Recitation 3: Competitive Analysis

Setup

input : sequence of requests $R = r_1, r_2, \ldots, r_k$

online algorithm → processes requests one at a time w/o knowledge of subsequent requests "myopic"

optimal offline algorithm (OPT) → knows entire sequence from the get-go and processes each request optimally w/ this knowledge

Competitive analysis

algorithm A is α-competitive if for any input R

$$\underbrace{C_A(R)}_{\text{cost of } A \text{ on } R} \le \alpha \underbrace{C_{OPT}(R)}_{\text{cost of OPT on } R}$$

competitive ratio
(we want α to be small!)

illustrating example: rent or buy?

context : renting ski gear r = \$50 per session ($b = 10r$)
buying ski gear b = \$500

ski $\le$ 10 times → rent gear
ski $\ge$ 10 times → buy gear

but: not sure how many times I'll ski!
do I rent or buy my gear?

strategy 1 : buy at start → terrible if I ski only once (renting is waaaaay better) $\alpha = 10$

strategy 2 : always rent! → bad if I end up skiing a TON ($k \gg 10$)

Strategy 3: rent first $\lceil b/r \rceil - 1 = 9$ times and buy on the next visit after that ("better-late-than-never" strategy)

Claim: this strategy is 2-competitive!

- if I end up skiing $k \le \lceil b/r \rceil - 1 = 9$ times, then I am optimal — woohoo!
- if $k > \lceil b/r \rceil - 1 = 9$, then I should have bought gear right at the start; OPT $= b$.

worst case: I buy on my $\lceil b/r \rceil$th visit and I never ski again

$$\alpha = \frac{r(\lceil b/r \rceil - 1) + b}{r} \quad \begin{matrix} \leftarrow \text{my cost} \\ \leftarrow \text{OPT cost} \end{matrix}$$

< 2 (given b is multiple of r)

claim: this strategy gives the best possible competitive ratio (of all deterministic algorithms)

Worked example: competitive scheduling

Setup: n identical machines $M_1, \dots, M_n$ that can process jobs

input: sequence of jobs $\sigma = J_1, \dots, J_k$ arriving all at once ($t = 0$)

J_i has processing time p_i

goal: schedule jobs on machine s.t. the time at which the last job finishes (i.e., the "makespan") is minimized

Our strategy: always assign incoming job to the least loaded machine

claim: this strategy is 2-competitive

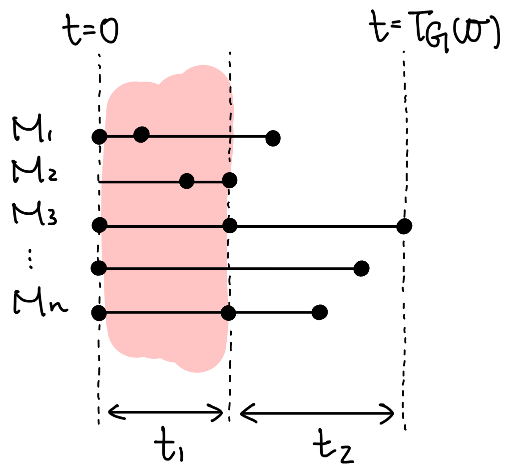
notation:

- $T_G(\sigma)$ makespan of our greedy approach
- $T_{OPT}(\sigma)$ makespan of the optimal schedule
- p_{max} processing time of the longest job
- t_1 time when some machine finishes (i.e., all other machines run for $\ge t_1$ time)

Observations:

(B1) $T_{OPT}(\sigma) \geq p_{max}$ "total time $\geq$ time required for the longest job"

(B2) $T_{OPT}(\sigma) \geq \frac{1}{n}\sum_{i=1}^{k} p_i$ "best possible schedule = distrube work completely evenly across machines"



total "work" over interval $[0, t_1]$:

$$\underbrace{n \cdot t_1}_{\text{total work up to } t_1} \leq \underbrace{\sum_{i=1}^{k} p_i}_{\text{total work over all time}}$$

and so $t_1 \leq \frac{1}{n}\sum_{i=1}^{k} p_i$

Let

- $t_2 = T_G(\sigma) - t_1$ "the amount of extra work done by the busiest machine compared to the least busy one"
- t_L: start time of LAST job

<u>claim</u>: $t_L \leq t_1$

by contradiction: suppose $t_L > t_1$.

$\Longrightarrow$ greedy approach assigns LAST to least busy machine
all machines busy until t_1

violates definition of t_1 ↯

putting it all together:

$$P_{LAST} = T_G(\sigma) - t_L \underset{\text{by claim}}{\geq} T_G(\sigma) - t_1 = t_2$$

since $P_{LAST} \leq p_{max}$, $t_2 \leq p_{max}$

$$T_G(\sigma) = t_1 + t_2 \underset{\text{"total work over } [0,t_1]\text{"}}{\leq} \underbrace{\frac{1}{n}\sum_{i=1}^{k} p_i}_{\leq T_{OPT}(\sigma)\ (B1)} + \underbrace{p_{max}}_{\leq T_{OPT}(\sigma)\ (B2)} \leq 2T_{OPT}(\sigma)$$

as desired. ▨

Worked example: LRU paging

least-recently-used (LRU) : cache w/ k slots stores the most recently requested k pages

hit if user requests for a page in the cache, **fault**, otherwise

claim: LRU is k-competitive
(i.e., OPT faults ≥ 1 times every time LRU faults k times)

Key insight: (pigeonhole principle)

LRU has k faults
⟹ ≥ k+1 distinct pages requested
⟹ OPT has at least 1 fault