COMS E6998-9 F15

Lecture 18:
Uniformity Testing
Monotonicity Testing

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Administrivia, Plan

- Admin:
  - PS3: pick up
  - Project proposals: Nov 16th

- Plan:
  - Uniformity Testing
  - Monotonicity Testing

- Scriber?

Uniformity testing

- Enough to distinguish:
  - $||D||_2^2 = 1/n$ (unif)
  - $||D||_2^2 > 1/n + \epsilon^2/n$ (non-unif)
- Lemma: $\frac{1}{M} \cdot C$ is a good enough as long as
  - $m = \Omega\left(\frac{\sqrt{n}}{\epsilon^4}\right)$
  - where $M = m(m-1)/2$
- Let $d = ||D||_2^2$
- Claim 1: $E\left[\frac{C}{M}\right] = d$
- Claim 2: $Var\left[\frac{C}{M}\right] \leq \frac{d}{M} + \frac{8d^2\sqrt{n}}{m}$
- Finish lemma proof…

Algorithm UNIFORM:

```
Input: n, m, x_1, … x_m
  C = 0;
  for(i=0; i<m; i++)
    for(j=i+1; j<m; j++)
      if (x_i = x_j)
        C++;

  if (C < a · m^2/n)
    return “Uniform”;
  else
    return “Not uniform”;
// a: constant dependent on ε
```

Identity Testing

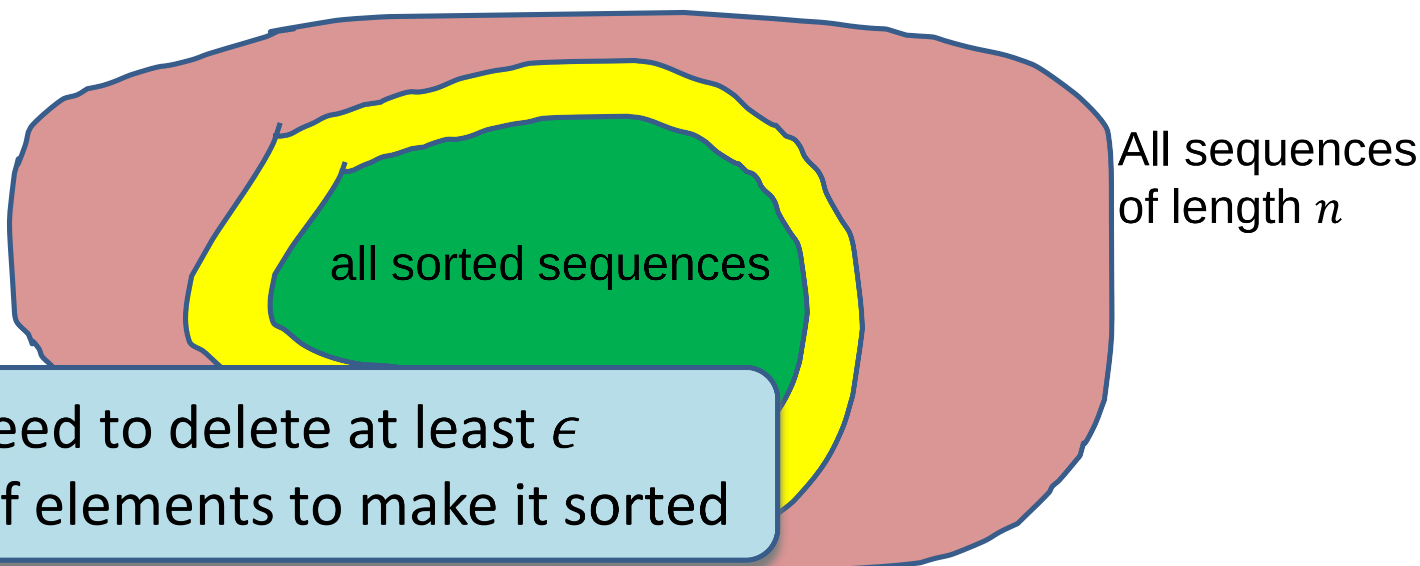
- Problem:
  - We have known distribution p
  - Given samples from q, distinguish between:
    - $p = q$ vs $||p - q||_1 \geq \epsilon$
    - Uniformity is an instance ($p = U_n$)
- Classic χ^2 test [Pearson 1900]:
  - Let $X_i = \#$ of occurrences of i
  - $\sum_i \frac{(X_i - kp_i)^2 - kp_i}{p_i} \geq \alpha$
- Test of [Valiant, Valiant 2014]:
  - $\sum_i \frac{(X_i - kp_i)^2 - X_i}{p_i^{2/3}} \geq \alpha$

Distribution Testing++

- Other properties?
- Equality testing:
  - Given samples from unknown p, q, distinguish
  - $p = q$ vs $||p - q||_1 \geq \epsilon$
  - Sample bound: $\Theta_\epsilon(n^{2/3})$
- Independence testing:
  - 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 distributions A, B on $[n]$
  - Sample bound: $\widetilde{\Theta_\epsilon}(n)$
- Many more…

Testing Monotonicity

- Problem: given a sequence $x_1, \ldots x_n$, distinguish:
  - sequence is sorted, vs
  - sequence is NOT sorted
- In $o(n)$ time?
- Hard exactly: can have just one inversion somewhere
- Approximation notion: ϵ-far



ϵ-far: if need to delete at least ϵ fraction of elements to make it sorted

Testing Monotonicity

ϵ-far: if need to delete at least ϵ fraction of elements to make it sorted

- A testing algorithm:
  - Sample random positions i
  - Check that $x_i \leq x_j$ iff $i < j$
- How many samples?
  - Bad case: 2,1,4,3,…,i,i-1,i+2,i+1,…,n,n-1
  - At least $\Omega(\sqrt{n})$ before we see an adjacent pair
- Fix?
  - Can sample adjacent pairs!
- Works?
  - Bad case too

Algorithm: Monotonicity

- Assumption:
  - $x_i \neq x_j$
- One iteration:
  - Pick a random i
  - Do binary search on $x = x_i$ in the sequence
    - Start with interval [s,t]=[1,n]
    - For interval [s,t], find middle $m = \frac{s+t}{2}$
    - If $x < x_m$, recurse on the left
    - If $x > x_m$, recurse on the right
- Fail if find inconsistency:
  - 1) x_i not found where it should be
  - 2) $x_m \notin [x_s, x_t]$

Algorithm MONOTONICITY:

Input: $n, x_1, \ldots x_n$
for(r=0; $r < 3/\epsilon$; r++)
  Let $x = x_i$
  perform binary search on x
  if (x not found at position i
    OR binary search inconsistent)
   return "not sorted";

If finished ok, return "sorted".

Analysis: Monotonicity

- If sorted, will pass the test
- If ϵ-far from sorted…
  - How do we argue?
  - Via contrapositive
- Lemma: suppose one iteration succeeds with probability $\geq 1-\epsilon$
  - Then, sequence $\leq \epsilon$ far from a sorted sequence
- Hence, $3/\epsilon$ repetitions are enough to catch the case of $> \epsilon$ far from sortedness with probability 90%:
  - Prob to report "sorted" when it's far: is at most $(1-\epsilon)^{3/\epsilon} \leq e^{-3} \leq 0.1$

Algorithm MONOTONICITY:

Input: $n, x_1, \ldots x_n$
for(r=0; $r < 3/\epsilon$; r++)
  Let $x = x_i$
  perform binary search on x
  if (x not found at position i
    OR binary search inconsistent)
   return "not sorted";

If finished ok, return "sorted".

Analysis: Monotonicity

- Lemma: suppose one iteration succeeds with probability $\geq 1 - \epsilon$
  - Then, sequence $\leq \epsilon$ far from a sorted sequence
- Proof:
  - Call $i \in [n]$ good if it passes the test
  - Claim: if $i < j$ are good, then $x_i < x_j$
    - Consider the binary search tree, and their *lowest common ancestor* x_m
    - It must be:
      - $x_i < x_m$ and
      - $x_m < x_j$
  - Hence: good i's are sorted!
  - "probability $\geq 1 - \epsilon$" $\Rightarrow$ number of good elements is at least $(1 - \epsilon)n$
  - End of proof!

Algorithm MONOTONICITY:

Input: $n, x_1, \ldots x_n$
for(r=0; $r < 3/\epsilon$; r++)
  Let $x = x_i$
  perform binary search on x
  if (x not found at position i
    OR binary search inconsistent)
   return "not sorted";

If finished ok, return "sorted".

Monotonicity: discussion

- Assumption?
  - Replace all x_i by (x_i, i)
  - Then sequence must be strictly monotonic
- Test is **adaptive:**
  - Where we query depends on what we learned from the previous queries
- Do we need adaptivity?
  - No!
  - A each iteration, we query for $x = x_i$
  - We know precisely where binary search is supposed to look at!
    - E.g., if $i = 1$, then it's positions: $\frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \ldots$
  - Can generate all the positions to query at the beginning and query them all at the same time
  - Unless, binary search is inconsistent, in which case we detect this from the queries positions

Algorithm MONOTONICITY:

Input: $n, x_1, \ldots x_n$
for(r=0; $r < 3/\epsilon$; r++)
  Let $x = x_i$
  perform binary search on x
  if (x not found at position i
    OR binary search inconsistent)
   return "not sorted";

If finished ok, return "sorted".

Monotonicity++

- $O(\log n)$ queries tight?
  - Yes
- Can consider the more general case:
  - Function $f: \{0,1\}^d \to \{0,1\}$
  - Monotone: if $f(x) \leq f(y)$ whenever $x \leq y$ (coordinate-wise)
  - Can test in $\tilde{O}_\epsilon(\sqrt{d})$ queries! [GGLRS'98, KMS'15]

Testing graphs

- We have a graph $G = (V, E)$
  - n vertices
  - m edges
- Dense case:
  - $m = \Theta(n^2)$
- Sparse case:
  - Degree $d \leq O(1)$
- Property testing:
  - Eg, is graph G connected?
  - Approximation?
    - ϵ-far: if we need to delete/insert $\geq \epsilon m$ edges

Connectivity in sparse graph

- Approximation?
  - ϵ-far: if we need to delete/insert $\geq \epsilon m$ edges
  - $m = dn = O(n)$
- When does it make sense?
  - $\epsilon d \ll 1$ (otherwise any sparse graph is close to being connected!)
- Assume: $\epsilon d \ll 1$
- Algorithm:
  - For $r = O\left(\frac{1}{\epsilon d}\right)$ times repeat:
    - Choose a random node s
    - Run a BSF from s
    - Until see more than $4/\epsilon d$ node in the CC
    - If the CC is smaller, then report "disconnected"
  - Otherwise, report "connected"

Analysis

- Claim: if ϵ-far, then graph has at most $\Omega(\epsilon d n)$ connected components
- Proof:
  - Suppose G has c connected component
  - Will connect, using $O(c)$ modifications
  - Idea:
    - Just connect each connected component consecutively
    - Issue: can get higher degree than d in a CC
      - Is really an issue when all nodes in a CC have full degree
      - Just delete one edge (preserving connectivity)
- Hence, on average a CC has $O\left(\frac{n}{\epsilon d n}\right) = O\left(\frac{1}{\epsilon d}\right)$ nodes
  - Will pick one of them with probability at least ϵd