotonicity in terms of distance of threshold functions.

Proof idea for main lemma: For f to be monotone, it must be monotone at each threshold (the associated threshold function must be monotone). Thus to change f to a monotone function, at each threshold t we need to change at least $L_1(f_{(t)}, M)$. This implies $L_1(f, M) \geq \int_0^1 L_1(f_{(t)}, M)\ dt$.

Let g_t be the closest monotone function to $f_{(t)}$ and let $g = \int_0^1 g_t\ dt$. We know g is monotone since it is the sum of monotone functions. This implies $L_1(f, M) \leq ||f - g||_1 \leq \int_0^1 L_1(f_{(t)}, M)\ dt$.

Proof of $L_1(f, M) \leq \int_0^1 L_1(f_{(t)}, M)\ dt$: For all t, let g_t be the closest monotone function to $f_{(t)}$ and let $g = \int_0^1 g_t\ dt$. (Remember that both $f_{(t)}$ and g_t are 0/1 functions, so for each x there is some t for which they have value 0 when less than that t, and value 1 when greater than that t). We know g is monotone since it is the sum of monotone functions. This implies $L_1(f, M) \leq ||f - g||_1 = ||\int_0^1 f_{(t)}\ dt - \int_0^1 g_{(t)}\ dt||_1 = ||\int_0^1 f_{(t)} - g_{(t)}\ dt||_1 \leq \int_0^1 ||f_{(t)} - g_{(t)}||_1\ dt = \int_0^1 L_1(f_{(t)}, M)\ dt$.

Proof of $L_1(f,M) \geq \int_0^1 L_1(f_{(t)},M)\ dt$: Let g be the closest monotone function to f with respect to L_1, so g_t is monotone for all $t \in [0,1]$. Further, we claim that $f(x) \geq g(x)$ if and only if $f_{(t)}(x) \geq g_{(t)}(x)$ for all $t \in [0,1]$. This implies in line (4) of the below calculation, that all terms in both sums are positive. Thus we have

$$\begin{aligned}
L_1(f,M) &= ||f-g||_1 && (1)\\
&= ||\int_0^1 f_{(t)} - g_{(t)}\ dt||_1 && (2)\\
&= \sum_{\substack{x \text{ s.t.}\\ f(x)\geq g(x)}} \int_0^1 (f_{(t)} - g_{(t)})(x)\ dt\ + \sum_{\substack{x \text{ s.t.}\\ f(x)<g(x)}} \int_0^1 (g_{(t)} - f_{(t)})(x)\ dt && (3)\\
&= \int_0^1 \sum_{\substack{x \text{ s.t.}\\ f(x)\geq g(x)}} (f_{(t)} - g_{(t)})(x) + \sum_{\substack{x \text{ s.t.}\\ f(x)<g(x)}} (g_{(t)} - f_{(t)})(x)\ dt && (4)\\
&= \int_0^1 ||f_{(t)} - g_{(t)}||_1\ dt && (5)\\
&\geq \int_0^1 L_1(f_{(t)},M)\ dt && (6)
\end{aligned}$$

Theorem: If T is a nonadaptive (it decides where to query Q, queries Q, and then makes the decision), 1-sided (monotone f always passed) error tester for $f:[n] \to \{0,1\}$ with respect to Hamming measure, then T is a nonadaptive, 1-sided error tester for $f:[n]\to[0,1]$ with respect to L_1 measure.

Proof of theorem: If f monotone, T always accepts.

If f is such that $L_!(f,M) \geq \epsilon n$, then by the lemma $L_1(f,M) = \int_0^1 L_1(f_{(t)},M)\ dt$, so there exists t^* such that $L_1(f_{(t^*)}),M) \geq \epsilon n$. Since $f_{(t^*)}$ boolean, then the Hamming distance and L_1 distance of $f_{(t^*)}$ from monotone are the same! Then, if T given $f_{(t^*)}$ fails with probability $\geq 3/4$, then with probability $\geq 3/4$ query set Q contains $x < y$ such that $f_{(t^*)}(x) = 1 > 0 = f_{(t^*)}(y) \implies f(x) \geq t^* > f(y)$, so Q will output fail.