Sublinear Algorithms September 10, 2015

Lecture 5

Lecturer: Sofya Raskhodnikova *Scribe(s): Youngtae Youn, Om Thakkar*

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 $A \in \mathcal{A}$, let $cost(A, x)$ denote the cost incurred by algorithm A on input x. It can be measured in terms of any quantity related to A, such as the running time of A or its space complexity, but for this lecture, we will measure $cost(A, x)$ in terms of the query complexity of A. The query complexity of an algorithm is the maximum number of queries it makes on any input. The query complexity of a problem P, denoted $q(P)$, is the query complexity of the best algorithm that solves P.

As a randomized algorithm for a problem is a probability distribution over the set of deterministic algorithms that solve the problem, we can regard it as a probability distribution $\mathcal{R}$ on $\mathcal{A}$. Let us define $cost(\mathcal{R}, x)$ as

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

where $A \overset{\mathcal{R}}{\leftarrow} \mathcal{A}$ means A is sampled from $\mathcal{A}$ according to $\mathcal{R}$. The intrinsic cost of $\mathcal{R}$ is defined to be $\max_{x \in \mathcal{X}} cost(\mathcal{R}, x)$, that is, the maximum cost of $\mathcal{R}$ on any input $x \in \mathcal{X}$. The *randomized complexity* of a problem can be 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 that solves the problem.

Similarly, we can measure the *distributional complexity* of a problem with respect to an input distribution $\mathcal{D}$. The cost of a deterministic algorithm A with respect to $\mathcal{D}$ can be defined as

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

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

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

which naturally captures the notion of the worst cost guaranteed by finding a good deterministic algorithm that solves the problem.

Yao's minimax principle states that, for any problem P, both (1) and (2) are equal, i.e.,

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

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

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

which naturally lower bounds the randomized complexity for P. If one can cleverly come up with a suitable distribution $\mathcal{D}$ on the inputs for P and prove that every *deterministic* algorithm that solves P incurs at least cost C on the distribution $\mathcal{D}$, it follows that the randomized complexity of P is at least C. Observe that the power of this technique lies in the fact that one can choose any distribution $\mathcal{D}$, and the lower bound is calculated by comparing the costs of all deterministic algorithms for the problem on $\mathcal{D}$.

1.2 Yao's Principle in Property Testing

The following two statements are equivalent:

1. For any *probabilistic* algorithm A having query complexity q, there exists an input x such that

$$\Pr_{\text{coin tosses of } A}[A(x) \text{ is wrong}] > 1/3.$$

2. There is a distribution $\mathcal{D}$ on the inputs such that for every *deterministic* algorithm having query complexity q,

$$\Pr_{x \overset{\mathcal{D}}{\leftarrow} \mathcal{X}}[A(x) \text{ is wrong}] > 1/3.$$

We will use the second statement for proving lower bounds in property testing.

2 Proving Lower Bounds in Property Testing

In this section, we review three examples where Yao's minimax principle is applied to prove lower bounds in property testing.

2.1 A Lower Bound for Testing 1^*

Theorem 1. *Given a string of n bits, where $n \geq 1$, any ϵ-tester for 1^n requires $\Omega(1/\epsilon)$ queries.*

The gist of the proof is that, by Yao's minimax principle, one can devise an input distribution $\mathcal{D}$ on which every deterministic algorithm with query complexity $o(1/\epsilon)$ 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. The input can be divided into $1/\epsilon$ blocks where each block is of length ϵn. Let y_i be an n-bit string where all 1's in the i^{th} block are flipped to 0's. Hence,

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

Observe that each y_i is ϵ-far from 1^n. We define our input distribution $\mathcal{D}$ as:

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

Proof. Fix a deterministic tester A which makes q queries.

1. If A doesn't accept 1^n, it is incorrect with probability at least 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 the q queries probe distinct blocks, A can only look in q blocks and thus, it has to accept $\left(\frac{1}{\epsilon} - q\right)$ number of y_i's. This means that A is incorrect with probability $(\frac{1}{\epsilon} - q)\frac{\epsilon}{2}$. If $q < \frac{1}{3\epsilon}$, then A's probability of failure is greater than $\frac{1}{3}$.

We have shown that $q < \frac{1}{3\epsilon}$ is 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 $\mathcal{R}$ with query complexity $q < \frac{1}{3\epsilon}$, there exists an input $x \in \mathcal{X}$ which doesn't guarantee $\Pr_{\text{coin tosses of } A}[A(x) \text{ is wrong}] < 1/3$. Hence, any ϵ-tester for 1^n requires $\Omega(1/\epsilon)$ queries. □

2.2 A Lower Bound for Testing Sortedness

In the previous lectures, we covered two different ϵ-testers for sortedness, with query complexity $O\left(\frac{\log n}{\epsilon}\right)$. One was based on spanners, whereas the other was based on binary search. For all constant $\epsilon \leq \frac{1}{2}$, it is shown that $\Omega(\log n)$ queries are required to decide, with probability at least 2/3, 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. *A $\frac{1}{2}$-tester for sortedness requires $\Omega(\log n)$ queries.*

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

For example, Figure 1 contains the desired lists for $n = 16$.

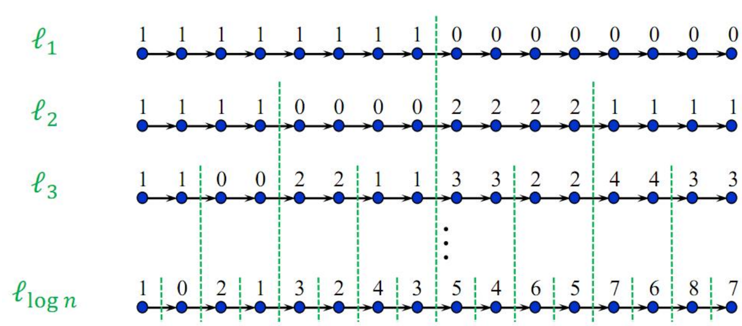

Figure 1: $\log n$ lists for proving a lower bound for the query complexity of a 1/2-tester for sortedness

Define an input distribution $\mathcal{D}$ in which each list ℓ_i has a probability of $\frac{1}{\log n}$. These lists have two useful properties:

- All lists are $\frac{1}{2}$-far from sorted. Each ℓ_i, for i such that $1 \le i < \log n$, contains 2^i runs†. For $1 \le k \le 2^{i-1}$, replacing each $(2k)^{\text{th}}$ run with $(2k-1)^{\text{th}}$ run makes the list sorted.
- For i, j such that $1 \le i < j \le n$, every pair (x_i, x_j) is violated in exactly one list above. This property directly follows from the construction. For example, consider the pair (x_3, x_5) in the above given lists. It is (1,1) in ℓ_1, (1,0) in ℓ_2, (0,2) in ℓ_3 and (2,3) in ℓ_4. Hence, the pair (x_3, x_5) is only violated in ℓ_2.

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

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 of consequtively queried positions. The algorithm rejects the list when it sees a violated pair. As every violated pair is observed in exactly one list, q queries can help the algorithm reject at most $q-1$ inputs. This amounts to a probability of at most $\frac{q-1}{\log n}$ for the algorithm to be correct. As all the input lists are $\frac{1}{2}$-far from sorted, A must reject all of them with probability at least 2/3. It means $\frac{q-1}{\log n} \ge \frac{2}{3}$, which implies that $q = \Omega(\log n)$. □

2.3 A Lower Bound for Testing Monotonicity on a Hypercube

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

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

A tester with 1-sided error accepts f if it detects no violated pair $(f(x), f(y))$, where $x, y \in \{0,1\}^n$. 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.

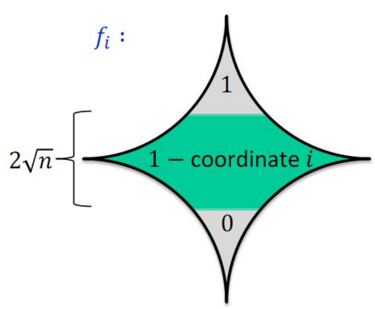

Figure 2: Pictorial depiction of f_i

In Figure 2, each point k in the green band of width $2\sqrt{n}$ has $f_i(k) = 1 - x_i$, where x_i is the i^{th} co-ordinate of k. Each point ℓ below this band has $f_i(k) = 0$ and each point m above the band has $f_i(m) = 1$. Consider an edge from $a = (x_1, ..., x_{i-1}, 0, x_{i+1}, ..., x_n)$ to $b = (x_1, ..., x_{i-1}, 1, x_{i+1}, ..., x_n)$. It is easy to see that this

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

edge is violated only when both a and b are in the green band. As the green band amounts to a constant fraction of vertices in $\{0,1\}^n$, each f_i, i such that $1 \leq i \leq n$, is ϵ-far from monotone, for some constant $\epsilon > 0$.

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 in f_i with probability at most $O(q/\sqrt{n})$.*

Before proving this lemma, we briefly describe the connection between Lemma 4 and Theorem 3. Suppose we show that, for a deterministic tester A that makes q queries, the number of functions for which A reveals a violation is $O(q\sqrt{n})$. Then, by an averaging argument, we can guarantee that there exists an f_i, where $i \in \{1, \ldots, n\}$, for which A detects a violation with probability at most $O(q/\sqrt{n})$. As a result, q must be $\Omega(\sqrt{n})$ to guarantee that A detects a violation in f_i with constant probability.

Proof. For the 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. Hence, any tester detects a violated pair in f_i only when it queries such a pair of vertices in the green band.

Let Q, such that $|Q| = q$, denote the set of queried vertices in the green band. We can consider Q as a graph where two vertices w and w' are connected by an edge if w' is obtained by changing exactly one bit in w. This graph has a spanning forest, and every violated pair must be in the same tree as there is a way to get one vertex from the other by a series of bit transformations, each of which are connected by an edge in Q. Also, as the violated pair differs in f_i at each vertex in the pair, there must exist adjacent vertices on the path between any violated pair that differ in their respective values for f_i. For any spanning forest of Q, the maximum number of edges possible in it is $q - 1$. Also, as every violated pair lies in the green band, the maximum distance between any 2 violated vertices is $2\sqrt{n}$. Hence, the total number of functions for which the queries reveal a violation is at most $((q-1) \times 2\sqrt{n})$, 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