Algoritmos e Incerteza **(PUC-Rio INF2982, 2022.2)**

Lecture 1: Online Algorithms

15 August 2022

Lecturer: Marco Molinaro *Scribe: Mateus Levi Simões Fernandes*

When a student first learns about algorithms, they'll commonly use algorithms that receive the entirety of their inputs *before* the start of their execution.

While such a situation is very helpful for the task of creating the most optimized algorithm possible for a given problem, it is hardly realistic. In an increasingly data-driven world, the constant arrival of new data should not only be expected but exploited while designing an algorithm, particularly in the case of real-time applications.

In order to tackle these circumstances, one can use online algorithms: algorithms that receive their inputs over time, processing new data as they run. At each instant $t \in [1, T]$, the algorithm:

1. Receives information: for example, the value of a financial stock at t or the string object resulting from a search engine query;

2. Decides on how to respond to the data received: for example, holding or selling the stock or choosing the ads to be shown along with an user's query results.

The end goal of an online algorithm is the optimization of some previously defined objective function; in simpler terms, the minimization of the cost associated with an action (inversely, the maximization of one's profit).

1 The Ski Rental Problem

To delve into the matter of online algorithms, we'll present a simple example: the Ski Rental Problem.

Suppose the ski season has just started and thus the local ski resort has just opened its doors. As big fans of skiing, we'd like to ski every day of the season; in order to do so, we need a pair of skis. Thankfully, the resort offers skis for both rent and purchase. Rent is done on a daily basis, but one can purchase a pair of skis at any point during the season.

Ideally, we'd like to minimize our costs with skiing while still getting to ski every day. The problem is that we do not know the duration of the ski season (and thus when the ski resort will close) and we have no information to try and deduce it. The only thing we can do is check daily if the resort is still open: if it is, we can then decide if we should buy or rent a pair of skis for that day (if we haven't already bought one).

Notice how we've presented what is the basis for an online algorithm. We have a daily input in the form of the status (open or closed) of the resort, based on which we make a decision to either buy or rent skis. Our goal is to minimize our total costs (the sum of renting and/or buying the skis) throughout the duration of the season.

Assuming that the rent cost is unitary (i.e. \$1), an instance of the Ski Rental Problem is defined by two parameters: the known *a priori* cost \$$b$ of buying a ski and the initially unknown duration l of the ski season (where l is the last day the season is open).

We can list a few initial conclusions about the problem:

1. A good algorithm shouldn't make the decision to buy the skis too "late" (that is, in a day close to l) - since that means spending a value close to l plus the full cost of the purchase;
2. It also shouldn't make the decision to buy the skis too early: if l is small, renting skis every day is probably cheaper than the upfront cost of a purchase;
3. The usefulness of the information we receive each day is limited to that day: the resort being open at a day t does not help us determine if it'll be open in $t + 1$.

Let's look at a specific instance of the problem in order to delve further into it. Table 1 shows an example of algorithm execution for an instance where $b = 4$ and $l = 5$, though it is important to remember that we do consider the value of the l parameter unknown until the end of the execution.

Day	Resort Status	Decision	Cost
1	Open	Rent	+1
2	Open	Buy	+4
3	Open	-	+0
4	Open	-	+0
5	Open	-	+0
6	Closed		

Table 1: An instance of the Ski Rental Problem where $b = 4$ and $l = 5$. In this case, total cost equals to \$5.

We can easily see that the optimal solution for the example above would be to just buy the skis in day one, with a total cost of \$4 - that is, if we knew that $l = 5$ beforehand. Therefore, what we've described is the offline optimal solution to this instance of the problem.

Definition 1.1. The *offline optimal solution* (also referred to as "OPT") is the optimal solution to an instance of a problem (that is, the one that maximizes profit or minimizes cost) considering that all information related to the instance is known by the algorithm before its execution.

For an instance $I = (b, l)$ of the Ski Rental Problem, we can formulate the offline optimal solution OPT(I) as:

$$\text{OPT}(I) = \begin{cases} b & \text{if } l \geq b, \\ l & \text{if } l < b \end{cases}$$

Note that the equation above can be summarized as OPT$(I) = \min(b, l)$; in simpler terms, the optimal solution is to rent every day if the cost of buying skis is bigger than the duration of the season, and to just buy the skis if it's cheaper than the cost of renting every day of the season.

Though the OPT(I) is unattainable for an online algorithm which does not have the luxury of perfect information (in this case, knowing the value of l), we can still use the OPT(I) as a measure for the performance of a given online algorithm. More specifically, we'll use it to define an algorithm's competitive ratio.

Definition 1.2. An algorithm ALGO is α-competitive if, for all instances I of a problem:

$$\text{ALGO}(I) \leq \alpha \cdot \text{OPT}(I)$$

Note that $\alpha = 1$ means the offline algorithm is equal to the offline optimal solution. Considering that, we'd like algorithms that are *at least* 2- or 3-competitive - that is, their worst case cost is no bigger than two or three times the optimal cost.

Observation 1.3. *The difference between the cost of an algorithm and the offline optimal solution for a given instance of a problem is equal to the value of perfect information (a concept seen more frequently in decision theory) for that instance. For example, if ALGO(I) = \$10 and OPT(I) = \$5, the value of perfect information would be \$5 - meaning that it'd be worth to pay maximum \$5 to buy the information we don't have and get a better performance than with our online algorithm.*

In order to devise an online algorithm for the Ski Rental Problem, we can consider two types of algorithms: deterministic and randomized ones.

2 Deterministic algorithms

A deterministic algorithm is an algorithm that always produces the same output for a given input.

As an example, consider an instance of the Ski Rental Problem with $b = 10$. Suppose we devise a simple algorithm which rents skis until the day $t = 9$ and then buys them on day $t = 10$. The worst value of l for this instance would be $l = 10$, since that would mean the algorithm would buy the skis on the last day of the season after renting all the previous days.

This algorithm, although extremely simple, is already a 2-competitive algorithm for the Ski Rental Problem!

Lemma 2.1. *An algorithm for the Ski Rental Problem that purchases the skis on the day $t = b$ is 2-competitive.*

Proof. Let ALGO be the algorithm that buys the skis on the day $t = b$. The resulting cost can be defined as below:

$$\text{ALGO}(I) = \begin{cases} l & \text{if } l < b, \\ (b-1) + b = 2b - 1 & \text{if } l \geq b \end{cases} \tag{1}$$

The offline optimal solution for the Ski Rental Problem, as defined above, is $\text{OPT}(I) = \min(b, l)$.

If $l < b$, $\text{OPT}(I) = l$ and $\text{ALGO}(I) = l$; thus, $\text{OPT}(I) = \text{ALGO}(I)$ and by extension $\text{ALGO}(I) \leq 2 \cdot \text{OPT}(I)$.

If $l \geq b$, OPT$(I) = b$ and ALGO$(I) = 2b - 1$; thus:

$$\text{ALGO}(I) = 2b - 1 \leq 2b = 2 \cdot \text{OPT}(I) \tag{2}$$

In conclusion, regardless of the values of the b and l instance parameters, ALGO$(I) \leq$ OPT(I). Therefore, ALGO(I) is 2-competitive. □

Observation 2.2. *There are other measures of algorithm competitiveness besides the competitive ratio. One possibility would be to compare average (and not worst case) performance of different algorithms, or try using the distribution of instance cases to make broader case comparisons. Since this adds uncertainty (for example, by making hypotheses on the form of the probability density function), we are not considering such types of analysis in this case.*

3 Lower bounds for deterministic algorithms

We can show that deterministic algorithms are limited in terms of the competitive ratio they can obtain in the case of the Ski Rental Problem.

Lemma 3.1. *For any deterministic algorithm ALGO, there exists an instance of the Ski Rental Problem for which the below equation is true:*

$$\textit{ALGO}(I) \geq 2 \cdot \textit{OPT}(I) - 1$$

Note that Lemma 3.1 can be understood as the lower bound for the competitive ratio of any deterministic algorithm being approximately 2.

Proof. Let A be a deterministic algorithm for the Ski Rental Problem. Assume a fixed value for b; the algorithm will buy skis on a fixed day $\bar{t}$. The cost of the algorithm can be defined as below:

$$A(I) = \begin{cases} l & \text{if } \ l < \bar{t}, \\ (\bar{t} - 1) + b & \text{if } \ l \geq \bar{t} \end{cases} \tag{3}$$

Note that the worst instance for this algorithm is the one where $l^* = \bar{t}$ - that is, ski season ends the day after A buys the skis.

We know from Section 1 that OPT$(b, \bar{t}) = \min(b, \bar{t})$ and from (3) that $A(b, \bar{t}) = (\bar{t} - 1) + b$. We then have two possible cases for this instance:

1. If $b \geq \bar{t}$:
$$A(b, \bar{t}) = (\bar{t} - 1) + b \geq 2\bar{t} - 1 = 2 \cdot \text{OPT}(b, \bar{t}) - 1$$
2. If $b < \bar{t}$:
$$A(b, \bar{t}) = (\bar{t} - 1) + b \geq 2b - 1 = 2 \cdot \text{OPT}(b, \bar{t}) - 1$$

In conclusion, $A(b, \bar{t}) \geq 2 \cdot \text{OPT}(b, \bar{t}) - 1$. □

4 Randomized algorithms

Now, we'll consider randomized algorithms: by using some degree of randomness in their execution, these algorithms can generate different outputs to the same given input.

Randomized algorithms are useful since they can assume the behavior of several different deterministic algorithms at once. While it's easy to construct a worst case instance for a single deterministic algorithm, it's much harder to do so for a randomized one.

Definition 4.1. The cost of a randomized algorithm is obtained by calculating the expected value of all possible cases.

To better understand randomized algorithms, we'll use an example. Let R be an algorithm that flips a coin before starting its execution.

1. If the algorithm get heads, it buys the skis on day 1.
2. If the algorithm gets tails, it buys on day $3b$.

Now suppose an instance I with $b = 10$ and $l = 40$. We can define $R(I)$ as:

$$R(I) = \frac{1}{2} \cdot b + \frac{1}{2}(3b - 1 + b) = \frac{10}{2} + \frac{39}{2} = \frac{49}{2} \approx 25$$

Note that R is not 2-competitive, since we know that $\text{OPT}(I) = 10$.

Nevertheless, we can make randomized algorithms with a better competitive ratio than 2 - that is, better than any deterministic algorithm.

Theorem 4.2. *The optimal algorithm for the Ski Rental Problem is $\frac{e}{e-1}$-competitive. [1]*

Proof. Let A be an algorithm that has non-zero probability p_t of buying on each day of the season. We can calculate the theoretical expected cost of A and choose the optimal values for each value of p_t. (For the full proof, see [1].) □

It can also be shown, through Yao's minimax principle, that there is no algorithm for the Ski Rental Problem that can achieve a better competitive ratio than $\frac{e}{e-1}$.

5 Game theory perspective of the problem

One way to think about the Ski Rental Problem is by framing it as a game between the algorithms and an adversary that tries to choose the worst instance for each of the algorithms.

We'll use a subset of the problem, limiting the instances to those with $b = 2$ and $l \leq 3$ and only considering relevant deterministic algorithms - that is, the algorithms that buy skis on days 1 ($A1$), 2 ($A2$) or 3 ($A3$) and an algorithm that just rents for the entire season ($A0$).

We can create a matrix of competitive ratios out of the problem instances and the algorithms listed above:

	$A1$	$A2$	$A3$	$A0$
$(b = 2, l = 1)$	2	1	1	1
$(b = 2, l = 2)$	1	$\frac{3}{2}$	1	1
$(b = 2, l = 3)$	1	$\frac{3}{2}$	2	$\frac{3}{2}$

Table 2: The competitive ratios of the four relevant algorithms for a subset of possible problem instances.

Looking at Table 2, $A0$ seems to be the best in terms of the instances and compared to the algorithms shown - it's $\frac{3}{2}$-competitive for this subset of the problem.

Now, consider what we said in Section 4: randomized algorithms are powerful because they can assume the behavior of multiple deterministic algorithms. The matrix shown in Table 2 is useful because it tells us which deterministic algorithms have non-overlapping worst instances; these can be used to create a randomized algorithm that has a lower expected cost than any of its "parent" deterministic algorithms and, therefore, a better competitive ratio.

As $A1$ and $A0$ have the lowest competitive ratios, we can use them to create a better randomized algorithm. Consider an algorithm RB that:

1. With $\frac{1}{4} = 25\%$ probability, follows the behavior of $A1$.
2. With $\frac{3}{4} = 75\%$ probability, follows the behavior of $A0$.

By calculating the expected value of RB for each instance, we'll find the following competitive ratios for the subset of the problem instances we're considering:

	RB
$(b = 2, l = 1)$	$\frac{5}{4}$
$(b = 2, l = 2)$	1
$(b = 2, l = 3)$	$\frac{11}{8}$

Table 3: The competitive ratios of the RB algorithm for the subset of problem instances with $(b = 2, l \leq 3)$.

Table 3 shows that the RB randomized algorithm is $\frac{11}{8}$-competitive for this subset of problem instances.

In conclusion, randomized algorithms can be used to diminish the impact worst-case instances can have on the competitive ratio of deterministic algorithms. In a way, they make it more difficult for an hypothetical adversary to accomplish his goal of creating a bad instance for the algorithm.

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*, SODA '90, page 301–309, USA, 1990. Society for Industrial and Applied Mathematics.