Lecture 24: Online Algorithms

Michael Dinitz

November 18, 2021
601.433/633 Introduction to Algorithms

Introduction

Class until now: difficulty was *computational power*

Today: difficulty is *lack of information*

Online:

- Input / data arrives *over time*
- Need to make decisions without knowing future

Ski Rental Problem

Want to go skiing, but don't know how many times you'll be able to go this year.

Should you rent or buy?

- Renting skis: \$50
- Buying skis: \$500
- Every day you ski and haven't yet bought, need to decide: rent or buy?

Buy right away:

- If you only ski once, should have rented (\$50), instead bought (\$500)

Never buy:

- What if you ski $\mathbf{M} \approx \infty$ times?
- Should have bought (\$500), instead rented ($\mathbf{M} \cdot \$50$)

What's the right strategy (for these costs)?

Better Late Than Never

Rent until you realize you should have bought!

BLTN: Rent $\mathbf{9}$ times, buy on $\mathbf{10}$'th.

If ski $\leq \mathbf{9}$ times: optimal

If ski $\geq \mathbf{10}$ times:

- $\mathbf{ALG = 450 + 500 = 950}$
- $\mathbf{OPT = 500}$

Never more than twice (actually $\frac{\mathbf{19}}{\mathbf{10}}$ times) what we should have done!

Competitive Ratio

Definition

The *competitive ratio* of algorithm **ALG** is the maximum over all inputs/futures $\boldsymbol{\sigma}$ of

$$\frac{\mathbf{ALG}(\sigma)}{\mathbf{OPT}(\sigma)},$$

where $\mathbf{ALG}(\boldsymbol{\sigma})$ is the cost of **ALG** on $\boldsymbol{\sigma}$ and $\mathbf{OPT}(\boldsymbol{\sigma})$ is the optimal cost for $\boldsymbol{\sigma}$ (knowing the future).

So on ski rental problem with previous values, competitive ratio is $\frac{19}{10}$.

Ski Rental: Generalized

$\mathbf{\$r}$ to rent, $\mathbf{\$p}$ to buy. Assume $\mathbf{r}$ divides $\mathbf{p}$ for simplicity.

BLTN: Rent $\frac{\mathbf{p}}{\mathbf{r}} - \mathbf{1}$ times, then buy.

Theorem

BLTN has competitive ratio at most $\mathbf{2} - \frac{\mathbf{r}}{\mathbf{p}}$.

Case 1: Ski $\mathbf{z} \leq \frac{\mathbf{p}}{\mathbf{r}} - \mathbf{1}$ times

- $\mathbf{ALG} = \mathbf{z} \cdot \mathbf{r}$
- $\mathbf{OPT} = \min(\mathbf{z} \cdot \mathbf{r}, \mathbf{p}) = \mathbf{z} \cdot \mathbf{r}$

$\implies \frac{\mathbf{ALG}}{\mathbf{OPT}} = \mathbf{1}$

Case 2: Ski $\mathbf{z} \geq \frac{\mathbf{p}}{\mathbf{r}}$ times

- $\mathbf{ALG} = \mathbf{r} \cdot \left(\frac{\mathbf{p}}{\mathbf{r}} - 1\right) + \mathbf{p} = \mathbf{p} - \mathbf{r} + \mathbf{p} = \mathbf{2p} - \mathbf{r}$
- $\mathbf{OPT} = \min(\mathbf{r} \cdot \mathbf{z}, \mathbf{p}) = \mathbf{p}$

$\implies \frac{\mathbf{ALG}}{\mathbf{OPT}} = \frac{\mathbf{2p-r}}{\mathbf{p}} = \mathbf{2} - \frac{\mathbf{r}}{\mathbf{p}}$

So for all inputs / futures, $\frac{\mathbf{ALG}}{\mathbf{OPT}} \leq \mathbf{2} - \frac{\mathbf{r}}{\mathbf{p}}$

Lower Bound

Theorem

No (deterministic) algorithm has competitive ratio better than BLTN.

Deterministic **ALG**: "ski $\mathbf{x}$ times, then buy".
Input: ski $\mathbf{x+1}$ times.

Case 1: $x \geq p/r$

- $\mathbf{OPT} = \min(p, (x+1)r) = p$
- $\mathbf{ALG} = xr + p \geq 2p$

$\implies \frac{\mathbf{ALG}}{\mathbf{OPT}} \geq 2 > 2 - \frac{r}{p}$

Case 2: $x \leq \frac{p}{r} - 1$

- $\mathbf{OPT} = \min(p, (x+1)r) = (x+1)r$
- $\mathbf{ALG} = xr + p$

$$\frac{\mathbf{ALG}}{\mathbf{OPT}} = \frac{xr+p}{(x+1)r} = \frac{xr+p}{xr+r} = 1 + \frac{p-r}{xr+r}$$
$$\geq 1 + \frac{p-r}{(\frac{p}{r}-1)r + r} = 1 + \frac{p-r}{p} = 2 - \frac{r}{p}$$

Elevator Problem

Trying to get up a building: takes $\mathbf{E}$ seconds by elevator, $\mathbf{S}$ seconds by stairs.

- How long should we wait for the elevator?
- Example: $\mathbf{E = 15, S = 45}$.

BLTN: Wait $\mathbf{S - E}$ seconds, then give up and take stairs

If elevator arrives at $\mathbf{x \le S - E}$:

- $\mathbf{OPT = \min(S, x + E) = x + E}$
- $\mathbf{ALG = x + E}$

$\Longrightarrow \quad \mathbf{\frac{ALG}{OPT} = 1}$

If elevator arrives at $\mathbf{x > S - E}$:

- $\mathbf{OPT = \min(S, x + E) = S}$
- $\mathbf{ALG = (S - E) + S = 2S - E}$

$\Longrightarrow \quad \mathbf{\frac{ALG}{OPT} = \frac{2S-E}{S} = 2 - \frac{E}{S}}$

Paging

Classical problem in computer systems/theory

- Disk (slow) with $\mathbf{N}$ pages
- Memory (fast) with room for $\mathbf{k} < \mathbf{N}$ pages
- If OS/application requests a page not in memory: "page fault"
  - Need to bring requested page into memory, *evict* a page from memory (if currently full)
- **Question:** What to evict?

Example: $\mathbf{k} = \mathbf{3}$. Requests: $\mathbf{1, 2, 3, 2, 4, 3, 4, 1, 2, 3, 4}$

(Convention: initial page faults to fill table don't count: only pay when we *evict* a page)

LRU

Standard algorithm: "Least Recently Used" (LRU)

- Evict page from memory that hasn't been used in the longest time
- Intuition:
  - Want to evict page that's next used furthest in the future. But don't know future!
  - Hope that since it hasn't been used for a long time, won't be requested again for a long time.

Is this a good algorithm? What's the competitive ratio? Cost = # evictions.

- $\mathbf{k} = \mathbf{3}$, $\mathbf{N} = \mathbf{4}$
- Requests: $\mathbf{1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, \dots}$

So LRU has competitive ratio $\approx \mathbf{k}$

- LRU evicts every time, **OPT** evicts $\mathbf{1}$ out of every $\mathbf{k}$ times.

Lower Bound

Theorem

No deterministic algorithm has competitive ratio less than $\mathbf{k}$.

Let **ALG** be some deterministic algorithm. Set $\mathbf{N} = \mathbf{k} + \mathbf{1}$
Request sequence: Whatever is not in memory for **ALG**!

$\implies$ **ALG** has an eviction every time (after initialization)

OPT: evict page whose next request is furthest in the future

- Every page in memory needs to be requested before next eviction. So next eviction is in at least $\mathbf{k}$ steps.

$$\implies \frac{\mathbf{ALG}}{\mathbf{OPT}} \geq \mathbf{k}$$

Marking Algorithm

Get around lower bound by using *randomization*

- Lower bound argument doesn't apply because can't set request sequence to ask for whatever's not in memory, since that involved randomness! (Oblivious adversary)

Assume memory initially $1, 2, \ldots, k$.
Set all pages in memory to be "unmarked"

When page requested:

- If already in memory, "mark" it
- If not in memory:
  - If all pages in memory "marked", unmark all
  - Choose an *unmarked* page uniformly at random to evict
  - Bring in new page, mark it

Marking Analysis

Theorem

Expected competitive ratio at most $\mathbf{O(\log k)}$*:*

$$\frac{\mathbf{E[ALG(\sigma)]}}{\mathbf{OPT(\sigma)}} \leq \mathbf{O(\log k)} \text{ for all request sequences } \boldsymbol{\sigma}.$$

Proof sketch for $\mathbf{N = k + 1}$: full generality more complicated

Phase: time between "unmark all" events.

In each phase:

- $\mathbf{OPT \geq 1}$, since all $\mathbf{N}$ pages requested

ALG in each phase

Key point: the one page *not* in memory is *uniformly distributed* among all *unmarked* pages.

When page requested:

- If marked: in memory, no eviction
- If unmarked: if currently $\mathbf{i}$ unmarked pages, then $\mathbf{Pr}[\text{eviction}] = \mathbf{Pr}[\text{requested page not in memory}] = \mathbf{1/i}$
  - Becomes marked

At beginning of phase $\mathbf{i} = \mathbf{N}$, at end of phase $\mathbf{i} = \mathbf{1}$. Goes down by one every time page gets marked.

$\Longrightarrow$ expected cost in phase at most $\frac{\mathbf{1}}{\mathbf{N}} + \frac{\mathbf{1}}{\mathbf{N-1}} + \frac{\mathbf{1}}{\mathbf{N-2}} + \cdots + \frac{\mathbf{1}}{\mathbf{2}} + \mathbf{1} = \mathbf{O(\log N)} = \mathbf{O(\log k)}$