Algoritmos e Incerteza **(PUC-Rio INF2978, 2018.2)**

Lecture 2: Lower Bounds for Randomized Algorithms

21 August 2018

Lecturer: Marco Molinaro *Scribe: Lucas Murtinho*

The main topic of this lecture are lower bounds for randomized algorithms. We'll return to the problem of **ski rental** to study how randomized instances can help us find such bounds, thanks to **Yao's minimax principle**, which we'll partially prove. We'll then see in a toy example how to find such bounds. We end the lecture by presenting a new online problem (**paging**).

1 Recap and Motivation

Recap. In the previous lecture, we discussed the **ski rental problem**, in which the player must decide whether to buy a ski (thus avoiding any additional costs for the remainder of skiing season) or to keep renting skis for one day at a time. Crucially, there is no information available on when skiing season will end.

We also found and discussed a deterministic algorithm that is 2-competitive for this online problem, meaning that, for any instance of the problem, the overall cost paid by following this algorithm will never be more than twice the minimum cost possible:

$$\exists\, A \mid \forall\, I,\ \frac{A(I)}{OPT(I)} \leq 2$$

We proved that this is a *lower bound* for deterministic algorithms — that is, no deterministic algorithmic can do better than twice the optimum in the worst instance (which can be different for each algorithm).

One way to improve on this bound is to use *randomized* algorithms — and we also saw that one way to think about randomized algorithms is as algorithms that can behave as different deterministic algorithms with a given probability. We did not prove this, but a randomized algorithm can be roughly 1.58-competitive for the ski rental problem. [1]

Motivation. The rationale for using randomized algorithms is that we can "mix up" different deterministic algorithms in such a way that the worst case will be "averaged over". For instance, we can select at random between 2 algorithms, A_1 and A_2, such that A_1 performs well in A_2's worst case and vice versa. This way the *expected* result for the worst case is not as bad as it was before, when we had to pick one deterministic algorithm and stick to it.

This, however, presents us with an analytical difficulty: if randomized algorithms can "average" worst cases out, how can we determine a lower bound for them? Is there a way of saying, as we did for deterministic algorithms, that no randomized algorithm can be better than α-competitive, for a given value of α?

2 Lower Bounds for Randomized Algorithms

Let's briefly return to the ski rental problem to see how we can tackle this issue. Recall that the worst case for any deterministic algorithm is for skiing season to end right after the player buys the skis — this means they spent a lot of money for nothing.

In a randomized algorithm, however, the very decision of when to buy the skis is not predefined — the player will buy on different days with different probabilities. So what would a worst case even look like?

For this particular problem, the answer is simple enough: just select the instance that will yield the worst **expected value** for the randomized algorithm. This sort of "brute force" approach will work well in this case, because the problem is quite simple and we can evaluate the algorithm's quality in closed form. For more involved problems, however, finding this instance may not be as easy, so we'll need another way of finding lower bounds for randomized algorithms.

The strategy presented below can be thought of as a *reduction* from the randomized world to the deterministic world; we'll see how **deterministic algorithms' expected results on randomized instances** can be used to derive a lower bound for **randomized algorithms' results on deterministic instances**.

2.1 Yao's Minimax Principle

The main tool we'll be using is **Yao's minimax principle**, presented below. Let $A(I)$ be the cost paid by algorithm A when running on instance I, and $OPT(I)$ be the minimum cost possible for instance I. Then, Yao's minimax principle states that:

Theorem 2.1 (**Yao's minimax principle**). *Provided some technical conditions (which we won't discuss here) are met, the following equality is true:*

$$\min_{A_r} \max_{I_d} \frac{E[A_r(I_d)]}{OPT(I_d)} = \max_{I_r} \min_{A_d} E\left[\frac{A_d(I_r)}{OPT(I_r)}\right], \tag{1}$$

where min_{Ar} ranges over all randomized algorithms; max_{I_d} ranges over all deterministic instances; max_{I_r} ranges over all randomized instances; and min_{A_d} ranges over all deterministic algorithms.

To understand (1), let's think about each side of the equation separately before putting them back together.

The left-hand side.

$$\max_{I_d} \frac{E[A_r(I_d)]}{OPT(I_d)}$$

is the maximum ratio between the expected cost of a randomized algorithm and the optimum cost for a given instance; in other words, it's the worst possible approximation factor for a randomized

algorithm. Minimizing this quantity means selecting the best possible randomized algorithm for this problem.[1] So the LHS of (1) represents exactly what we are after: a **lower bound** on the competitive ratio for a randomized algorithm.

The right-hand side.

$$\min_{A_d} E\left[\frac{A_d(I_r)}{OPT(I_r)}\right]$$

is the best possible result in terms of approximation factor (for a deterministic algorithm) given a randomized instance. That is, we are no longer dealing with individual, deterministic instances, but with a **distribution of instances**, much in the same way we were looking at a distribution of algorithms before. Maximizing this quantity means finding the randomized instance for which the best approximation factor will be as bad as possible.

2.2 Yao's Theorem: First Interpretation

Online algorithms as a game. Perhaps the most natural interpretation of Yao's theorem is through the lens of game theory. Any online problem can be interpreted as a game between the *player*, who builds the algorithm, and the *adversary*, who builds the instance. The player's goal is to minimize the algorithm's competitive ratio; the adversary's goal is to maximize it.

The LHS of (1) can be interpreted under this light as follows:

1. The player builds a randomized algorithm.
2. *Knowing which algorithm the player built*, the adversary selects a deterministic instance to maximize its competitive ratio.

This puts the adversary in a position of strength, since they already know what the player did when making their decision. The player can randomize their algorithm while the adversary must choose a deterministic instance, which may at first seem like a disadvantage for the latter. But once the algorithm is known, there is no reason to randomize the instances; the adversary will simply select the instance that yields the worst approximation factor for the algorithm the player built.[2]

On the RHS of (1), the order is reversed:

1. The adversary builds a randomized instance.
2. *Knowing which instance the adversary built*, the player selects the deterministic algorithm to minimize the approximation factor over it.

[1] That is, if we rank algorithms according to their competitive ratios.

[2] This idea that randomization can only go so far, because the expected result of a distribution is bound by its minimum and maximum values, will play a crucial role in the partial proof of Yao's principle presented below.

Here the adversary is weaker, since it's the player who has information before making their move. And, as per the above, it makes no difference whether we allow the player to select a randomized algorithm; the best possible deterministic algorithm will do just as well.

Combining these understandings of the LHS and RHS of (1) leads to a remarkable conclusion: the same competitive ratio will be reached, whether selecting the worst possible instance for a randomized algorithm or the best possible algorithm for a randomized instance.

Still thinking in terms of game theory, this means that the *value* of the game (the quantity the players want to minimize or maximize; in our case, the competitive ratio) does not depend on who plays first. In other words, there is an optimal (mixed, or randomized) strategy for each player, and if both players follow these optimal strategies the game's result will always be the same, no matter who is in an apparent position of strength. This result, which holds for "every finite, zero-sum, two-person game" [2], is the crux of **Von Neumann's minimax theorem**, the foundation stone of game theory.

2.3 Second Interpretation of Yao's Principle

We want to find a lower bound α — a value that limits how low competitive ratios for the problem at hand can go. Since the LHS of (1) gives us precisely this best possible competitive ratio, this is equivalent of showing that LHS $\geq \alpha$ — or that, for all possible algorithms, there is an instance such that its approximation factor will be no smaller than α.

$$\forall \boldsymbol{A}_r, \exists I \mid \frac{E[A_r(I)]}{OPT(I)} \geq \alpha \tag{2}$$

But Yao's principle says that the LHS of (1) is equal to its RHS, which means that RHS $\geq \alpha$ — or that there is one randomized instance for which no algorithm will have an approximation factor smaller than α.

$$\exists I_r \mid \forall A,\ E\left[\frac{A(I_r)}{OPT(I_r)}\right] \geq \alpha \tag{3}$$

This greatly simplifies what needs to be done to find a lower bound for randomized algorithms. Going from (2) alone, we would need to show that every possible randomized algorithm has competitive ratio worse than α — so we would need to find one bad instance for each possible algorithm. Thanks to (3), however, we can simply build a randomized instance for which no algorithm can achieve a better approximation factor than the desired lower bound, and we're done. The fact that a single randomized instance can be "bad" for any given algorithm (and that this "badness" is enough to prove lower bounds) means we can more easily find lower bounds for randomized algorithms.

2.4 Proof of (One Direction of) Yao's Principle

To show that Yao's principle holds, we should prove that its LHS is greater than or equal to its RHS, and vice-versa. One of these directions, however, is much easier to prove than the other; since the adversary wants to maximize the competitive ratio, and they are in a position of strength on the LHS of (1), it stands to reason that the easier direction is to prove that the LHS of (1) is no smaller than its RHS. (A similar reasoning regarding the player also holds.)

Let's start by proving a claim which relates the maximization of the approximation factor for a randomized algorithm to the minimization of the approximation factor for a randomized instance:

Claim 2.2. *For any online problem, the following inequality always holds:*

$$\max_{I_d} \frac{E[A_r(I_d)]}{OPT(I_d)} \geq \min_{A_d} E_{I_r}\left[\frac{A_d(I_r)}{OPT(I_r)}\right], \tag{4}$$

where max_{I_d} ranges over all deterministic instances and min_{A_d} ranges over all deterministic algorithms.

Proof.

$$\max_{I_d} \frac{E[A_r(I_d)]}{OPT(I_d)} \geq E_{I_r}\left[E_{A_r}\left[\frac{A_r(I_r)}{OPT(I_r)}\right]\right] = E_{A_r}\left[E_{I_r}\left[\frac{A_r(I_r)}{OPT(I_r)}\right]\right] \geq \min_{A_d} E_{I_r}\left[\frac{A_d(I_r)}{OPT(I_r)}\right]$$

The first inequality follows from the fact that the expected value will be no larger than the maximum; the third term follows from the second by simple rearrangement (the expected values are weighted sums that can be calculated in any order); and the second inequality follows from the fact that the expected value will be no smaller than the minimum. □

Now we can prove that the LHS of (1) is no smaller than its RHS.

Lemma 2.3. *For any online problem, the following inequality always holds:*

$$\min_{A_r}\max_{I_d} \frac{E[A_r(I_d)]}{OPT(I_d)} \geq \max_{I_r}\min_{A_d} E_{I_r}\left[\frac{A_d(I_r)}{OPT(I_r)}\right], \tag{5}$$

where min_{Ar} ranges over all randomized algorithms; max_{I_d} ranges over all deterministic instances; max_{I_r} ranges over all randomized instances; and min_{A_d} ranges over all deterministic algorithms.

Proof. Claim 2.2 means that the competitive ratio of any randomized algorithm will be no smaller than the best approximation factor for any randomized instance. Since its proof makes no assumption about which randomized algorithm and which randomized instance we are dealing with, it

is valid for the best possible randomized algorithm, which minimizes the LHS of (4), and for the worst possible randomized instance, which maximizes the RHS of (4). So we have

$$\min_{A_r} \max_{I_d} \frac{E[A_r(I_d)]}{OPT(I_d)} \geq \max_{I_r} \min_{A_d} E_{I_r}\left[\frac{A_d(I_r)}{OPT(I_r)}\right] \qquad \square$$

It remains to show that the LHS of (1) is no larger than its RHS. This, it turns out, is much more complicated; it is equivalent to problems such as linear programming duality, the Hahn-Banach separation theorem, or Brouwer's fixed point theorem. It is also, however, equivalent to online linear optimization, which we'll see later on in the course (and which we'll then be able to use to prove this other direction of the equality).

2.5 An Application of Yao's Principle on a Subset of the Ski Rental Problem

Let's return to the toy version of the Ski Rental Problem we previously worked with, in which we look only at the three first days of skiing season and consider that the cost of buying the ski, b, is 2. In Table 1:

- Each row I_i is an instance of the problem, with skiing season ending on day i.
- Each column A_j is a deterministic algorithm, with the player buying the skis on day j (A_0 is the instance in which the player never buys the skis).
- Each cell (i, j) represents the ratio between A_j's cost for instance I_i and the optimal cost for this same instance.

	A_1	A_2	A_3	A_0
I_1	2	1	1	1
I_2	1	3/2	1	1
I_3	1	3/2	2	3/2

Table 1: A simplified instance of the Ski Rental problem.

For this reduced version of the problem, a randomized algorithm A_r that behaves as algorithm A_1 with probability $1/3$ and as algorithm A_2 with probability $2/3$ will have a competitive ratio of $4/3$. Indeed, if $A(I)$ represents the approximation factor of algorithm A for instance I, we have:

$$A_r(I_1) = \frac{1}{3} \times 2 + \frac{2}{3} \times 1 = \frac{2}{3} + \frac{2}{3} = \frac{4}{3}$$
$$A_r(I_2) = \frac{1}{3} \times 1 + \frac{2}{3} \times \frac{3}{2} = \frac{1}{3} + 1 = \frac{4}{3}$$
$$A_r(I_3) = \frac{1}{3} \times 1 + \frac{2}{3} \times \frac{3}{2} = \frac{1}{3} + 1 = \frac{4}{3}$$

This is a better result than those achieved by any of the deterministic algorithms considered. Is this the best we can do? Can we prove it?

The answer to both questions is yes, and we can easily show it by using the minimax principle. By finding a randomized instance for which no deterministic algorithm can perform better than $4/3$ times the optimum, we will define a lower bound for the competitiveness of any randomized algorithm; and, since we just found an algorithm with that very same competitive ratio, this is a tight bound.

Let's consider the randomized instance I_r that behaves as I_1 with probability $1/3$ and as I_3 with probability $2/3$. The ratio between each algorithm's result and the optimum for I_r will be:

$$A_1(I_r) = \frac{1}{3} \times 2 + \frac{2}{3} \times 1 = \frac{2}{3} + \frac{2}{3} = \frac{4}{3}$$
$$A_2(I_r) = \frac{1}{3} \times 1 + \frac{2}{3} \times \frac{3}{2} = \frac{1}{3} + 1 = \frac{4}{3}$$
$$A_3(I_r) = \frac{1}{3} \times 1 + \frac{2}{3} \times 2 = \frac{1}{3} + \frac{4}{3} = \frac{5}{3}$$
$$A_0(I_r) = \frac{1}{3} \times 1 + \frac{2}{3} \times \frac{3}{2} = \frac{1}{3} + 1 = \frac{4}{3}$$

No deterministic algorithm can do better than $4/3$ of the optimum for this instance — and therefore, neither will any randomized algorithm. More importantly for our purposes, no algorithm (randomized or not) will be better than $4/3$-competitive.

The expanded matrix in Table 2 shows the results for the randomized algorithm A_r and the randomized instance I_r presented above. As expected, the competitive ratio for algorithm A_r on instance I_r is exactly 4/3 — it couldn't be higher, for this is a 4/3-competitive algorithm given the deterministic instances we're working with; and it couldn't be lower, given we proved every available deterministic algorithm will have at least this competitive ratio for this instance.

	A_1	A_2	A_3	A_0	A_r
I_1	2	1	1	1	$4/3$
I_2	1	$3/2$	1	1	$4/3$
I_3	1	$3/2$	2	$3/2$	$4/3$
I_r	$4/3$	$4/3$	$5/3$	$4/3$	$4/3$

Table 2: Expanded version of Table 1, including an optimal randomized algorithm and the randomized instance that proves its bound.

3 Stray Observations

Before presenting the next problem with which we'll be working, let's just register a couple of observations made in class that can be useful:

Ski Rental applications. Even though it is a very simple problem, ski rental can be regarded as a stylized version of some real world problems of note:

- **IP snooping** [1] and other network-related problems.
- **Sleep or not**: Suppose there's a machine that will be used at some point in the future. Leaving it on means paying a small cost at each period t; turning it off eliminates such costs, but a large cost must be paid in the future when the machine needs to be turned back on. When (if at all) should we turn off the machine? (In this scenario, turning off the machine is equivalent to buying the ski, as we'll incur on a large cost later on.)
- Should we **wait for the elevator** or take the stairs? Each moment we wait for the elevator means "paying" a small amount of time. The stairs take longer, but if the wait for an elevator is long enough it makes sense to take them — just as it makes sense to buy a ski to avoid large rental costs.

Adversary models. The implicit assumption about the adversary described above is that they are *oblivious*, meaning they know the *distribution* of algorithms the player selects (i.e., they know which randomized algorithm the player built), but not the *outcome* of any given play. If the outcome were known, we would be dealing with an *adaptive* adversary — and, although we won't expand on this, it's easy to see this means randomization will no longer help the player, since the adversary can simply adapt their behavior to the deterministic algorithm drawn from the randomized algorithm's distribution. The standard adversary model used when thinking about online problems is the oblivious one, and it's the one we'll use throughout this course.

4 The Paging Problem

The next problem we'll deal with is the **paging problem**, which can be defined as follows. Suppose we have two levels of memory: a smaller and faster one, named **cache**; and a larger, slower one, which we'll call **disk**. At each point in time, there is a **request** for a **page**; all pages are stored in the disk, and must be copied to the cache upon request if they are not already there.

Ideally, all pages would be in the cache all the time, but there are more pages than we can fit there. So, once the cache is full and a request for a page not in it comes up, the problem becomes: which page must we delete from the cache in order to put the requested one there? Our goal is to minimize the number of removals in the process.

4.1 Definition

Let's define the problem more formally:

- There are K pages stored in disk.
- There are $k < K$ slots in cache, and each slot may contain at most one page.

- There is an unknown sequence of n requests, $r_1, ..., r_n$; each requests calls for one of the K pages to be in the cache.
- If, at moment i, the page r_i is not in the cache, it must be copied there; if the cache is full at that moment, a page that's already in the cache must be removed to make place for the requested one.
- Each removal from the cache has a cost of 1.[3]
- The goal is to minimize the cost (i.e., minimize the number of removals) to fulfill all n requests.

The online aspect of the problem comes from the fact that the request sequence is not known in advance.

4.2 Example

A simple example may help understand the problem, and also what a good strategy for it would be. Suppose we have $K = k + 1$ pages, meaning once the cache is full a single page will be left out. Suppose also that the request sequence is $[1, 2, ..., k, k + 1, 1, 2, ..., k - 1]$. In other words, all pages are requested in order; and, once all pages have been requested, all pages except the last two are requested again in the same order. (So we have $(k + 1) + (k - 1) = 2k$ requests.)

In this case, the first k requests would be fulfilled at cost zero, because the cache is not full and there is no need to remove anything from it. So the first decision to be made is when $k + 1$ is requested. Which page should we remove them?

A classic online algorithm for this problem is **Least Recently Used**, or LRU: remove from the cache the least recently used page. In this case, LRU would tell us to remove page 1 from cache, at a cost of 1.

The very next request would be for page 1, and LRU would tell us to remove page 2 — which would be the very next request, and so on. So LRU would have us pay 1 removal per request from moment $k + 1$ to moment $2k$, and the total cost for this instance would be $2k - k = k$.

What about the minimum possible cost? If we knew the whole request sequence in advance, then we'd know in the moment $k+1$ (when page $k+1$ is requested) that all pages in the cache will be requested again, except for page k. So page k would be removed and no other removal would be needed. The minimum cost for this instance would therefore be 1.

Since LRU's cost for this instance is k and the best possible cost is 1, it doesn't look like it's a very competitive algorithm. But in fact it can be shown that no deterministic algorithm can be better than k-competitive for this problem.[4] This proof will be presented in the next set of lecture notes.

[3] It would be more natural to consider a cost of *adding* pages to the cache instead of removing them from it, but in practice it doesn't matter; the only difference is that you get some page additions for free at first, while the cache is not full.

[4] We did not show that LRU is k-competitive, only that it can be *at best* k-competitive; but it is indeed the case that LRU is k-competitive.

References

[1] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki. Competitive randomized algorithms for non-uniform problems. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 301–309. Society for Industrial and Applied Mathematics, 1990.

[2] E. W. Weisstein. Minimax theorem. [Online; accessed 31-August-2018 at http://mathworld.wolfram.com/MinimaxTheorem.html].