4810-1183 Approximation and Online Algorithms with Applications
Lecture Note 7: Online Algorithms for Ski Rental Problem

Ski Rental Problem

At The University of Tokyo, we have a sport facility called "Gotenshita building". When you arrive at the building, you can buy a ticket at the automatic machine. There are several types of tickets, but, for simplicity of this lecture note, let us assume that we only can buy a yearly ticket, which would cost 9,000 yen, and a one-time ticket, which would cost 400 yen.

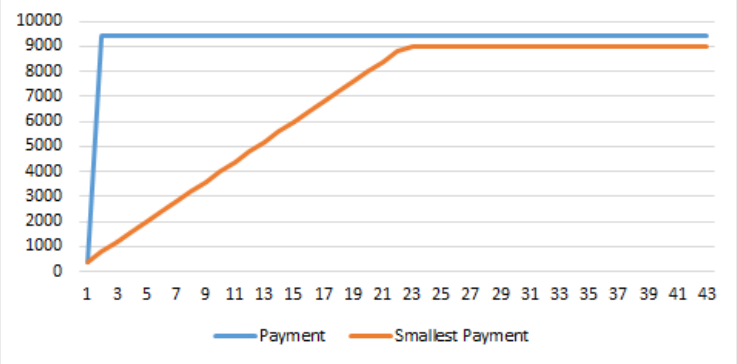
It is not hard to choose the types of ticket, if we know the number of times we go to the building. If we go there for less than 9000/400 ≈ 22.5 times, it is more rational to buy the one-time ticket. Otherwise, we may want to buy the yearly ticket. However, we do not actually know the number of times. Everyone believe that they can play sport regularly, but end up going there just for a few times. We do not know what we will face during a semester.

If we always buy the yearly ticket at the first time, we always have to pay 9,000 yen, no matter how many times we go there. The worst case is when we go only for one time. While we suppose to pay 400 yen, we pay 22.5 times of the optimal payment. We call the gap 22.5 times as competitive ratio. To be more precise,

$$\text{competitive ratio} = \max_{I \in \mathbb{Z}_+} \frac{\text{our payment when we go for } I \text{ days}}{\text{possible smallest payment when we go for } I \text{ days}}.$$

We call a strategy with competitive ratio equals α as α-competitive strategy. So, the strategy of always buying the yearly ticket at the first time is 22.5-competitive.

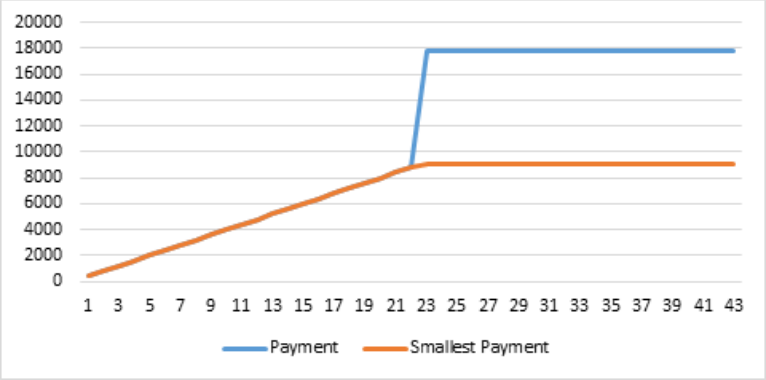
Let now slightly change our strategy. We will buy the one-time ticket on the first time, and, if we go there again, we will buy the yearly ticket. The following graph compares how much we pay and the smallest payment.



The worst case is clearly when the number of days, denoted by I, is 2. Under this strategy, we pay 9,400 yen, while the smallest payment is 800 yen. The competitive ratio is then 11.75.

We can see here that, by slightly changing our strategy, we significantly improve the competitive ratio by almost half. Let us further modify our strategy. We buy the one-time tickets until we are at the building for the 23rd time when we buy the yearly ticket. The comparison between our payment and the smallest payment is now as in the next page. The worst case is now when $I = 30$. We pay 17,800 yen, while the smallest payment is 9,000 yen. The competitive ratio becomes 17,800/9,000 ≈ 1.98.

It is straightforward to show that, for any strategy, we cannot have a smaller competitive ratio than 17,800/9,000. The strategy in the previous paragraph is the best strategy.



Optimization Model in Online Setting

The problem in the previous section, called ski rental problem, can be formally written by an optimization model as:

Input: $g_1, \dots, g_{365} \in \{\text{Yes}, \text{No}\}$
$g_i =$ Yes if we go to the Gotenshita Building at day i

Output: $t_1, \dots, t_{365} \in \{400, 9000, 0\}$

Constraint: 1) When $g_i =$ Yes, $t_i = 0$, only if $t_j = 9000$ for some $j < i$.
2) t_i must be decided based on only $g_1, \dots g_i$

Objective Function: Minimize $\sum_{i=0}^{365} t_i$

We usually call an optimization model with the second constraint as an optimization model in online setting, and we usually call a strategy or an algorithm that is devised for the model as online strategy or online algorithm.

Competitive Ratio

We can consider our payment as a measurement of how "good" our strategy is. Previously, in approximation algorithm, we denote the measurement on solution from our algorithm as SOL, and denote the measurement on the best solution as OPT. We will do the same thing here. For a particular strategy, let us denote our payment on that strategy when we go to Gotenshita for I days as SOL_I, and denote the possible smallest payment as OPT_I. By the notation, we have

$$\text{competive ratio} = \max_{I \in \mathbb{Z}_+} \frac{SOL_I}{OPT_I},$$

$$\text{competive ratio} \geq \frac{SOL_I}{OPT_I} \qquad \text{for all } I \in \mathbb{Z}_+,$$

$$SOL_I \leq (\text{competitive ratio}) \cdot OPT_I \quad \text{for all } I \in \mathbb{Z}_+.$$

From the above inequality, we know that competitive ratio is actually quite similar to the approximation ratio. Competitive ratio is for online algorithms, while the approximation ratio is for approximation algorithms.

In most of the cases, approximation algorithms and online algorithms will not give us OPT, and two of the ratios are not 1. For approximation algorithms, we do not have OPT because the problem is not solvable, but, for online algorithms, it is because we do not have a complete information. While the approximation ratio is the only way to justify an approximation algorithm, the competitive ratio is not the only way to justify an online algorithm. We will introduce other ways in the upcoming lecture notes.

Optimal Number of Operating Servers

From now, we will consider the problem proposed in [2]. We are now operating a number of servers, which will serve a number of demands from users. Servers have operating costs, so we want to have the smallest number of servers operating. The trivial idea is to have the number of operating servers just equal to the demands. However, the authors found that we have to pay for booting a server. When the number of operating servers sharply or frequently increase, the booting cost would be large. Because of that, we may want to have a stable number, although that may increase the operating costs. From the concern, the authors construct the following optimization model

Input:	Demand at each time step $d_1, \dots, d_n \in \mathbb{Z}_{\geq 0}$ Operating cost for one server $c_O \in \mathbb{R}_{\geq 0}$ Booting cost for one server $c_B \in \mathbb{R}_{\geq 0}$
Output:	Number of operating servers $s_1, \dots, s_n \in \mathbb{Z}_{\geq 0}$
Constraint:	1) $s_i \geq d_i$ 2) s_i must be decided based on only $d_1, \dots, d_i$
Objective Function:	Operating cost is $C_O \coloneqq c_O \cdot \sum_{i=1}^{n} s_i$ Booting cost is $C_B \coloneqq c_B \cdot \sum_{s_{i-1}<s_i} \lvert s_i - s_{i-1} \rvert$ Minimize total cost $C_O + C_B$

We have the second constraint because we should not know the future demand. If we do not have the constraint, we can solve this problem. Let us work on an example to understand this optimization in a clearer way.

Example: Suppose that the demands are $d_1 = 5$, $d_2 = 2$, and $d_3 = 3$. The cost for each server $c_0 = 1$ and $c_B = 2$.

If we have the number of operating servers just equal the number required, $s_1 = 5$, $s_2 = 2$, $s_3 = 3$. The operating cost C_O is $c_o \cdot (5 + 2 + 3) = 10$. We have to boot 5 servers at the first time step, and one server at the third. Because of that, the booting cost is $2 \cdot (5 + 1) = 12$. The total cost is 22.

On the other hand, if we have $s_1 = 5$, $s_2 = 3$, $s_3 = 3$. The operating cost C_O would become 11, but, as we do not need to boot the server at the third time step, the booting cost C_B is 10. We have $C_O + C_B = 21$. This output is actually an optimal solution.

❑

Let us try to find an idea for an online algorithm. Consider the situation when we want to calculate the value of s_{i+1} when we have $s_1, \dots, s_i$ and $d_1, \dots, d_{i+1}$. The calculation would be quite easy when $d_{i+1} \geq s_i$. The only choice for that case would be $s_{i+1} = d_{i+1}$. We have to make $s_{i+1} \geq d_{i+1}$, and, when $s_{i+1} > d_{i+1}$, we will spend excessive operating cost and booting cost regardless of what demands are in future.

When $d_{i+1} < s_i$, we have to make a decision. The value of s_{i+1} can be any number between s_i and d_{i+1}. If we have s_{i+1} equal to the smallest choice d_{i+1}, as in the example, if the demand immediately raise at the time step $i + 2$ to $s_i + 1$, we will have to pay for a larger booting cost than any other choice. On the other hand, if we have s_{i+1} equal to the largest choice s_i, we will have to pay a larger operation cost. Especially, if we continue to choose the largest choice for many time steps and the demand keeps smaller than s_i, we might end up paying a large operating cost.

When we decide to have s_i smaller by 1, we may have to pay the booting cost c_B to restart the server that we close in future. For simplicity, let us now assume that we always have to pay to booting

cost. Although the cost is actually paid in future, we will now assume that the cost is immediately paid after s_i is decreased. After we pay for the booting cost, the number of operating servers will be reduced by 1. We will not have to pay for the operating cost c_O. In addition to that, one decrease in the number of servers can cut the operating cost at more than one time steps. The save in operating cost will be until we have to raise the number of servers to s_i.

The discussion in the previous paragraph makes the problem look very similar to the ski rental problem. We have to make a decision whether to pay a large cost for one time c_B or to pay a small cost at every time steps. In the ski rental problem, the large cost is the yearly ticket, and the small cost at every time steps is the one-time ticket.

In the ski rental problem, we will buy the yearly ticket when the amount we pay for the one-time ticket is almost equal to the yearly ticket's price. For example, for our case for Gotenshita building, the yearly ticket costs 9,000 yen, so we will spend 8,800 = 400 x 22 yen on the one-time ticket. By that, we will have the smallest competitive ratio, which is 2.

We will apply the same idea to the problems on the operating servers. We will decrease the number of servers, when the operating cost of abundant servers is almost equal to the booting cost. The algorithm for the problem is as follows:

```
1: For i = 1 to n:
2:    if d[i+1] >= s[i]:
3:       s[i+1] = d[i+1]
4:       count[j] = 0 for all j
5:    else:
6:       count[j] += cb for all d[i+1] <= j < s[i]
7:       count[j] = 0 for all other j
8:       s[i + 1] = j' when j' is smallest j such that count[j] > co
```

`count[j]` will keep track on how much operating cost we spend for not reducing the number of servers to j. We will reduce the number of servers to j when the counting is larger than the booting cost.

Although we will not prove the following theorem in this lecture note as we to take a lot of detail in the proof, one can easily prove it based on the ideas we have discussed so far.

Theorem: The above algorithm is 2-competitive for the optimal number of operating servers problem.

Reference

[1] S. Krumke and C. Thielen, "*Introduction to Online Optimization*", Kaiserslautern, 2014. Chapter 1 http://www.mathematik.uni-kl.de/fileadmin/AGs/opt/Lehre/SS14/Online_SS14/online-optimization.pdf

[2] M. Lin, A. Wierman, L. Andrew, E. Thereska, "*Dynamic right-sizing for power-proportional data centers*", IEEE/ACM Transactions on Networking (TON), Vol. 21, No. 5, pages 1378-1391, 2013.