Georgia Tech S'22 CS 6550/8803: Advanced Algorithms & Uncertainty

Lecture 8: Hardness: Yao's Minimax Lemma

Lecturer: *Sahil Singla* Last updated: February 5, 2022

In the last weeks, we have often seen that randomized algorithms admit better competitive ratios than any deterministic one. We could lower-bound the performance of a deterministic algorithm by using the perspective of an adversary that tried to make our algorithm look as badly as possible. Today, we will turn our attention to lower bounds and randomized (online) algorithms.

1 Yao's Minimax Lemma

Yao's Minimax Lemma is a very simple, yet powerful tool to prove impossibility results regarding worst-case performance of randomized algorithms, which are not necessarily online. We state it for algorithms that always do something correct but the profit or cost may vary. Such algorithms are called *Las Vegas* algorithms. We first state it for minimization problems because this is the usual way.

We assume that we have a class of deterministic algorithms $\mathcal{A}$ and a class of instances $\mathcal{X}$. In order to avoid technicalities, assume that both classes are finite. Algorithm $a \in \mathcal{A}$ on instance $x \in \mathcal{X}$ incurs cost $c(a, x) \in \mathbb{R}$. A randomized algorithm is simply a probability distribution over the set of deterministic algorithms $\mathcal{A}$. So, let A be a randomized algorithm (which is now a random variable), then A's worst-case cost is $\max_{x \in \mathcal{X}} \mathbb{E} c(A, x)$.

Theorem 1 (Yao's Minimax Lemma)**.** *Let A be any random variable with values in $\mathcal{A}$ and let X be any random variable with values in $\mathcal{X}$. Then,*

$$\max_{x \in \mathcal{X}} \mathbb{E} c(A, x) \geq \min_{a \in \mathcal{A}} \mathbb{E} c(a, X) \ .$$

Before proving the theorem, let us interpret what it means. The left-hand side of the inequality is what will will try to lower-bound: It is the worst-case performance of randomized algorithm A. The right-hand side will be easier to talk about, because algorithms are deterministic. This is a sort of average-case performance of the best deterministic algorithm in our class. The distribution over instances is arbitrary.

Roughly, the intuition for the proof is the following: if the input to a cost-minimization algorithm is chosen at random from some *known* distribution (instead of being worst-case), then we may assume W.L.O.G. that the optimal algorithm is deterministic. This is because we can choose the "best" (lest expected cost) deterministic algorithm in the support of the optimal randomized algorithm, whose cost can only be at most that of the randomized algorithm. Hence, to prove a lower bound of c on the cost of randomized algorithms, it suffices to design a distribution over inputs and argue that every deterministic algorithm incurs at least cost c for this distribution.

Proof. Let us first write the expectations as sums over all possible outcomes of X and A.

$$\mathbb{E}c(A,x) = \sum_{a\in\mathcal{A}} \Pr[A=a]c(a,x) \quad \text{and} \quad \mathbb{E}c(a,X) = \sum_{x\in\mathcal{X}} \Pr[X=x]c(a,x)$$

Now we use that the weighted average of a sequence is always upper-bounded by its maximum value. In our case, the weights are $\Pr[X=x]$. As $\sum_{x\in\mathcal{X}} \Pr[X=x]=1$, we have

$$\max_{x\in\mathcal{X}} \mathbb{E}c(A,x) = \max_{x\in\mathcal{X}} \sum_{a\in\mathcal{A}} \Pr[A=a]c(a,x) \geq \sum_{x\in\mathcal{X}} \Pr[X=x] \sum_{a\in\mathcal{A}} \Pr[A=a]c(a,x) \ .$$

We can now reorder the sums and get

$$\sum_{x\in\mathcal{X}} \Pr[X=x] \sum_{a\in\mathcal{A}} \Pr[A=a]c(a,x) = \sum_{a\in\mathcal{A}} \Pr[A=a] \sum_{x\in\mathcal{X}} \Pr[X=x]c(a,x) \ .$$

Finally, we can use that $\sum_{a\in\mathcal{A}} \Pr[A=a]=1$ the same way as above to obtain

$$\sum_{a\in\mathcal{A}} \Pr[A=a] \sum_{x\in\mathcal{X}} \Pr[X=x]c(a,x) \geq \min_{a\in\mathcal{A}} \sum_{x\in\mathcal{X}} \Pr[X=x]c(a,x) = \min_{a\in\mathcal{A}} \mathbb{E}c(a,X) \ . \qquad \square$$

The analogous statement holds for maximization problems, where we have a profit $p(a,x)$ that algorithm a achieves on instance x. By setting $c(a,x) = -p(a,x)$, we get the following corollary.

Corollary 2. . *Let A be a random variable with values in $\mathcal{A}$ and let X be a random variable with values in $\mathcal{X}$. Then,*

$$\min_{x\in\mathcal{X}} \mathbb{E}p(A,x) \leq \max_{a\in\mathcal{A}} \mathbb{E}p(a,X) \ .$$

2 Randomized Lower Bound for Online Set Cover

We will now use Yao's Lemma to show that for online Set Cover even randomized algorithms cannot be strictly $o(\log n)$-competitive (or $o(\log m)$-competitive), where n is the number of elements and m is the number of sets.

Theorem 3. *No algorithm for Online Set Cover is α-competitive for $\alpha < 1 + \frac{1}{2}\log_2 m$ (or $\alpha < \log_2(n+1)$-competitive).*

Proof. We consider m that are powers of two. Define the set system as follows. Take a complete binary tree of $n = 2m-1$ vertices, height $h = \log_2 m + 1$, and m leaves.

Potential elements e to be covered correspond to nodes in the tree. Only some of these potential elements will be have to be covered by our online algorithm, in particular elements along an (unknown) root-leaf path U^* will have to covered.

The sets $S \in \mathcal{S}$ correspond to the m leaves of the tree, where S_k contains all elements corresponding to the root-leaf path for the k-th leaf. We set $c_S = 1$ for all $S \in \mathcal{S}$.

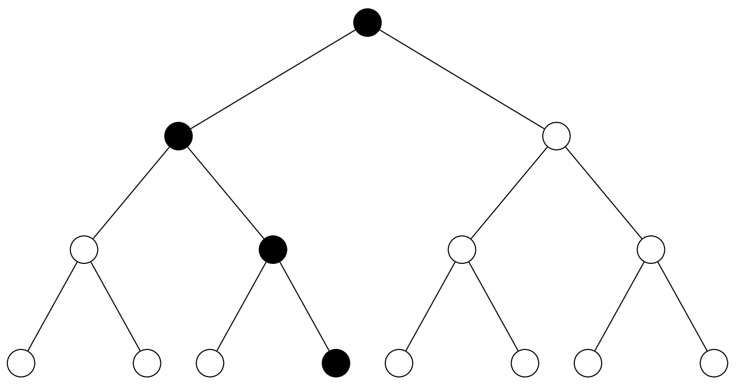

Figure 1: The binary tree used in the construction of the lower bound. The black nodes correspond to a potential sequence.

Our sequence σ reveals the set U^*, starting from the root and going downward. Note that any such σ can be covered by a single set $S \in \mathcal{S}$, namely the one that corresponds to the leaf where the path ends. That is, $\text{cost}(\text{OPT}(\sigma)) = 1$.

However, the algorithm initially cannot distinguish the m different sets in $\mathcal{S}$ because we only get to know what sets an element is contained in when it arrives. So, initially, we see only m placeholders in $\mathcal{S}$. Then the root arrives, which in contained in all of these sets. The second node on the path arrives, which is only contained in half of the sets and so on. Only in the very last step, we will know which set would have covered all of U^*.

We now claim that for any randomized online algorithm $\mathbb{E}\text{cost}(ALG(\sigma)) \geq 1 + \frac{1}{2}\log_2 m$ for some sequence σ of this form. In the notation of Yao's Lemma, this means that $\max_{x\in\mathcal{X}} \mathbb{E}c(A,x) \geq 1 + \frac{1}{2}\log_2 m$, where now A is an arbitrary randomized algorithm and $\mathcal{X}$ is the set of sequences we are considering.

Yao's Lemma tells us that $\max_{x\in\mathcal{X}} \mathbb{E}c(A,x) \geq \min_{a\in\mathcal{A}} \mathbb{E}c(a,X)$ for any choice of the random variable X, so it is sufficient to show how to choose X such that $\min_{a\in\mathcal{A}} \mathbb{E}c(a,X) \geq 1 + \frac{1}{2}\log_2 m$.

In our case, we will draw one leaf uniformly at random and take the path ending at this leaf. Equivalently, we can determine this leaf as follows: Start at the root and flip a coin whether to go left or right. At every inner node continue this procedure. Note that this is exactly the way the online algorithm gets to the path. It always only gets to know whether the following elements are in the left or the right subtree. The elements to further distinguish the leafs have not yet arrived.

Without loss of generality, consider a *lazy* (deterministic) algorithm. This means that the algorithm only buys an $S \in \mathcal{S}$ if the current element is not yet covered. If so, it buys only a single one. In Lemma, an algorithm may buy multiple sets at a time but it is easy to see that this cannot be cheaper.

Let C_t be the cost of the algorithm in the t-th step. Clearly $C_1 = 1$. Now consider any later step $t > 1$. We will show that $\mathbb{E}C_t = \frac{1}{2}$. To this end, observe that the algorithm has chosen exactly one set that covers the element appearing in step $t-1$. This can either belong to the left or to the right subtree. If the element in step t comes from this subtree, the algorithm does not choose another set and does not incur any cost. If the element comes from a different subtree, then the algorithm chooses a new set and incurs cost of 1. Both events happen with probability $\frac{1}{2}$, so $\mathbb{E}C_t = \frac{1}{2}$.

Overall, we get for any $a \in \mathcal{A}$

$$\mathbb{E}c(a, X) = \sum_{t=1}^{h} \mathbb{E}C_t = 1 + (h-1)\frac{1}{2} = 1 + \frac{1}{2}\log_2 m \ . \qquad \square$$

Remark: The above proof also implies the same hardness result for Online Fractional Set Cover since making a fractional decision at step t doesn't add power to the algorithm.

3 Randomized Lower Bound for Ski Rental

We now consider the Ski Rental Problem. We will show that no randomized algorithm can be better than $\left(1 - \left(1 - \frac{1}{B}\right)^B\right)^{-1}$-competitive. Note that this matches the guarantee of the randomized algorithm that we derived from the fractional primal-dual one.

Theorem 4. *For a fixed B, no Ski Rental algorithm is α-competitive for $\alpha < \left(1 - \left(1 - \frac{1}{B}\right)^B\right)^{-1}$.*

Also note that $\lim_{B\to\infty} \left(1 - \left(1 - \frac{1}{B}\right)^B\right)^{-1} = \frac{e}{e-1} \approx 1.58$.

Proof. The instances have a very simple structure, they are simple non-negative integers representing the number of skiing days. For the offline optimum OPT, we now have

$$c(\mathrm{OPT}, x) = \begin{cases} x & \text{if } x < B \\ B & \text{otherwise} \end{cases}$$

For any random variable X, this implies

$$\mathbb{E}c(\mathrm{OPT}, X) = \sum_{t=1}^{B-1} t \Pr[X = t] + B \Pr[X \geq B] = \sum_{t=1}^{B} \Pr[X \geq t] \ .$$

Deterministic algorithms are also easy to describe. They only differ in how long they wait until buying the skis. So a is again a non-negative integer, which means that the algorithm rents skis for a days before buying them on day $a+1$ (if this day exists). The cost of a is

$$c(a, x) = \begin{cases} x & \text{if } x \leq a \\ a + B & \text{otherwise} \end{cases}$$

So, the expected cost for a random X is

$$\mathbb{E}c(a, X) = \sum_{t=1}^{a} t \Pr[X = t] + (a + B) \Pr[X > a] = \sum_{t=1}^{a} \Pr[X \geq t] + B \Pr[X > a] \ .$$

Now, we have to find a distribution such that all deterministic algorithms a perform badly. The idea is to make all algorithms incur the same cost in expectation. This means that algorithms a and $a-1$ have the same cost

$$\mathbb{E}c(a-1, X) = \mathbb{E}c(a, X) \enspace ,$$

which means that

$$\sum_{t=1}^{a-1} \Pr[X \geq t] + B \Pr[X > a-1] = \sum_{t=1}^{a-1} \Pr[X \geq t] + \Pr[X \geq a] + B \Pr[X > a] \enspace .$$

So we need

$$B \Pr[X \geq a] = \Pr[X \geq a] + B \Pr[X \geq a+1] \quad \Leftrightarrow \quad \Pr[X \geq a+1] = \left(1 - \frac{1}{B}\right) \Pr[X \geq a] \enspace .$$

This is fulfilled if we set

$$\Pr[X \geq t] = \left(1 - \frac{1}{B}\right)^{t-1} \qquad \text{for all } t \geq 1 \enspace .$$

For this choice of X, we have

$$\mathbb{E}c(\mathrm{OPT}, X) = \sum_{t=1}^{B} \left(1 - \frac{1}{B}\right)^{t-1} = B\left(1 - \left(1 - \frac{1}{B}\right)^{B}\right) \enspace ,$$

whereas for all a

$$\mathbb{E}c(a, X) = \sum_{t=1}^{a} \left(1 - \frac{1}{B}\right)^{t-1} + B\left(1 - \frac{1}{B}\right)^{a} = B \enspace .$$

Now suppose a randomized algorithm A is α-competitive for $\alpha < \left(1 - (1 - \frac{1}{B})^{B}\right)^{-1}$. Then this means that

$$\mathbb{E}c(A, X) \leq \alpha \mathbb{E}c(\mathrm{OPT}, X) \enspace .$$

As we know $\mathbb{E}c(A, X) = \sum_{a \in \mathcal{A}} \Pr[A = a] \mathbb{E}c(a, X) = B$ and $\mathbb{E}c(\mathrm{OPT}, X) = B \cdot \left(1 - (1 - \frac{1}{B})^{B}\right)$. This is a contradiction. $\square$

Notes

The lecture is partly based on Thomas Kesselheim's notes[1].

References

[1] Andrew Chi-Chih Yao: Probabilistic Computations: Toward a Unified Measure of Complexity (Extended Abstract). FOCS 1977 (original paper introducing Yao's Minimax Lemma)

[1] http://tcs.cs.uni-bonn.de/lib/exe/fetch.php?media=teaching:ss20:vl-aau:lecturenotes05.pdf