Algorithms and Uncertainty, Winter 2017/18 Lecture 5 (4 pages)

Yao's Principle

Instructor: Thomas Kesselheim

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 Principle

Yao's principle is a very simple, yet powerful tool to prove impossibility results regarding worst-case performance 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 $\max_{x\in\mathcal{X}} \mathbf{E}\left[c(A,x)\right]$.

Theorem 5.1 (Yao's Principle)**.** *Let A be a random variable with values in $\mathcal{A}$ and let X be a random variable with values in $\mathcal{X}$. Then,*

$$\max_{x\in\mathcal{X}} \mathbf{E}\left[c(A,x)\right] \geq \min_{a\in\mathcal{A}} \mathbf{E}\left[c(a,X)\right] \ .$$

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.

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

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

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

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

We can now reorder the sums and get

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

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

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

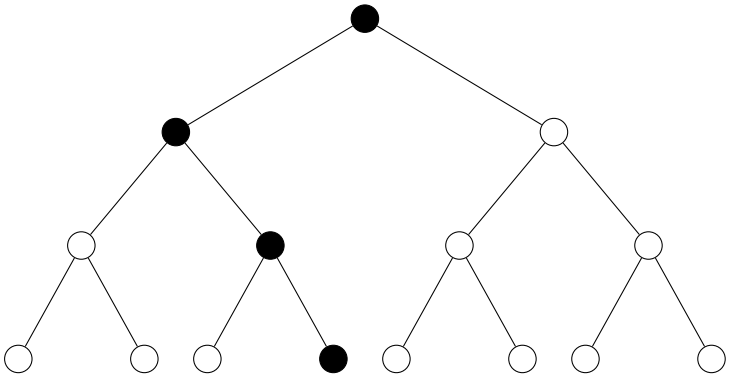

Figure 1: The binary tree used in the construction of the lower bound. The black nodes correspond to a potential sequence.

The analogous statement holds 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 5.2. . *Let A be a random variable with values in $\mathcal{A}$ and let X be a random variable with values in $\mathcal{X}$. Then,*

$$\min_{x \in \mathcal{X}} \mathbf{E}\left[p(A, x)\right] \leq \max_{a \in \mathcal{A}} \mathbf{E}\left[p(a, X)\right] \quad .$$

2 Randomized Lower Bound for Online Set Cover

We will now use Yao's principle to show that indeed for online Set Cover even randomized algorithms cannot be strictly $o(\log n)$-competitive.

Theorem 5.3. *No algorithm for online Set Cover is strictly c-competitive for $c < 1 + \frac{1}{2}\log_2 n$.*

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

Potential elements e to be covered correspond to nodes in the tree. Sets $S \in \mathcal{S}$ correspond to leaves, where S contains all ancestors in the tree. We set $c_S = 1$ for all $S \in \mathcal{S}$.

Our sequence σ will be the elements on one root-leaf path, starting from the leaf 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 $\text{cost}(\text{OPT}(\sigma)) = 1$.

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

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

In our case, we will drawn 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 principle, 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 tth step. Clearly $C_1 = 1$. Now consider any later step $t > 1$. We will show that $\mathbf{E}[C_t] = \frac{1}{2}$. 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 $\mathbf{E}[C_t] = \frac{1}{2}$.

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

$$\mathbf{E}[c(a, X)] = \sum_{t=1}^{h} \mathbf{E}[C_t] = 1 + (h-1)\frac{1}{2} = 1 + \frac{1}{2}\log_2 n \ .$$

□

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 $\frac{\mathrm{e}}{\mathrm{e}-1}$-competitive.

Theorem 5.4. *For a fixed B, no algorithm for Ski Rental is c-competitive for $c < \left(1 - \left(1 - \frac{1}{B}\right)^{B+1}\right)^{-1}$.*

Note that $\lim_{B\to\infty} \left(1 - \left(1 - \frac{1}{B}\right)^{B+1}\right)^{-1} = \frac{\mathrm{e}}{\mathrm{e}-1}$.

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

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

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

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

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

$$\mathbf{E}\left[c(a-1,X)\right] = \mathbf{E}\left[c(a,X)\right] \ ,$$

which means that

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

So we need

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

This is fulfilled if we set

$$\mathbf{Pr}\left[X \geq a\right] = \left(1 - \frac{1}{B}\right)^a \ .$$

For this choice of X, we have

$$\mathbf{E}\left[c(\mathrm{OPT},X)\right] = \sum_{t=1}^{a} \left(1 - \frac{1}{B}\right)^t = B\left(1 - \left(1 - \frac{1}{B}\right)^{B+1}\right) \ ,$$

whereas for all a

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

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

$$\mathbf{E}\left[c(A,X)\right] \leq c\mathbf{E}\left[c(\mathrm{OPT},X)\right] \ .$$

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