Lecture 5

Lecturer: Sofya Raskhodnikova *Scribe(s): Youngtae Youn*

1 Yao's Minimax Principle

What is the best a probabilistic algorithm can do for the worst-case input? Perhaps it might be easier to show the limitations of a deterministic algorithm on the average over an adversarially chosen distribution of inputs. Andrew Yao observed these values are one and the same. (Lance Fortnow) *

1.1 Original Definition

Yao's minimax principle is a generic tool for proving lower bounds on randomized algorithms. Let P be a problem with a finite set $\mathcal{X}$ of inputs and a finite set $\mathcal{A}$ of deterministic algorithms that solve P. For $x \in \mathcal{X}$ and $A \in \mathcal{A}$, $cost(A, x)$ denotes the cost incurred by an algorithm A on input x. This cost can be any quantities related to algorithms such as running time or space complexity, but this lecture focuses on query complexity. Query complexity of a problem P, denoted $q(P)$, is the minimum number of queries that algorithms makes to solve P.

As a randomized algorithm is a probability distribution over the deterministic algorithms, we can regard it as a probability distribution $\mathcal{R}$ on $\mathcal{A}$. Let $cost(\mathcal{R}, x)$ denote the cost of $\mathcal{R}$ on input x. Then it can be defined as

$$cost(\mathcal{R}, x) = \mathbf{E}_{A \overset{r}{\leftarrow} \mathcal{R}} cost(A, x)$$

where $A \overset{r}{\leftarrow} \mathcal{R}$ means A is randomly sampled from the distribution $\mathcal{R}$. The intrinsic cost of $\mathcal{R}$ is defined to be the worst input x; that is, $\max_{x \in \mathcal{X}} cost(\mathcal{R}, x)$. The *randomized complexity* of the problem is defined as

$$\min_{\mathcal{R}} \max_{x \in \mathcal{X}} cost(\mathcal{R}, x) \tag{1}$$

which naturally captures the notion of the intrinsic cost of the best randomized algorithm.

Similarly, we can measure the complexity with respect to input distribution $\mathcal{D}$. Let $cost(A, \mathcal{D})$ denote the cost of a deterministic algorithm A, which is defined as

$$cost(A, \mathcal{D}) = \mathbf{E}_{x \overset{r}{\leftarrow} \mathcal{D}} cost(A, x).$$

Under the distribution $\mathcal{D}$, the best that any deterministic algorithm can do is $\min_{A \in \mathcal{A}} cost(A, \mathcal{D})$. The *distributional complexity* of the problem is defined as

$$\max_{\mathcal{D}} \min_{A \in \mathcal{A}} cost(A, \mathcal{D}) \tag{2}$$

which naturally captures the notion of the average cost guaranteed by finding a good deterministic algorithm.

Yao's minimax principle states that both (1) and (2) are equal,

$$\max_{\mathcal{D}} \min_{A \in \mathcal{A}} cost(A, \mathcal{D}) = \min_{\mathcal{R}} \max_{x \in \mathcal{X}} cost(\mathcal{R}, x) \tag{3}$$

which can be proved by applying von Neumann's Min-Max theorem for zero-sum games. Dropping off $\max_{\mathcal{D}}$ and $\min_{\mathcal{R}}$ from both sides respectively leads to an interesting inequality

$$\min_{A \in \mathcal{A}} cost(A, \mathcal{D}) \leq \max_{x \in \mathcal{X}} cost(\mathcal{R}, x) \tag{4}$$

*From his blog post at `http://alturl.com/i2ri7`

which naturally lower bounds randomized complexity. If one can cleverly come up with a suitable distribution $\mathcal{D}$ on the inputs for some problem P and prove that every *deterministic* algorithm for P incurs some cost C on the distribution $\mathcal{D}$, it follows from Yao's minimax principle that the randomized complexity is at least C. Observe that the power of this technique lies in the fact that one can choose any distribution $\mathcal{D}$ at her will and the lower bound is calculated on deterministic algorithm. Also, remember that the distribution is *known* to the deterministic algorithm.

1.2 Yao's Principle in Property Testing

The following two statements are equivalent.

1. For any *probabilistic* algorithm A of query complexity q there exists an input x such that
$$\Pr_{\text{coin toss of } A}[A(x) \text{ is wrong}] > 1/3.$$
2. There is a distribution $\mathcal{D}$ of the inputs such that for every *deterministic* algorithm of query complexity q,
$$\Pr_{x \xleftarrow{r} \mathcal{D}}[A(x) \text{ is wrong}] > 1/3.$$

For the proof of lower bound in property testing, we use the second statement.

2 Three Examples of Applying Yao's Principle

We review three examples where Yao's minimax principle is applied to prove lower bound of query complexity.

2.1 A Lower Bound for Testing 1^*

Theorem 1. *Consider a regular language $L = \{1^n : n \geq 1\}$. For any n, an ϵ-tester for $L \cap \{0,1\}^n$ requires $\Omega(1/\epsilon)$ queries.*

The gist of Yao's minimax principle is devising an input distribution $\mathcal{D}$ on which every deterministic algorithm with query complexity q fails. For this purpose, we define the input distribution as follows. [1]

The input to the tester is an n-bit string. A yes instance is a string of 1^n. It may be divided into $1/\epsilon$ blocks where each block has the same length ϵn. Let y_i be an n-bit string where all 1's in the i^{th} block are flipped to 0's.

$$\begin{array}{lccccc}
 & \overbrace{}^{1^{\text{st}}} & \overbrace{}^{2^{\text{nd}}} & \overbrace{}^{3^{\text{rd}}} & & \overbrace{}^{\frac{1}{\epsilon}\,^{\text{th}}} \\
1^n: & 1\cdots1 & 1\cdots1 & 1\cdots1 & \cdots & 1\cdots1 \\
y_1: & 0\cdots0 & 1\cdots1 & 1\cdots1 & \cdots & 1\cdots1 \\
y_2: & 1\cdots1 & 0\cdots0 & 1\cdots1 & \cdots & 1\cdots1 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
y_{\frac{1}{\epsilon}}: & 1\cdots1 & 1\cdots1 & 1\cdots1 & \cdots & 0\cdots0
\end{array}$$

Observe that each y_i is ϵ-far from 1^n. Then the input distribution is defined as follows.

$$\mathcal{D} = \begin{cases} 1^n & \text{with probability } 1/2, \\ y_i & \text{with } i \text{ uniformly drawn from } \{1, ..., 1/\epsilon\} \text{ with probability } \frac{1}{2(1/\epsilon)}. \end{cases}$$

Proof. Fix a deterministic tester A making q queries.

1. If A doesn't accept 1^n, it's already incorrect with probability $1/2$ which is larger than $1/3$.

2. Otherwise, A accepts the input if all the q queries return 1's. Even if all q queries probe distinct blocks, A can only look for q blocks and thus accept $1/\epsilon - q$ of y_i's which amounts to the probability of $(1/\epsilon - q)\frac{\epsilon}{2}$. If $q < \frac{1}{3\epsilon}$, the failure probability is larger than 1/3.

We have shown that $q < \frac{1}{3\epsilon}$ queries are not enough to guarantee that $\Pr_{x\sim\mathcal{D}}[A(x) \text{ is wrong}] < 1/3$. By Yao's minimax principle, it means for any randomized algorithm A of query complexity $q < \frac{1}{3\epsilon}$ there exists an input x which doesn't guarantee $\Pr_{\text{coin toss of } A}[A(x) \text{ is wrong}] < 1/3$. Hence it requires $\Omega(1/\epsilon)$ queries. □

2.2 A Lower Bound for Testing Sortedness

We covered two different $O(\frac{\log n}{\epsilon})$ testers based on spanner and binary search. For all constant $\epsilon \leq \frac{1}{2}$, it is shown that $\Omega(\log n)$ queries are required to decide whether the list is sorted or ϵ-far from sorted. [2, 3] In this lecture, we prove that a non-adaptive $\frac{1}{2}$-tester for sortedness requires $\Omega(\log n)$ queries. Without loss of generality, we assume that n is a power of 2.

Theorem 2. *An $\frac{1}{2}$-tester for sortedness requires $\Omega(\log n)$ queries.*

For the sake of proof based on Yao's minimax principle, we need an input distribution defined on lists which are $\frac{1}{2}$-far from sorted. Consider the recursive definition of $\log n$ lists $\ell_1, ..., .\ell_{\log n}$ of length n.

1. ℓ_1 is a list where the first half consists of 1's only and the second half 0's only.
2. For ℓ_{i+1} where $i \geq 1$, shrink each run of the same number in ℓ_i by half to generate a length-$n/2$ list L. Make a duplicate copy of $L' = L$, increase each element in L' by 2^{i-1}, and concatenate L and L' to yield a list ℓ_{i+1}.

For example, Figure 1 contains the desired list for $n = 16$. An input distribution $\mathcal{D}$ is defined so that each ℓ_i has equal probability of $\frac{1}{\log n}$. These lists have two useful properties.

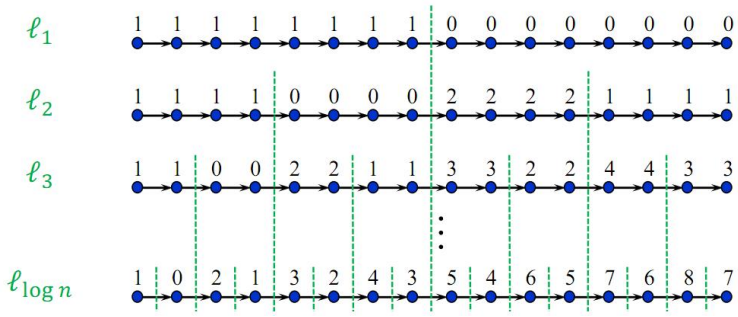


Figure 1: $\log n$ lists for proving query complexity lower bound for 1/2-tester for sortedness

- All lists are $\frac{1}{2}$-far from sorted: Each ℓ_i contains 2^i runs† of the same numbers. For $1 \leq k \leq 2^{i-1}$, replacing each $(2k)^{\text{th}}$ run with $(2k-1)^{\text{th}}$ run makes the list sorted.
- For $1 \leq i < j \leq n$, every pair (x_i, x_j) is violated in exactly one list above: Consider (x_3, x_5) in the lists. It is (1,1) in ℓ_1, (1,0) in ℓ_2, (0,2) in ℓ_3 and (2,3) in ℓ_4, but $(x_3, x_5) = (1, 0)$ in ℓ_2 is the only violated pair. This property directly follows from the construction.

Observe that '$\leq$' is a transitive relation. If $a \leq b$ and $b \leq c$, then $a \leq c$. Consider three indices $1 \leq i < j < j < k \leq n$. If a pair (x_i, x_k) is violated, it means either (x_i, x_j) or $(x_j, x_)$ is violated. If we queried q positions $(a_1, ..., a_q)$ of strictly increasing indices, we only have to check whether any of $q-1$ pairs $(x_{a_1}, x_{a_2}), (x_{a_2}, x_{a_3}), ..., (x_{a_{q-1}}, x_{a_q})$ is violated.

†A run is a sequence of more than one consecutive identical numbers.

Proof. Fix a deterministic algorithm A that probes q positions when the input is drawn from $\mathcal{D}$. These q queries yield $q-1$ pairs. The algorithm rejects the list when it sees a violated pair. As every violated pair is observed in exactly one list, q queries reject at most $q-1$ inputs. This amounts to the probability of $\frac{q-1}{\log n}$. All the input lists are $\frac{1}{2}$-far from sorted, so A must reject all of the with probability at least $2/3$. It means $\frac{q-1}{\log n} \geq \frac{2}{3}$ and $q = \Omega(\log n)$. □

2.3 A Lower Bound for Testing Monotonicity on a Hypercube

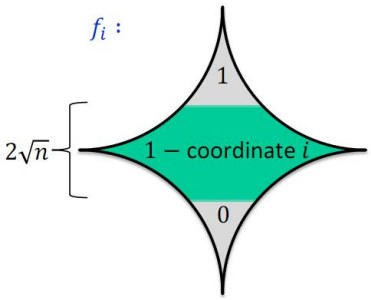
We finally review query complexity lower bound on testing monotonicity on a hypercube. [4]

Theorem 3. *Every 1-sided error non-adaptive test for monotonicity of functions $f : \{0,1\}^n \to \{0,1\}$ requires $\Omega(\sqrt{n})$ queries.*

A 1-sided error tester accepts a function f if it detects no violated pair $(f(x), f(y))$. For $x = (x_1, ..., x_n)$, we define a function f_i for $i \in \{1, ..., n\}$ as follows

$$f_i(x_1, ..., x_n) = \begin{cases} 1 & \text{if } ||x|| > n/2 + \sqrt{n} \\ 0 & \text{if } ||x|| < n/2 - \sqrt{n} \\ 1 - x_i & \text{otherwise} \end{cases}$$

where $||x||$ denotes its Hamming weight.



The lecture slide depicts the function f_i as a curved quadlateral, and the green band in the middle of width $2\sqrt{n}$ contains function values $1 - x_i$. All f_i's below this band has value 0 and all f_i's above has value 1. Consider an edge from $(x_1, ..., x_{i-1}, 0, x_{i+1}, ..., x_n)$ to $(x_1, ..., x_{i-1}, 1, x_{i+1}, ..., x_n)$. It is easy to see that this edge is violated only when both of its endpoints are in the green band. And this green band amounts to a constant fraction of vertices. Hence, all f_i's for $1 \leq i \leq n$ are ϵ-far from monotone for some constant ϵ.

The following lemma immediately implies Theorem 3.

Lemma 4. *For every non-adaptive monotonicity tester with query complexity q, there exists an index $i \in \{1, ..., n\}$ such that the tester detects a violation with probability at most $O(q/\sqrt{n})$.*

Before proving this lemma, we briefly check the connection between Lemma 4 and Theorem 3. Suppose we show that a deterministic tester A with q queries reveals a violation for at most $O(q\sqrt{n})$ of the function f_i's.

$$\sum_{i=1}^{n} \Pr[A \text{ detects a violation for } f_i] = O(q\sqrt{n})$$

Then averaging argument guarantees that there exists an f_i for which A detects a violation with probability at most $O(q/\sqrt{n})$. And q must be $\Omega(\sqrt{n})$ to guarantee the detection probability to be a constant.

Proof. For a function f_i, consider two vertices u and v in the green band that differ in the i^{th} coordinate. Without loss of generality, we assume that i^{th} coordinate of u is 0 while that of v is 1. As $f_i(u) = 1$ and $f_i(v) = 0$, it is a violated pair by the definition of f_i. So a tester detects a violated pair in f_i only when it queries such u and v in the green band.

Let Q denote the set of queried vertices of size q in the green band. We may consider Q as a graph where two vertices w and w' are connected by an edge if w' is obtained by increasing one bit from 0 to 1 in w. This graph has a spanning forest, and such a violated pair u and v must be in the same tree. And there must exist adjacent vertices on the path between u and v that differ in the i^{th} coordinate. Hence, the number of functions f_i for which the queries reveal a violation is at most

$$\text{(max number of edges in the forest)} \times \text{(max number of distance between adjacent vertices)} = (q-1) \times 2\sqrt{n}$$

which is $O(q\sqrt{n})$. □

References

[1] Noga Alon, Michael Krivelevich, Ilan Newman and Mario Szegedy. *Regular Languages are Testable with a Constant Number of Queries*, SIAM J. Comput. 30(6): 1842-1862 (2000)

[2] Eldar Fischer. *On the strength of comparisons in property testing*, Inf. Comput. 189(1): 107-116 (2004)

[3] Funda Ergün, Sampath Kannan, Ravi Kumar, Ronitt Rubinfeld and Mahesh Viswanathan. *Spot-Checkers*. J. Comput. Syst. Sci. 60(3): 717-751 (2000)

[4] Eldar Fischer, Eric Lehman, Ilan Newman, Sofya Raskhodnikova, Ronitt Rubinfeld, and Alex Samorodnitsky. *Monotonicity testing over general poset domains*. STOC 2002: 474-483