COMS E6998-9: Algorithmic Techniques for Massive Data — Nov 10, 2015

Lecture 18 — Distribution & Monotonicity & Graph Testing

Instructor: *Alex Andoni* — Scribes: *Yuan Feng, Vishal Juneja*

1 Introduction

In today's lecture, we firstly finished proof for Uniformity Testing lemma. Then we went through other property testings, including Identity Testing, Equality Testing, Independence Testing, etc. In detail, we discussed Monotonicity Testing algorithm in sub-linear time, with the approximation note $\epsilon-$far. Then we see how to use such an approximation in Sparse Graph Testing.

2 Uniformity Testing

Remember in last lecture we prove the lemma with 2 claims:

Lemma 1. *$\frac{C}{M}$ allows us to distinguish between the two cases above, as long as $m = \Omega(\frac{\sqrt{n}}{\epsilon^4})$. M is the normalization constant $M = \binom{m}{2}$.*

We let $d = \|D\|_2^2$, and have the two claims:

Claim 2. $E\left[\frac{C}{M}\right] = d$

Claim 3. $Var\left[\frac{C}{M}\right] \leq \frac{d}{M} + \frac{8d^2\sqrt{n}}{m}$

Now let's finish the proof for the lemma:

Proof. We have $\frac{1}{n}$ as d's lower bound

$$\begin{aligned} Var\left[\frac{C}{M}\right] &\leq \frac{d}{M} + \frac{8d^2\sqrt{n}}{m} \\ &\leq \frac{d^2}{Md} + \frac{8\sqrt{n}}{m}\cdot d^2 \\ &\leq \frac{n}{M}\cdot d^2 + \frac{8\sqrt{n}}{m}\cdot d^2 \\ &\leq O(1)\cdot\left[\epsilon^8 d^2 + \epsilon^4 d^2\right] \\ &\leq O(\epsilon^4 d^2) \end{aligned}$$

By Chebyshev's, this means with probability 90%, If D is uniform distribution:

$$\begin{aligned}\frac{C}{M} &\leq d + O(\epsilon^4 d^2)^{\frac{1}{2}} \\ &\leq d + O(\epsilon^2 d) \\ &\leq \frac{1}{n} + \frac{O(\epsilon^2)}{n} \\ &< \frac{1}{n} + \frac{\epsilon^2}{2n}\end{aligned}$$

If D is not uniform distribution:

$$\begin{aligned}\frac{C}{M} &\geq d - O(\epsilon^2 d) \\ &\geq d(1 - O(\epsilon^2)) \\ &\geq \frac{1}{n} + \frac{\epsilon^2}{2n}\end{aligned}$$

This corresponds to the algorithm:

```
if $C < \frac{am^2}{n}$ then
  return uniform
else
  return nonuniform
end if
```

Note: choose 'a' as a quality constant in above alogorithm. □

3 Identity Testing

Problem:
If we have known distribution p, and given samples from another unknown distribution q, we need to distinguish between: $p = q$ vs. $\|p - q\|_1 \geq \epsilon$.

Note that uniformity Testing is just an instance of this, where $p = U_n$. Another example is suppose if we want to test is one 'q' is a binomial distribution or not.
By the classic χ^2 test [Pearson 1900], we let X_i to be the number of occurrences of i:

$$\sum_i \frac{(X_i - kp_i)^2 - kp_i}{p_i} \geq \alpha$$

Where k is the total count of samples, the same as m in Uniformity Testing. When k is very large, we can have $E[X_i] = kp_i$. So in this case only $O(n)$ samples are needed, which gives better bound, but not optimal.

The test of [Valiant, Valiant 2014] gives better bound $O(\sqrt{n})$, which is near to be optimal:

$$\sum_i \frac{(X_i - kp_i)^2 - X_i}{p_i^{\frac{2}{3}}} \geq \alpha$$

4 More properties in Distribution Testing

4.1 Equality Testing

Problem: Given samples from unknown distributions p, q, distinguish: $p = q$ vs. $\|p - q\|_1 \geq \epsilon$.
Sample bound: $\Theta_\epsilon(n^{\frac{2}{3}})$

4.2 Independence Testing

Problem: Given samples from $(p, q) \in [n] \times [n]$, distinguish: p is independent of q vs. $\|(p-q) - A \times B\|_1 \geq \epsilon$ for any distribution A, B on $[n]$.
Sample bound: $\widetilde{\Theta_\epsilon}(n^{\frac{4}{3}})$

Since the input space is $O(n^2)$ so the above algorithm is still sub-linear.

4.3 More

There are more properties testing. Some of them use linear programming.

5 Monotonicity Testing

This is a little bit different set-up:
Problem: Given a sequence $x_1, ...x_n$, distinguish between: the sequence is sorted vs. the sequence is NOT sorted.

We want this to be done in sub linear time $o(n)$, which is impossible because there can be just one inversion somewhere. Hence we need some notion of approximation. Here we introduce the concept of $\epsilon-$far.
$\boldsymbol{\epsilon - far}$ in Monotonicity Testing is defined as: two sequences are $\epsilon-$far if we need to delete ϵ fraction of elements from one to convert it into a sorted sequence.

5.1 Testing

Randomly sample positions i, then check: $x_i < x_j$ iff. $i < j$.
Analysis: But consider the bad case, where we have the sequence like $2, 1, 4, 3, ...i, i-1, i+2, i+1, ...n, n-1$, we need at least $\Omega(\sqrt{n})$ times to see consecutive integers. Factor of $\sqrt{n}$ comes from the birthday problem discussed in the last lecture.

Fix: Let's check if the sequence is sorted locally, i.e. to sample adjacent pairs.
Analysis: Still we can find the bad case where the sequence is like $[\sqrt{n}+1, \sqrt{n}+2, ...2\sqrt{n},][1, 2, ..., n]......[n\sqrt{n},$

$n\sqrt{n}+1, ...n,][n-2\sqrt{n}, n-2\sqrt{n}+1, ...n-\sqrt{n}]$, with at most $o(\sqrt{n})$ times we can easily sample at the border of each consecutive sub-sequences or sample two consecutive blocks.

5.2 Algorithm

For simplicity we assume $x_i \neq x_j$ i.e. strict monotonicity and do Binary Search. For each iteration starting with $[s,t]=[1,n]$, take random i to find x_i: take middle $m=\frac{s+t}{2}$,

- if $x_i < x_m$, we recurse on the left
- if $x_i > x_m$, we recurse on the right

We only need to repeat $\frac{3}{\epsilon}$ iterations to distinguish if the sequence is sorted or not with sufficient guaranty. The intuition is to identify cases of violation in an iteration. Note that in this process, violation occurs or sequence fails if:

1. some x_i is not where it should be
2. some $x_m \notin [x_s, x_t]$

And if there is no failure in any repetition, we will argue that the sequence is sorted.

5.3 Analysis

If the sequence pass the algorithm, it is sorted. We will argue it to be $\epsilon-$far sorted via contrapositive.

Lemma 4. *If one iteration succeeds with probability $\geq \epsilon$, then sequence $\leq \epsilon$ far from a sorted sequence. So we need only $\frac{3}{\epsilon}$ iterations to catch the case when sequence is $> \epsilon$ sorted with probability at least 90%.*

Proof. We call $i \in [n]$ good if it passes the test

Claim 5. *if $i < j$ are good, then $x_i < x_j$*

Consider the lowest common ancestor of two nodes in BST such that it is between x_i and x_j. Hence good i's are sorted.
With this claim, if we have probability of good at least $1-\epsilon$, then it definitely means that at least $(1-\epsilon)n$ elements in the sequence are good.

□

5.4 Discussions

1) Duplications
Now we need to deal with the assumption made at first, i.e. for the case $x_i = x_j$:
solution: Just replace all x_i with (x_i, i), then we have if $i=j$, $(x_i, i) < (x_j, j)$ for $i<j$. In this way, the sequence must be strictly monotonic.
2) Adaptivity
In binary search, where we query next step depends on what we learned from the previous query. But we don't want such adaptivity.
solution: Because in each iteration we query for x_i, we know precisely where the Binary Search is supposed to look at, we can generate the precise positions to query at beginning of the query. Then, we can query them all at the same time. We will detect any inconsistency from the queries' positions.

5.5 Complexity

$O(\log n)$ for Binary Search is tight.
In more general case, we consider function $f : \{0,1\}_d \to \{0,1\}$, we can test if the function is monotone, i.e. $f(x) \le f(y)$ whenever $x \le y$, in $O_\epsilon(\sqrt{d})$ queries.

6 Graph Testing

Graph $G = (V, E)$ have n vertices and m edges. For dense graph, where almost every possible edge exists, $m = \Theta(n^2)$. And for sparse graph, Degree $d \le O(1)$.
How to represent the graph is different in each case. If the graoh is dense, we can just query each (i, j) edge. But for sparse graph, most such query returns null. So, we need to prepresent the graph by each node in this case.

6.1 Property Testing: Connectivity

We want to check connectivity for sparse graphs, where we have the $\boldsymbol{\epsilon - far}$ be defined as: we need to delete or insert $\ge \epsilon n$ edges to make the graph connected, where $m = dn = O(n)$.

This will only make sense when $\epsilon d << 1$.

Proof. If $\epsilon d > 1$,
$\forall G$ with $deg \le d$ is $\epsilon d-$close to a completed graph.
If we add a spanning tree, we will have $n - 1 \le \epsilon m = \epsilon dn$ □

6.2 Algorithm

For each iteration: we randomly select a node s, and do BFS from s, until we see more than $\frac{4}{\epsilon d}$ node in the connected components (CC). If the CC's size is smaller than that, report "disconnected."
We repeat for $r = O(\frac{1}{\epsilon d})$ times then report "connected".

6.3 Analysis

Claim 6. *If the graph is $\epsilon-$far connected, it has at most $\Omega(\epsilon dn)$ connected components.*

Proof. Suppose G has c connected components, it will connect using $O(c)$ modifications, by just connecting each connected component consecutively.
Special case: a CC can have all its nodes with full degree. In this case, we can delete one edge within the CC and add edge between the CC and another. So for the worst case when every CC is of full degree, we still need $O(c) = 2c$ modifications. □

So, on average a CC has $O(\frac{n}{\epsilon dn}) = O(\frac{1}{\epsilon d})$ nodes. We can pick one of them with probability at least ϵd.