Approximation and Online Algorithms with Applications

#7

Gotenshita problem (Buy-vs-rent problem)

- You have 2 choices
  - One-time pass (¥300)
  - Yearly pass (¥8,000)
- But, you don't know how much you will go there.
- If we know the # times, n.
  - $n \leq 26$ → one time pass ¥(300·n)
  - $n \geq 27$ → yearly pass ¥8000
- Apply On the first time
  - $n=1$ pay ¥8000 optimal ¥300
  - $n=2$
  - $n=3$
  - ⋮
  - $n \geq 27$ pay

pay $\frac{8000}{300} = 26.67$ times than we should

13.33

$\frac{8000}{99}$

1 time than we shop

↳ competitive ratio

$$\text{Competive ratio} = \max_n \frac{\text{amount we pay}}{\text{smallest amount (when we known)}}$$

$$= 26.67.$$

- Apply on the second time

  - $n=1$ pay ¥300 optimal ¥300 — 1 time.
  - $n=2$ pay ¥8300 optimal ¥600 — 13.8 times.
  - $n=3$ pay ¥8300 optimal ¥900 — 9.22 times.

  ⋮

  - $n \geq 28$ pay ¥8300 optimal ¥8300 — 1 time

Competitive ratio = 13.8

- Apply on the k^{th} time.

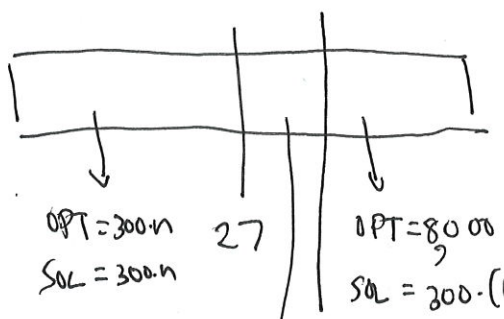
  - $n < k$ pay $300 \cdot n$
  - $n \geq k$ pay $300 \cdot (k-1) + 8000$

~~$k \geq 27$~~ ~~optimal~~

$300 \cdot n$ → optimal

$300 \cdot n$ or 8000

$k \geq 27$



$\frac{SOL}{OPT} = 1$

OPT = 8,000

SOL = 300·n

$\frac{SOL}{OPT} = \frac{n}{26.67} \leq \frac{k-1}{26.67}$

$\frac{SOL}{OPT} = \frac{300(k-1)+8000}{8000}$

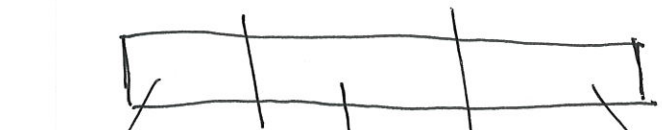
$= \frac{k-1}{26.67} + 1$

Competitive ratio $= \frac{k-1}{26.67} + 1$ 48-1 16229

Smallest when $k=27$. $\rightarrow$ competitive ratio $= \frac{27-1}{26.67} + 1$

≈ 1.97.

$k<27$



OPT = 300n

SOL = 300·n

$\frac{SOL}{OPT} = 1$

OPT = 8,000

SOL = 300(k-1) + 8000

$\frac{SOL}{OPT} = \frac{300(k-1)+8000}{8000}$

$= \frac{k-1}{26.67} + 1$

OPT = 300·n

SOL = 300(k-1) + 8000

$\frac{SOL}{OPT} = \frac{300(k-1)+8000}{300n}$

↑ worse than n is small

Competitive ratio $\frac{300(k-1)+8000}{300k}$

smallest when $k=26$

↓

competitive ratio $= \frac{300 \cdot 25 + 8000}{300 \cdot 26}$

≈ 1.99.

∴ The best strategy is to

subscript the yearly pass at the 27th time.

∴ The amount we have to pay is no more than 2 times

of the optimal amount.

Gotenshita Problem (we go there for n times)

Amount Pay [when we choose to pay at the k^{th} time]

$n < k$ $\quad pay_k(n) = 300 \cdot n$

$n \geq k$ $\quad pay_k(n) = 300 \cdot (k-1) + 8000$

Optimal

$n < 27$ $\quad opt(n) = 300 \cdot n$

$n \geq 27$ $\quad opt(n) = 8,000$

Competitive Ratio

Competive ratio when we pay at the k^{th} time

$$comp_k = \max_n \frac{pay_k(n)}{opt(n)}$$

Shown on last time

$comp_1 = 26.67$

$comp_2 = 13.38$

Our Problem: Find the best strategy → minimize $comp_k$

↓

$$k^* := \arg\min_k comp_k.$$

$$= \arg\min_k \max_n \frac{pay_k(n)}{opt(n)}$$

$$\arg\min_{k<27} \max_{n} \frac{pay_k(n)}{opt(n)} = \arg\min_{k<27} \left\{ \max_{n<k} \frac{pay_k(n)}{opt(n)}, \max_{k\le n<27} \frac{pay_k(n)}{opt(n)}, \max_{n\ge 27} \frac{pay_k(n)}{opt(n)} \right\}$$

$$= \arg\min_{k<27} \max \left\{ \max_{n<k} \frac{300n}{300n}\ (=1), \max_{k\le n<27} \frac{300(k-1)+8000}{300n}, \max_{n\ge 27} \frac{300(k-1)+8000}{8000} \right\}$$

maximize when $n=k$

$$= \arg\min_{k<27} \max \left\{ 1, \frac{300(k-1)+8000}{300k}, \frac{300(k-1)+8000}{8000} \right\}$$

largest

$$= \arg\min_{k<27} \frac{300(k-1)+8000}{300k}$$

$$= \arg\min_{k<27} \frac{300k+7700}{300k}$$

$$= \arg\min_{k<27} \left[1 + \frac{7700}{300k} \right]$$

minimize when $k=26$

$$= 26 \quad \left(\text{competitive ratio} = 1 + \frac{7700}{300 \cdot 26} = 1.99. \right)$$

$$\arg\min_{k\ge 27} \max_{n} \frac{pay_k(n)}{opt(n)} = \arg\min_{k\ge 27} \max \left\{ \max_{n<27} \frac{pay_k(n)}{opt(n)}, \max_{27\le n<k} \frac{pay_k(n)}{opt(n)}, \max_{n\ge k} \frac{pay_k(n)}{opt(n)} \right\}$$

$$= \arg\min_{k\ge 27} \max \left\{ \max_{n<27} \frac{300n}{300n}\ (=1), \max_{27\le n<k} \frac{300n}{8000}\ (k-1), \max_{n\ge k} \frac{300(k-1)+8000}{8000} \right\}$$

$k \ge 28$

$$= \arg\min_{k\ge 27} \max \left\{ 1, \frac{300(k-1)}{8000}, \frac{300(k-1)+8000}{8000} \right\}$$

largest

$$= \max \left\{ 1, \frac{300 \cdot 27}{8000}, \frac{300 \cdot 26 + 8000}{8000} \right\}$$

$$= 27 \quad (\text{Competitive ratio} = 1.98).$$

The best strategy is to buy a yearly pass at the 27th time.

Theorem When the prize of the one-time pass is k_1,

the prize of the yearly pass is k_2.

The best strategy is to buy the yearly pass at the $\left(\frac{k_2}{k_1}\right)^{th}$-time.

The competitive ratio for that case is $\simeq 2$.

Summary

- Easy problem, but we have to make decisions without knowing all inputs.

SOL := objective value obtained from the information that is not completely available.

OPT := objective value obtained from completed information.

- A competitive ratio of an algorithm is α, if

$SOL \leq \alpha \cdot OPT$ ($\alpha \geq 1$) (maximization problem)

$SOL \geq \alpha \cdot OPT$ ($\alpha \leq 1$) (minimization problem).

Secretary Problem

- We want to hire 1 secretary.
- All applicants come to our interviews at different days.
- After the interview, we have to immediately inform the applicants the result.

  (Otherwise, they might be hired by somebody.)
- A rejected applicant will never be hired.
- We want to maximize the possibility of hiring the best applicant.

Note: not maximize the secretary's ability.

[If we have all information, probability that we have the best applicant is 1.]

Assumption

- We have n applicants.
- All applicants come in random order. E.g., when $n=3$,

$$Pr(\#1 \to \#2 \to \#3) = Pr(\#1 \to \#3 \to \#2) = Pr(\#2 \to \#1 \to \#3)$$
$$Pr(\#2 \to \#3 \to \#1) = Pr(\#3 \to \#1 \to \#2) = Pr(\#3 \to \#2 \to \#1)$$
$$= \frac{1}{6}$$

When $n=4$, $Pr(\text{any order}) = \frac{1}{4!} = \frac{1}{24}$.

Algorithm

1: Reject all first $n/e \approx n/2.718 \approx 36\%$ of all applicants.

2: For other applicant, (except the last one)

Accept if he/she is better than all first n/e applicants.

Reject otherwise.

3: If we reach the last applicant, accept him/her.

Theorem Probability that we have the best applicant is $1/e \approx 0.36$. When $n \to \infty$

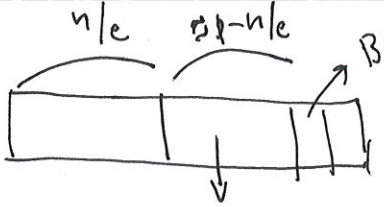
Proof **case 1** The best applicant is among the first n/e applicants.

↓

We select the last candidate. [not the best one].

Probability $= 1/e \approx 0.36$

Case 2



Same one better than first n/e candidates.

Best candidate among the first p-1 applicants.

is not among the first n/e.

Probability $\left(\frac{1}{n}\right) \cdot \frac{p - n/e}{p} = \left[\frac{1}{n} - \frac{n}{ep} \cdot \frac{1}{n}\right]$

$= \left[\frac{1}{n} - \frac{1}{ep}\right]$

Prob. that best candidate position = p

All probabilities $\simeq \int_{n/e}^{n} \left[\frac{1}{n} - \frac{1}{ep}\right] dp$

$= \left[\frac{p}{n} - \frac{1}{e} \ln p\right]_{p=n/e}^{p=n}$

$= \left(\frac{n}{n} - \frac{1}{e} \ln n\right) - \left(\frac{n/e}{n} - \frac{1}{e} \ln n/e\right)$

$= 1 - \frac{1}{e} - \frac{1}{e} \ln n + \frac{1}{e} \ln n - \frac{1}{e} \ln e$

$= 1 - \frac{2}{e}$

prob. of case 1 + case 2 $= \frac{1}{e} + 1 - \frac{2}{e} = 1 - \frac{1}{e}$

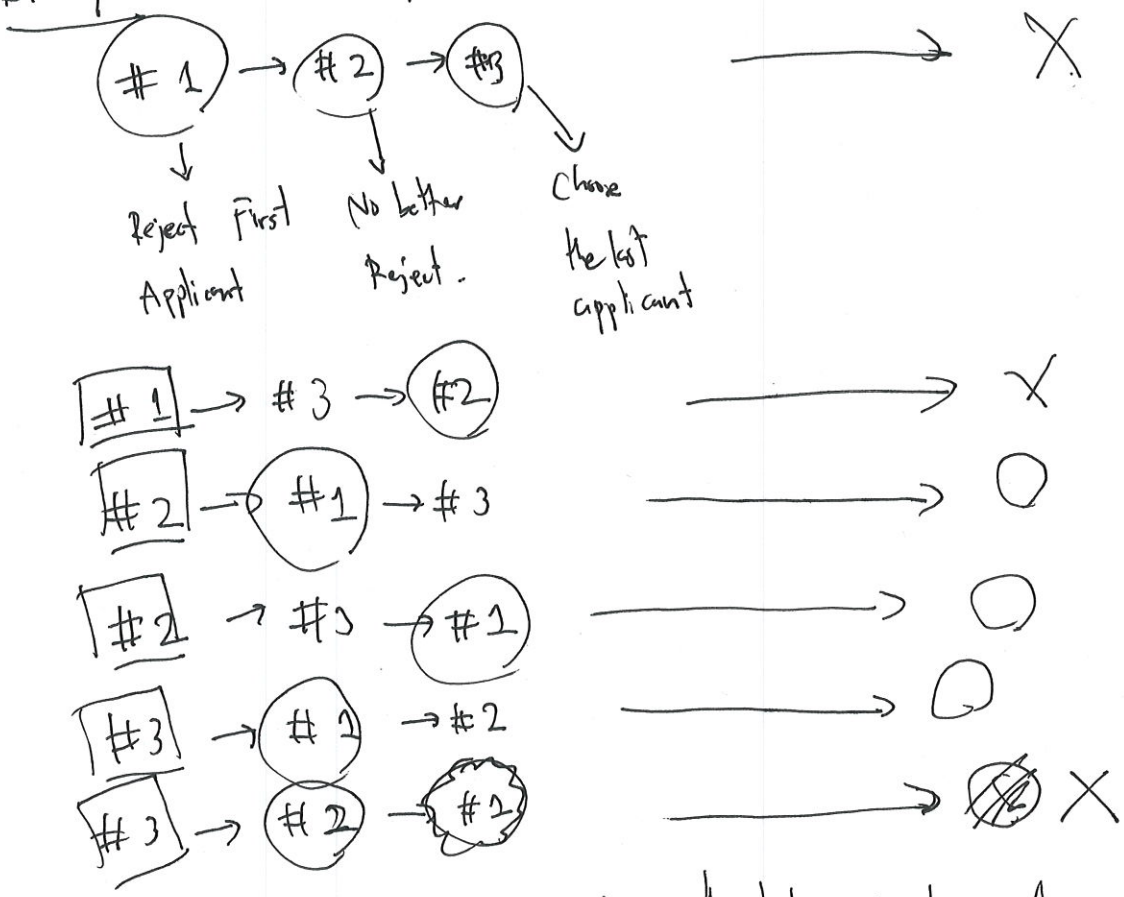
prob. of having the best candidate

= prob. of not case 1 or case 2

$= 1 - \left(1 - \frac{1}{e}\right) = \frac{1}{e}$

VG

Example $n=3$ $n/e \simeq 3/2.71r \approx 1$.



Probability that we have the best candidate $= \frac{1}{2.} = 0.5$.

(Random choose → Prob. $= \frac{1}{3} \simeq 0.33$).

We use the information from the first candidate to help us increasing the probability.

Note: If we allow randomness, we will have $\frac{e}{e-1}$ - competitive algorithm. ($e-1 \simeq 1.58$)

for ski rental problem.

[Instead of 2 - competitive algorithm.]

[Algorithm with competitive ratio $e/(e-1)$]

How many servers we should operate in our data centers?

[Lin, Wierman, Andrew, Thereska, Trans of Networking '14]

Caltech Swinburne Microsoft Research.

Input:
- Operating time : $1, \dots, T$
- # Requests to our data center : $r_1, \dots, r_T \in \mathbb{Z}_+$

$r_1 = 1$ (1 million request at time 1).

Output:
- # Servers we operate : $x_1, \dots, x_T \in \mathbb{Z}_+$

Constraint: $x_i \geq r_i$ for all $1 \leq i \leq T$.

Objective Function:

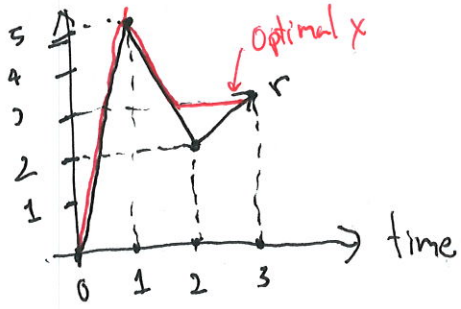
Operating Cost : $cost_0(t) = x_t$

Booting Cost : $cost_B(t) = \begin{cases} \beta(x_t - x_{t-1}) & \text{if } x_t > x_{t-1} \\ 0 & \text{otherwise} \end{cases}$

Cost at time t : $cost(t) = cost_0(t) + cost_B(t)$

Minimize whole cost $\sum_{t=1}^{T} cost(t)$.

Example $n=3$ $r_1=5$ $r_2=2$ $r_3=3$ $\beta=2$



When $x_1=5$ $x_2=2$ $x_3=3$

$cost_0(1)=5$

$cost_B(1)=2\cdot(5-0)=10$

$cost(1)=15$.

$cost_0(2)=2$

$cost_B(2)=0$

$cost(2)=2+0=2$

$cost_0(3)=3$

$cost_B(3)=2\cdot(3-2)=2$

$cost(3)=cost_0(3)+cost_B(3)=5$.

$cost=15+2+5=22$.

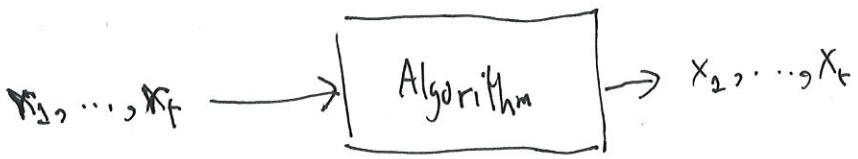
When $x_1=5$ $x_2=3$ $x_3=3$,

$cost(1)=15$ $cost(2)=3$ $cost(3)=3$

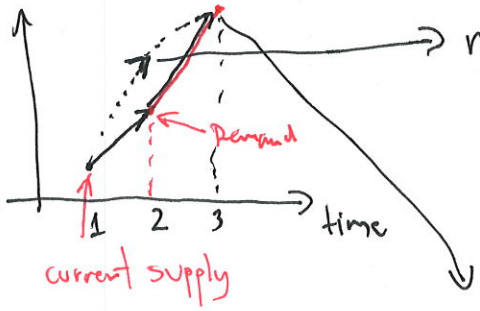
$cost=15+3+3=21$.

Optimal solution!!!

Online Setting: We have to decide x_t just after we know r_t



Situation 1

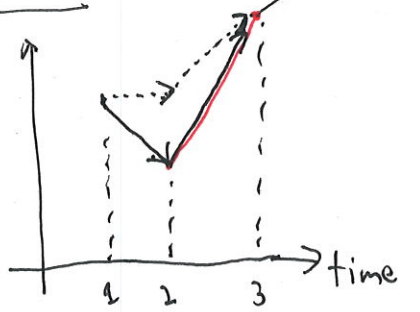


→ not profitable because
- more operation cost
- more booting cost

→
- same booting cost
- more operation cost in the middle.

∴ If $r_i \geq x_{i-1}$, then x_i should be r_i

Situation 2



→
- ~~less~~ more operation cost in the middle.
- less booting cost

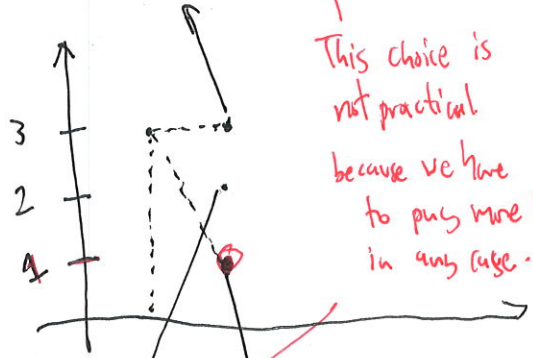
(r_3)

[If we know the demand at time 3, we can choose the best way to go.
But, in online setting, we have to decide x_2 before we know r_3]

General Case

- ~~2~~ 2 operating cost
- 0 booting cost

This choice is not practical because we have to pay more here in any case.



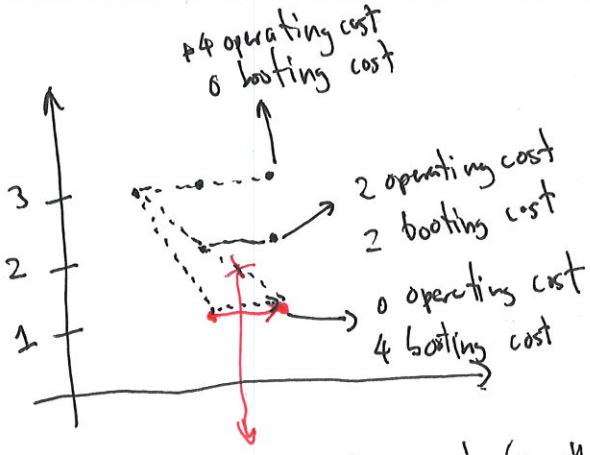
- 1 operating cost
- ~~2~~ 1.5 booting cost (may be)

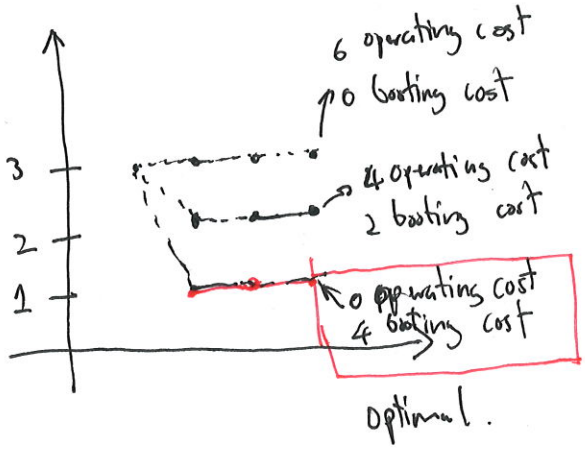
- ~~0~~ 0 operating cost
- ~~4~~ 1 booting cost (may be)

$\beta = 2$.

$\beta = 0.5$.

- When $\beta < 1$, we should have $x_t = r_t$





optimal.

- more operating cost (in the middle)
- same booting cost (in case there is

Booting Cost → fixed for a long period. → yearly pass to Gotenshita

Operation Cost → larger for a longer period → one-time pass to Gotenshita.

Things learnt from Gotenshita problem.

Theorem If is ~~2-competitive~~ 2-competitive to buy yearly pass when we pay for the one-time pass in almost the same amount as the yearly pass.

Algorithm (Suppose that we decided $x_1, \dots, x_{t-1}$ and we know $r_1, \dots, r_t$)

1: If $r_t \geq x_{t-1}$: $x_t = r_t$

2: ~~If~~ If $r_t < x_{t-1}$ and $\beta < 1$: $x_t = r_t$

3: If $r_t < x_{t-1}$ and $\beta \geq 1$:

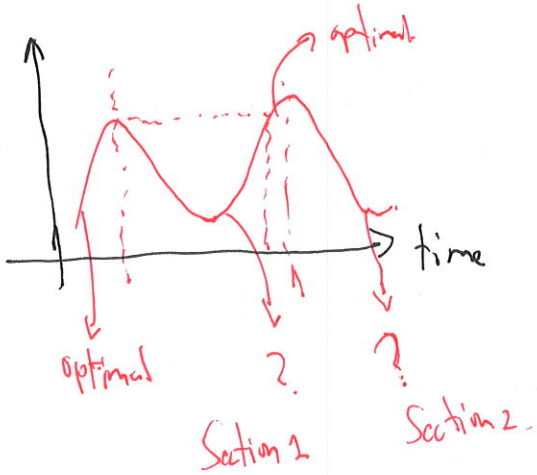
~~If~~ While operation cost saved by reducing x_t to x_{t-j} from the first step possible is more than β

$x_t := x_{t-j}$ [If there are many of such j's, choose largest.]

Theorem Our algorithm is 2-competitive.

Proof

(Outline)



SOL = Fundamental Cost + Additional Operation cost
+ Additional Boting cost

= F + O + B

OPT = F + $(O^* + B^*)$ → Optimal Additional cost

$O + B \le 2 \cdot (O^* + B^*)$

$F + O + B \le F + 2 \cdot (O^* + B^*) \le 2F + 2(O + B^*)$

$= 2(F + O^* + B^*)$ → OPT

↓ SOL

$SOL \le 2 \cdot OPT$ ∎

Theorem It is impossible to have 1.999-competitive algorithm.

↓

(because of the short in information)