PROPERTY TESTING

Arnab BHATTACHARYYA (in lieu of Seth)
29/08/2019
Lecture Outline

- What is property testing?
- Identify what goes into showing correctness of a testing algorithm. Some examples.
- Identify what goes into showing impossibility of fast testing. Some examples.
A motivating example

- DNA: strings in 4 characters \{A, C, T, G\}

- **Problem**: Given two DNA strands \(X\) and \(Y\), are they from the same species or from different?
If $X$ and $Y$ are from the same species, then we expect the strings are similar. Otherwise, not.

But similar in what sense?
If $X$ and $Y$ are from the same species, then we expect the strings are similar. Otherwise, not.

But similar in what sense? **Need a metric.**

- One possibility is *Levenshtein distance* (# of insertions, deletions or substitutions to turn one string into another)
Want an algorithm that outputs:

- **SAME** if $d_L(X, Y)$ is “small”
- **DIFFERENT** if $d_L(X, Y)$ is “large”
For exactly computing $d_L$, only $O(n^2)$ algorithms are known. Too expensive for bio applications.
For exactly computing $d_L$, only $O(n^2)$ algorithms are known. Too expensive for bio applications.

Is there a more efficient algorithm that outputs

- **SAME** if $d_L(X, Y) \leq T_1$
- **DIFFERENT** if $d_L(X, Y) \geq T_2$?
Is there a more efficient algorithm that outputs
- SAME if $d_L(X, Y) \leq T_1$
- DIFFERENT if $d_L(X, Y) \geq T_2$?

Indeed, there is! If $T_1$ and $T_2$ are sufficiently apart, you only need to look at $\ll n$ characters in the strings to make the correct decision with high probability!
Bad inputs are $\epsilon$-far from good, which means:

For a distance function $d$: Inputs $\rightarrow [0,1]$, for any good $X$ and bad $Y$,

$$d(X,Y) > \epsilon.$$
Definition. An algorithm is a **tester for a property** $\mathcal{P}$ if:

- The inputs are: integer $n > 0$, real $\epsilon \in (0, 1)$, and query access to an object $x$ of size $n$.
- It accepts with probability $\geq 2/3$ if $x \in \mathcal{P}$.
- It rejects with probability $\geq 2/3$ if $x$ is $\epsilon$-far from $\mathcal{P}$.
Query complexity: The number of query accesses made by the tester.

Main focus of this course will be understanding the query complexity for various properties $\mathcal{P}$. 
Data representation decides what is revealed by each query.

For example, can represent graph as an adjacency matrix or list.
**Distance function** decides what is meant by $\epsilon$-far.

The default choice is the **Hamming distance**. For two functions $f, g: [n] \to \mathbb{R}$,

$$d_H(f, g) = \frac{|\{i \in [n]: f(i) \neq g(i)\}|}{n}.$$
Often, our testers will be one-sided, meaning the tester will accept with probability 1 if $x \in P$. 
- Inputs are strings of length $n$. Property $\mathcal{P}$ is satisfied only by the all-1's string. Distance measure is the Hamming distance, $d_H$.

- Want tester to accept $x$ with probability $\geq 2/3$ if $x = 1^n$. Want tester to reject $x$ with probability $\geq 2/3$ if $\#\{i : x_i \neq 1\} > \epsilon n$.

- Tester: Sample $2/\epsilon$ random locations $i \in [n]$. Accept iff for all such $i$, $x_i = 1$.

- One-sided error. If $x$ is $\epsilon$-far from $\mathcal{P}$,
  \[
  \Pr[\text{tester rejects}] \geq 1 - (1 - \epsilon)^{2/\epsilon} \geq 2/3
  \]
To show that an algorithm $\mathcal{A}$ is a tester for a property $\mathcal{P}$ with query complexity $q(\epsilon, n)$, you need to do three things:

1. Prove that for any $x \in \mathcal{P}$, $\mathcal{A}$ accepts with probability $\geq 2/3$ (or 1 for one-sided)
2. Prove that for any $x$ that is $\epsilon$-far from $\mathcal{P}$, $\mathcal{A}$ rejects with probability $\geq 2/3$
3. Prove that the number of queries is at most $q(\epsilon, n)$ for all inputs
\( \mathcal{P} = \text{monotonicity} \)

- Input: array of \( n \) distinct numbers.

- Array \( A \) is **monotone** if \( A[i] < A[j] \) when \( i < j \).

- Array \( A \) is **\( \epsilon \)-far from monotone** if:
  \[
  \min_{\text{monotone } B} d_H(A, B) > \epsilon
  \]
Test1(\(\epsilon, n, A\)):
for t=1,…,q:
    choose random \(i \in [1, n-1]\) \(\frac{1}{n-1}\)
    output “NO” if \(A[i] > A[i+1]\)
output “YES”

For what choice of \(q\) is Test1 a tester for monotonicity?

\[ P_r[\text{YES}] = \left(1 - \frac{1}{n-1}\right)^q \]
\[ q = \Omega(n) \]
Test2(ε,n,A):
  for t=1,…,q:
    choose random i ∈ [1,n−1] 
    choose random j ∈ [i + 1,n] 
    output “NO” if A[i] > A[j] 
  output “YES”

For what choice of q is Test2 a tester for monotonicity?
Test3(ε, n, A):
  for t = 1, ..., 2/ε:
    choose random i ∈ [1, n]
    x ← A[i]
    output “NO” if binary search \( x \) does not end at i
  output “YES”

**Theorem:** Test3 is a one-sided tester for monotonicity with query complexity \( O((\log n)/\epsilon) \).
NO case analysis

Call a coordinate $i$ **searchable** if the binary search for $A[i]$ ends at $i$.

**Claim 1**: If $A$ is $\epsilon$-far from monotone, then the number of searchable $i$’s is at most $(1 - \epsilon)n$.

NO case done with this claim.

Why?

Suppose $A$ is $\epsilon$-far.

In each iteration, searchable $i$ is picked w.p. $\leq 1 - \epsilon$.

$\Pr[\text{Yes}] \leq (1 - \epsilon)^{2/3} \leq \frac{1}{2}$. 
Proof of Claim 1

Claim 2: The array $A$ restricted to its searchable coordinates is monotone.

Claim 1 follows from Claim 2. Why?

Suppose $A$ is $\varepsilon$-far but not searchable, i.e., $i's \geq (1 - \varepsilon) \cdot n$. 

$\Rightarrow A$ can be changed in $\leq \varepsilon n$ faults and made monotone.
Proof of Claim 2

Claim 3: If $i < j$ and both $i$ and $j$ are searchable, then $A[i] < A[j]$. 

Some notes

- Tester is **adaptive**, meaning that its queries may depend on the answers to its past queries.

- It is possible to make the tester non-adaptive.

- Test2 is a valid tester with query complexity $O(\epsilon^{-1})$ when the inputs are Boolean arrays.
Lower bounds on query complexity

Three common approaches

Yao’s Minimax Principle
Gap-Preserving Reductions
Communication Complexity
Lower bounds on query complexity

Three common approaches

Yao’s Minimax Principle

Gap-Preserving Reductions

Communication Complexity
Lower bounds for randomized testers

- Testers are randomized algorithms. You can think of a randomized algorithm as a random element of a collection of deterministic algorithms:
  \[ \mathcal{A} = \{A_1, A_2, A_3, \ldots \} \]

- Showing limitations for randomized algorithms is usually trickier than for deterministic algorithms
For any randomized tester $T$ making $q$ queries, there exists an input $x$ such that:

$$\Pr_T[T(x) \text{ is wrong}] > \frac{1}{3}$$

There exists a distribution $\mathcal{D}$ on inputs such that for any deterministic tester $T$ making $q$ queries:

$$\Pr_{x \sim \mathcal{D}} [T(x) \text{ is wrong}] > \frac{1}{3}$$
For any randomized tester $T$ making $q$ queries, there exists an input $x$ such that:

$$\Pr_T[T(x) \text{ is wrong}] > \frac{1}{3}$$

There exists a distribution $\mathcal{D}$ on inputs such that for any deterministic tester $T$ making $q$ queries:

$$\Pr_{x \sim \mathcal{D}}[T(x) \text{ is wrong}] > \frac{1}{3}$$

There exists some strategy for choosing $x$'s so that no matter how $T$ makes, $x$, player wins w.p. $> \frac{1}{3}$.

(i, j)th entry = 1 if $T_i(x_j)$ is wrong. 0 o.w.
For any randomized tester $T$ making $q$ queries, there exists an input $x$ such that:

$$\Pr_T[T(x) \text{ is wrong}] > \frac{1}{3}$$

There exists a distribution $\mathcal{D}$ on inputs such that for any deterministic tester $T$ making $q$ queries:

$$\Pr_{x \sim \mathcal{D}}[T(x) \text{ is wrong}] > \frac{1}{3}$$
It suffices to come up with a distribution of inputs that is hard on average for any low-query deterministic tester.

Yao’s Minimax Principle: $\mathcal{P}$ is a property over objects. Suppose there are two distributions $\mathcal{F}_1$ and $\mathcal{F}_2$ such that:

- $\Pr_{x \sim \mathcal{F}_1} [x \in \mathcal{P}] \geq 1 - \eta_1$
- $\Pr_{x \sim \mathcal{F}_2} [x \text{ is } \epsilon\text{-far from } \mathcal{P}] \geq 1 - \eta_2$
- For any deterministic algorithm $T$ making $q(n, \epsilon)$ queries: $\left| \Pr_{x \sim \mathcal{F}_1} [T \text{ accepts}] - \Pr_{x \in \mathcal{F}_2} [T \text{ accepts}] \right| \leq \eta_3$

If $\eta_1 + \eta_2 + \eta_3 < 1/3$, then the query complexity of testing $\mathcal{P}$ is more than $q(n, \epsilon)$. 
Example

Suppose $\mathcal{P} = \{1^n\}$. The query complexity of testing $\mathcal{P}$ is $\Omega(\epsilon^{-1})$. 

No det. algo with $q < \frac{1}{3\epsilon}$ queries distinguishes $T_1$ & $T_2$ up $\frac{1}{3}$.

The queries fall into $\leq q$ segments $\frac{1}{\epsilon} - q$ unqueried segments.
What about $\mathcal{P} = \{0^n, 1^n\}$?

$\mathcal{P} = \{z\}$ for a fixed string $z \in \{0,1\}^n$?
Example

Suppose $\mathcal{P} = \{x \in \{0,1\}^n : |x| \leq \frac{n}{2} (1 - \epsilon)\}$. The query complexity of testing $\mathcal{P}$ is $\Omega(\epsilon^{-2})$. 
Takeaways

- Property testing is about how you can uncover differences in the global structure by using local queries.

- For showing correctness of testers, you need to verify its query complexity and its performance on YES and NO input instances.

- For proving lower bounds on the query complexity via Yao’s minimax principle, you explicitly come up with a hard input distribution for deterministic testers.