Algoritmos e Incerteza **(PUC-Rio INF2979, 2017.1)**

Lecture 01: Introduction to Online Algorithms

14/08/2018

Lecturer: Marco Molinaro *Scribe: Tomás Gutierrez*

When studying algorithms to solve problems, usually we start with traditional algorithms. In this situation, every information of an instance I is known prior to the beginning of the decision process. However, several situations present a different challenge, in which different pieces of information of an instance are revealed through time. For example, we may be dealing with the flow of prices for a particular stock and deciding wether to buy it or sell it to maximize our profits, or with a flow of different items that need to processed minimizing its cost. In this context, one possible approach is the so called online algorithms, which makes decisions as new information is revealed regarding a particular instance. Online algorithms can be characterized as the following:

1. Time is split in periods t = 1, 2, 3, ...
2. At time t', an information concerning the instance is revealed (such as the stock price at t')
3. The algorithm takes some action using all information available up to time t' (buy, sell or hold the stock, for example)
4. The objective is measured by some objective function (measure) that must be maximized or minimized, depending on the context.

Some observations are in order: First, we usually consider that there is little or no prior information to the beginning of the process. Also, there is no prediction (or distribution) for future outcomes in this first model. We begin focusing on a worst-case model, and we do not focus on efficiency measures such as running time etc.

1 The Ski Rental Problem

The Ski Rental Problem is a name give to a class of problems in which there is a choice between paying a one-time cost (b) or a repeating cost ($r < b$) through the process. An illustrative example, from which this class is named can be described as:

Suppose a Ski Resort is open during ski season. When you arrived at the Ski Resort, the only information concerning the weather that you have is wheter the station will be open or closed for the day. You realize that you do not have any equipment, so you face a choice. You can either rent it for \$1 for one day or buy it, for \$b, where b $>$ 1. In case you decide to rent it, you will face the same choice again in the following day, in case the resort is opened. In case you buy it, you do not need to do any other action. At each day, you do not know whether the resort will be open the day

after, and thus do not know how long the ski season lasts. Propose an algorithm that minimizes the cost of skiing during the ski season.

Some observations:

- Once the resort closes for a day, it will not reopen, and you cannot rent equipment if the station is closed.
- You are addicted to skiing, so as long as the resort is open you will be there.
- When deciding to rent or buy at time t, you only know that the resort will be open in the t, but do not have any information about the following day (no probability distribution or prediction models).
- At the end of the season, you cannot sell the equipment in case you buy it.

How do we approach this problem? The first thing we need to do is analyze it from an algorithmic perspective. Assume that the ski station will be opened for l days, which is unknown to the decision maker. Thus, an instance I of this problem can be characterized by the pair (b, l), where b is the purchase price of the equipment, known at $t = 0$ (prior to the beginning of the season).

Let us analyze a first instance, and a possible sequence of decisions, for $l = 5$ (unknown to the decision maker / player) and $b = 10$ (known prior to the beginning of the season). Remember that at each step the player does not know whether the station will be opened the following day.

Time t	Track Status	Decision	Cost
t = 1	Open	Rent	+1
t = 2	Open	Rent	+1
t = 3	Open	Rent	+1
t = 4	Open	Rent	+1
t = 5	Open	Buy	+10
t = 6	Close	-	-
Total			14

How good is this solution? Moreover, how can we measure and compare algorithms given the uncertainty of the instance? To answer that, we need to formalize a concept called **offline optimum**.

Definition: The **offline optimum** is the lowest cost (in case of a minimization problem; highest reward in case of a maximization problem) possible for an algorithm that already knows all information concerning the instance (perfect information). In the Ski Rental problem, it would be someone that somehow knows how long the ski season would last *a priori*.

More specifically, in our example the offline optimum would be to rent the ski everyday for the 5 days, totalling a final cost of +5. Notice that the offline optimum will always be equal or better

than the online solution (provided by the online algorithm), since it is the best strategy possible. But since the instances can vary, how do we compare an algorithm solution to the offline optimum?

Definition: An algorithm $algo$ is α-competitive. if, for *every* instance I:

$$alg(I) \leq opt(I), \tag{1}$$

where $alg(I)$ returns the cost of the algorithm $algo$ for the instance I and $opt(I)$ is the offline optimum for instance I.

Notice that in the Ski Rental problem, the offline optimum of a given instance (b, l) is given by the minimum value between b and l. In case the season lasts more that the purchase cost ($l \geq b$), the best strategy is to buy the equipment in day 1. In case $l \leq b$, the best strategy is to rent the equipment everyday. Thus:

$$opt(I) = \min\{b, l\} \tag{2}$$

1.1 A Deterministic Algorithm for the Ski Rental problem

Let us analyze it from the player's perspective through an example. Now, we do not know when the season will finish, so l is unknown to us. The only information we have is the purchase price b, say, \$10. Suppose our algorithm $algo$ decides to buy the equipment at day 1. How competitive is it?

When evaluating competitiveness, it is useful to think that there is a game being played, where player 1 (the decision maker) chooses an algorithm that minimizes his cost, and player 2 tries to maximizes the cost for player one, given the algorithm selected. This is equivalent to say that for each algorithm proposed, we will focus on its worst case, to check its robustness. With that in mind, it is easy to notice that the worst scenario for $algo$ would be l = 1. In this case, the cost is 10 and the offline optimum is equal to 1. Thus, $algo$ is 10-competitive, α = 10.

$$algo(I) \leq 10 \times opt(I) \tag{3}$$

Maybe we can get a better algorithm in terms of competitiveness. Suppose that now we decide to buy the equipment in case $t = b$ and rent until then (for up to $b - 1$ days). How does it perform? To answer this question, it is useful to write generally the cases for each algorithm, the proposed $algo$ and the offline optimum strategy opt for a given instance I:

$$opt(I) = \begin{cases} b & \text{if } b \leq l \\ l & \text{if } b > l \end{cases} \tag{4}$$

$$algo(I) = \begin{cases} 2b - 1 & \text{if } b \leq l \\ l & \text{if } b > l \end{cases} \tag{5}$$

To understand the results above, it is useful to think about each case:

- case (1), $b \leq l$: In this situation, we reach the trigger point and buy the equipment at $t = b$. Thus, our total cost is $b - 1$ (for each rent in the first $b - 1$ days) plus b for buying the equipment. Then, we simplify $b - 1 + b$ to $2b - 1$. The ratio between *algo(I)* and *opt(I)* is, then, $\frac{2b-1}{b} < 2$.

- case (2), $l < b$: In this case, we rent for l days until the season ends. Therefore, our cost is l. The ratio between *algo(I)* and *opt(I)* is, in this case, $\frac{l}{l} < 1 < 2$.

Lemma 1.1. *The algorithm that buys at time $t = b$ is 2-competitive.*

Can we do better? (with a deterministic algorithm)

1.2 Lower Bound for Deterministic Algorithms

Lemma 1.2. *For any deterministic algorithm* algo*, there exists an instance* I *such that*

$$algo(I) \geq 2 \times opt(I) - 1 \tag{6}$$

Proof. Consider an instance with a given b. Note that the algorithm *algo* indicates when to buy, let us say x. Consider the instance I_x where the decision maker buys at time *x*, and the station closes the following day (that is, $l = x$ and the station closes at $x + 1$). Thus, we have that $algo(I_x) = (x - 1) + b$ while $opt(I_x) = \min\{b, x\}$.

Consequently,

$$algo(I_x) \geq 2 \times \min\{b, x\} - 1 = 2 \times opt(I_x) - 1, \tag{7}$$

as desired. □

1.3 Randomized Algorithms

Informally, we can think of it as an algorithms that decides how to behave based on some random outcome. The main ideia is to include a random variable in our decision making process.

For example, in the Ski Rental problem, consider the following algorithm that behaves according to a flipped coin:

$$algo(I) = \begin{cases} \text{buys on day one (t = 1)} & \text{if it is heads,} \\ \text{rents every day} & \text{if it is tails.} \end{cases} \tag{8}$$

How can we assess the cost of this algorithm? Expected value of the cost, or expected cost.

$$\mathbb{E}_{algo}(algo) = p_{tails} \times l + p_{heads} \times b \tag{9}$$

Definition: A randomized algorithm $algo$ is α-competitive if for every instance I, the expected cost $\mathbb{E}(algo)$ is less or equal to $\alpha \times opt(I)$, i.e

$$\mathbb{E}_{algo}(algo) \leq \alpha \times opt(I). \tag{10}$$

Note that if $algo$ is a deterministic algorithm, then $\mathbb{E}[algo(I)] = algo(I)$.

Back to our example, we know that $\mathbb{E}_{algo}(algo) = p_{tails} \times l + p_{heads} \times b$. But how competitive is it? Once again, let's think about each case:

- case(1), $l \geq b : \mathbb{E}_{algo}(algo(I)) = \frac{1}{2} \times (b + l) and opt(I) = b$ However, how much bigger is l compared to b? We cannot establish a value fo α, as it can be that $l >> b$. We say that this algorithm is ∞-competitive.
- case(2), $l < b$: as case(1) is unsolved, we do not need to explore case(2).

1.4 A Matrix/Game view

Lets draw a matrix with every possible deterministic algorithm in the columns and every possible instance (in our Ski Rental problem (b, l)) in the rows. That way, each entry in the matrix is $\frac{algorithm_j(instance_i)}{opt(instance_i)}$, which is the competitiveness of the algorithm j to the particular instance i. Ranging through every instance possible, the biggest value represents the competitiveness of the algorithm. In order words, the biggest entry in each column j is the competitiveness of the algorithm j, α.

	A_0	A_1	A_2	A_4	...
I_0	$\frac{A_0(I_0)}{opt(I_0)}$	$\frac{A_1(I_0)}{opt(I_0)}$	$\frac{A_2(I_0)}{opt(I_0)}$	$\frac{A_3(I_0)}{opt(I_0)}$	...
I_1	$\frac{A_0(I_1)}{opt(I_1)}$	$\frac{A_1(I_1)}{opt(I_1)}$	$\frac{A_2(I_1)}{opt(I_1)}$	$\frac{A_3(I_1)}{opt(I_1)}$	...
I_2	$\frac{A_0(I_2)}{opt(I_2)}$	$\frac{A_1(I_2)}{opt(I_2)}$	$\frac{A_2(I_2)}{opt(I_2)}$	$\frac{A_3(I_2)}{opt(I_2)}$	...
...	...	...	...	...	...

Therefore, this matrix presents every possibility for every deterministic algorithm possible in a particular problem. It is a powerful tool to analyze algorithms, specially when projecting random algorithms. Let us see why.

We focus only on a part of this matrix. Assume that $b = 2$ (purchase price of the equipment) and $l \leq 3$. That way, we have 3 possible instances $(b, l) : (2, 1), (2, 2)$ and $(2, 3)$. In this "world", we can build 4 deterministic algorithms: A_0 (always rent), A_1 (buy at day 1), A_2 (buy at day 2) and A_3 (buy at day 3). With that in mind, we can build our matrix representing every combination of algorithm and instance. Each entry represents the competitiveness (compared to the offline optimum) of each algorithm in each instance.

	A_0	A_1	A_2	A_4
$(2, 1)$	1	2	1	1
$(2, 2)$	1	1	3/2	1
$(2, 3)$	3/2	1	3/2	2

Comparing the deterministic algorithms, we see that A_0 and A_2 are (3/2)-competitive in this subset. *Is it possible to design a better algorithm?*

To answer this question, we need first to acknowledge that every random algorithm can be thought as a distribution over deterministic algorithms. To every stochastic event that drives the behavior of the algorithm, one deterministic algorithm takes place. In the example above, the result of the coin (the random event) determines the deterministic algorithm that takes place.

Now, consider an algorithm called ALGO that is defined in the following way:

$$ALGO(I) = \begin{cases} A_0 & \text{p = } 0.5 \\ A_2 & \text{p = } 0.5 \end{cases} \tag{11}$$

That is, it is a combination of both algorithms (A_0 and A_2) with equal probabilities. Since it is a random algorithm, we fulfill the entries in the columns of the matrix by its expected value divided by the offline optimum. In our case:

$$\frac{\mathbb{E}(ALGO(instance_i))}{opt(instance_i)} = \frac{\frac{1}{2} \times (A_0(instance_i) + A_2(instance_i))}{opt(instance_i)} \tag{12}$$

In this case, we reach the following table:

	ALGO
$(2,1)$	1
$(2,2)$	5/4
$(2,3)$	3/2

Notice that the competitiveness (biggest value) remains the same, 3/2. This leads to a key insight: *we want to combine algorithms with advantages in different instances!* In ALGO, we combined two algorithms with bad performances in the same instance. The power of random algorithms is exactly to function simultaneously as different deterministic algorithms with different worst-cases. In this case, there will not be an "extremely" worst instance for all behaviors of the algorithm.

Can we build a random algorithm with a competitiveness better than 3/2? Lets consider a new algorithm ALGOR that is defined in the following way:

$$ALGOR(I) = \begin{cases} A_1 & \text{p = } 0.25 \\ A_2 & \text{p = } 0.75 \end{cases} \tag{13}$$

That way, we reach the following table:

	ALGOR
$(2,1)$	5/4
$(2,2)$	11/8
$(2,3)$	11/8

Analyzing the table above, we see that the competitiveness for $ALGOR$ under these conditions is 11/8, lower than 3/2. This is an example of a random algorithm that is better than all deterministic algorithm, by combining different deterministic algorithms with different weaknesses and strengths.

How much better can we be? Is there a lower bound for how competitive can an algorithm be for the Ski Rental problem?

Theorem 1.3. *There exists an randomized algorithm for the Ski Rental problem that is* $\frac{e}{e-1}$*-competitive. And there is no better algorithm than this one.*

Proof sketch. To design the algorithm, we need to set the probability p_t that the algorithm buys on time t (if the game reaches this point). We can then calculate the theoretical value of $\mathbb{E}_{\mathcal{A}}\mathcal{A}(I)$, $OPT(I)$ for every instance I (since the instances are simple and parametrized by just the two values (b,l)). Then we can find the worst instance I^* and optimize the probabilities p_t for that instance. □