Exercise: Packing and Paging

1 Missing cow

Consider you are a cow getting lost on a misty grassland. Nearby there is a very long fence. You know that somewhere at the fence, there is a hole so you can escape. However, you don't know where the hole is nor which direction the hole is in. Furthermore, because it is so foggy, you see the hole only when it is exactly in front of you. In such a situation, all you can do is walk straightly toward one direction and search for the hole, or turn around, walk toward the other direction, and search for the hole. You want to escape using as small a total walking distance as possible.

Answer the following questions:

1. Let your starting position be the origin and the fence is the x-axis. Assume that the hole is at position n (where n can be positive or negative), what is the optimal solution cost?

   The optimal solution knows where the hole is. It can walk directly toward the whole and the cost is $|n|$.

2. Consider the following algorithm ALG_1 that first go to position 1, then go to position -1, and then 2, -2, 4, -4, ... (Figure 1):

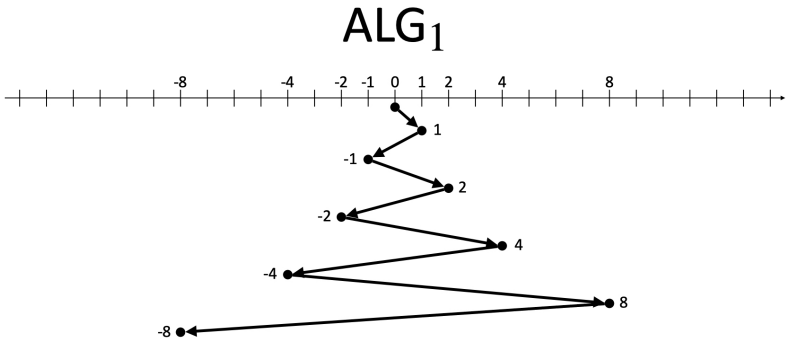


Figure 1: Cow path for ALG_1

   (a) Find an adversary for ALG_1 on the *right* hand side of the cow and give a lower bound of ALG_1's competitive ratio.

   Let $n \in (2^k, 2^{k+1}]$. The cost of ALG_1

$$
\begin{aligned}
&= 1+1+1+1+2+2+2+2+4+4+4+4+\cdots+2^k+2^k+2^k+2^k+|n| \\
&> 4 \cdot \Sigma_{i=0}^{k} 2^i + 2^k \\
&\approx 4 \cdot 2^{k+1} + 2^k \\
&= 9 \cdot 2^k
\end{aligned}
$$

Hence, $\frac{\text{ALG}_1(n)}{\text{OPT}(n)} \geq \frac{9 \cdot 2^k}{n} > \frac{9 \cdot 2^k}{2^k} = 9$, and ALG_1 is at least 9-competitive.

(b) Find an adversary for ALG_1 on the *left* hand side of the cow and give a lower bound of ALG_1's competitive ratio.

Let $n \in (2^k, 2^{k+1}]$. The cost of ALG_1

$$
\begin{aligned}
&= 1 + 1 + 1 + 1 + 2 + 2 + 2 + 2 + 4 + 4 + 4 + 4 + \cdots + 2^k + 2^k + 2^k + 2^k \\
&\quad + 2^{k+1} + 2^{k+1} + |n| \\
&> 4 \cdot \Sigma_{i=0}^{k} 2^i + 2 \cdot 2^{k+1} + 2^k \\
&\approx 4 \cdot 2^{k+1} + 2 \cdot 2^{k+1} + 2^k \\
&= 13 \cdot 2^k
\end{aligned}
$$

Hence, $\frac{\text{ALG}_1(n)}{\text{OPT}(n)} \geq \frac{13 \cdot 2^k}{n} > \frac{13 \cdot 2^k}{2^k} = 13$, and ALG_1 is at least 13-competitive.

(c) From (a) and (b), which adversary gives a stronger lower bound?

(b) is a stronger lower bound as it is bigger.

(d) Prove that ALG_1 is 13-competitive.

For any instance $n \in (2^k, 2^{k+1}]$,
$\text{ALG}_1 = 4 \cdot \Sigma_{i=0}^{k} 2^i + n \leq 4 \cdot 2^{k+1} + n$.
Hence, $\frac{\text{ALG}_1(n)}{\text{OPT}(n)} \leq \frac{8 \cdot 2^k + n}{n}$. The ratio increases as n decreases. However, $|n| \geq 2^k$ and $\frac{\text{ALG}_1(n)}{\text{OPT}(n)} \leq \frac{8 \cdot 2^k + |n|}{|n|} \leq \frac{8 \cdot 2^k + 2^k}{2^k} = 9$.

For any instance $n \in [-2^{k+1}, -2^k)$,
$\text{ALG}_1 = 4 \cdot \Sigma_{i=0}^{k} 2^i + 2 \cdot 2^{k+1} + |n| \leq 4 \cdot 2^{k+1} + 2 \cdot 2^{k+1} + |n|$.
Hence, $\frac{\text{ALG}_1(n)}{\text{OPT}(n)} \leq \frac{12 \cdot 2^k + |n|}{|n|}$. The ratio increases as $|n|$ decreases. However, $|n| \geq 2^k$ and $\frac{\text{ALG}_1(n)}{\text{OPT}(n)} \leq \frac{12 \cdot 2^k + |n|}{|n|} \leq \frac{12 \cdot 2^k + 2^k}{2^k} = 13$.

The competitive ratio of ALG_1 is $\max\{9, 13\} = 13$.

3. Consider the following algorithm ALG_2: First go to 1, then go to -2, 4, -8, 16, $\cdots$ (Figure 2.

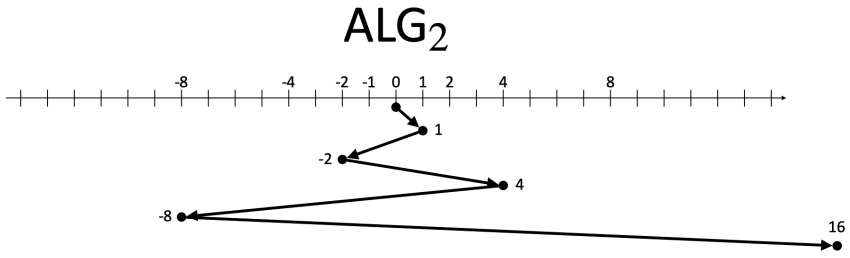


Figure 2: Cow path for ALG_1

(a) By intuition, do you think this ALG_2 is better than ALG_1 or worse?

ALG_2 is better. To get the same position, ALG_2 passes through smaller number of zig-zags compared to ALG_1.

(b) Show that this algorithm is 9-competitive.

For any instance $n \in (2^k, 2^{k+1}]$,
$\mathrm{ALG}_2 = 2 \cdot \Sigma_{i=0}^{k} 2^i + n \leq 2 \cdot 2^{k+1} + n$.
Hence, $\frac{\mathrm{ALG}_2(n)}{\mathrm{OPT}(n)} \leq \frac{4 \cdot 2^k + n}{n}$. The ratio increases as n decreases. However, $|n| \geq 2^k$ and $\frac{\mathrm{ALG}_2(n)}{\mathrm{OPT}(n)} \leq \frac{4 \cdot 2^k + |n|}{|n|} \leq \frac{4 \cdot 2^k + 2^k}{2^k} = 5$.

For any instance $n \in [-2^{k+1}, -2^k)$,
$\mathrm{ALG}_2 = 2 \cdot \Sigma_{i=0}^{k+1} 2^i + |n| \leq 2 \cdot 2^{k+2} + |n|$.
Hence, $\frac{\mathrm{ALG}_2(n)}{\mathrm{OPT}(n)} \leq \frac{8 \cdot 2^k + |n|}{|n|}$. The ratio increases as $|n|$ decreases. However, $|n| \geq 2^k$ and $\frac{\mathrm{ALG}_2(n)}{\mathrm{OPT}(n)} \leq \frac{8 \cdot 2^k + |n|}{|n|} \leq \frac{8 \cdot 2^k + 2^k}{2^k} = 9$.

The competitive ratio of ALG_2 is $\max\{5, 9\} = 9$.

2 Applying doubling technique

Recall the ski rental problem with the buying price of B and the renting price of 1 per day. Use the doubling technique to analyze the optimal online algorithm that buys a pair of skis on the B-th day.

(i) Let the parameter r be B and adapt the analysis for doubling.

(ii) Let the parameter r be 2 and adapt the analysis for doubling. What happens if $B = 2^k$ for some integer k, and what happens if $B \neq 2^k$?

(i) In phase 0 (from day B^0 to day $B^1 - 1$), the algorithm pays $B - 1$ for renting while the optimal solution lower bound increases from 0 to 1. In phase 1 (from day B to day $B^2 - 1$, the algorithm pays B for buying while the optimal solution lower bound increases from 1 to B. For any later phase, both the optimal solution lower bound and the incurred cost of the algorithm are 0. Therefore, the algorithm is $\max\{\frac{B-1}{1}, \frac{B}{B-1}\} = B - 1$ competitive.

(ii) We first assume $B = 2^k$ for some integer k. In phase 0 (where there is exactly one day), the algorithm pays 1 for renting while the optimal solution lower bound increases from 0 to 1. For $i = 1, 2, \cdots, k-1$, the optimal solution lower bound increment is $2^i - 2^{i-1}$ while the algorithm pays 2^{i-1} for renting. In phase k, the increment of the optimal solution lower bound is $2^k - 2^{k-1}$, and the online algorithm pays $B = 2^k$ for buying. Therefore, the competitive ratio is $\max\{1, \frac{2^k}{2^{k-1}}\} = 2$.

However, when $B = 2^k + c$ for some $0 < c < 2^k$, in the k-th phase, the optimal solution lower bound increases by 2^{k-1} while the algorithm pays $(c-1) + B \leq 3 \cdot 2^k - 3$ since $c \leq 2^k - 1$. Therefore, the competitive ratio by this analysis is 6.

3 NextFit algorithm

Recall that we introduced the FirstFit algorithm for the BinPacking problem during the lecture. Now consider another algorithm **NextFit**:

`NextFit` Algorithm

When a new item arrives, put it into the *last* bin if the item fits into that bin.
Otherwise, close the last bin, open a new bin, and put this new item to this new bin.

1. Claim that in the NextFit solution, the total size of jobs in any two consecutive bins is at least 1.

   If there are two consecutive bins together have items with total size less than 1, the items in the second bins can be assigned to the first bin without exceeding the bin capacity. According to the NextFit algorithm, the jobs in the second bin would be put in the first bin. Hence, the case won't happen.

2. Prove that NextFit $\leq 2 \cdot \text{OPT}$.

   Assume that NextFit uses k bins. By (a), any two consecutive bins (but the last one) have total item sizes at least 1. Hence, the total size of all items is at least $\lfloor \frac{k-1}{2} \rfloor$. Moreover, the optimal solution uses at least $\lfloor \frac{k-1}{2} \rfloor$ bins. Hence,

$$\begin{aligned} \lfloor \frac{k-1}{2} \rfloor &\leq \text{OPT} \\ \frac{k-1}{2} &\leq \text{OPT} + 1 \\ k - 1 &\leq 2 \cdot (\text{OPT} + 1) \\ \text{NextFit} = k &\leq 2 \cdot \text{OPT} + 3 \end{aligned}$$

4 Another lower bound of First-Fit

Recall the adversary for First-Fit algorithm introduced in the lecture that contains m items of size $\frac{1}{6} - 2\varepsilon$, m items of size $\frac{1}{3} + \varepsilon$, and m items of size $\frac{1}{2} + \varepsilon$. Note that the largest denominator 6 in this adversary is a *perfect number*. That is, it is equal to the sum of its positive divisors, excluding the number itself. (Ex: 6's divisors are $1, 2, 3, 6$, and $6 = 1 + 2 + 3$.) The fact of 6 being a perfect number gives us the convenience where $\frac{1}{6} + \frac{1}{3} + \frac{1}{2} = \frac{1+2+3}{6} = \frac{6}{6} = 1$, and the adversary is designed based on that.

Consider the next perfect number, 28, whose divisors are $1, 2, 4, 7, 14, 28$, and $1+2+4+7+14 = 28$. What is the best adversary against First-Fit you can come up with? Is it a stronger lower bound than the one we saw in the lecture? Can you see why?

We release m items with size $\frac{1}{28} - 4\varepsilon$, m items with size $\frac{1}{14} + \varepsilon$, m items with size $\frac{1}{7} + \varepsilon$, m items with size $\frac{1}{4} + \varepsilon$, and m items with size $\frac{1}{2} + \varepsilon$ for some very large m. In this order, the First-Fit algorithm will open:

- $\frac{m}{28}$ bins, each contains 28 size-$\frac{1}{28} - 4\varepsilon$ items,
- $\frac{m}{13}$ bins, each contains 13 size-$\frac{1}{14} + \varepsilon$ items,
- $\frac{m}{6}$ bins, each contains 6 size-$\frac{1}{7} + \varepsilon$ items,
- $\frac{m}{3}$ bins, each contains 3 size-$\frac{1}{4} + \varepsilon$ items, and
- m bins, each contains 1 size-$\frac{1}{2} + \varepsilon$ items.

That is, the total number of bins used by First-Fit is $(\frac{1}{28} + \frac{1}{13} + \frac{1}{6} + \frac{1}{3} + 1)m$. Meanwhile, the optimal can pack all items into m bins by putting exactly one item from each of the types in one bin. Therefore, the competitive ratio is at most $\frac{1}{24} + \frac{1}{13} + \frac{1}{6} + \frac{1}{3} + 1 = 1.618589... \leq 1.61859$.

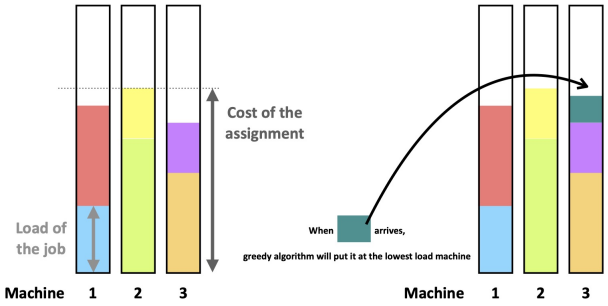
This adversary is weaker than the one introduced in the lecture.

5 Online load balancing

There is a finite set of m machines. A sequence of n jobs is arriving where each job J_i is specified by its processing load ℓ_i. Each job must be assigned to exactly one of the machines upon arrival. The total load of a machine is the sum of loads of the jobs assigned to it. The cost of an assignment is the maximum total load among the machines. The goal of the LoadBalancing problem is to find an assignment with the minimum cost.

Consider the greedy algorithm (also see the illustration):

`Assign each arriving job` r_i `to the machine with the lowest load (breaking ties arbitrarily).`



1. Consider any job J_k. Claim that the cost of the optimal assignment is at least ℓ_k.

   In the optimal solution, there is a machine where job J_k is assigned. Hence, the load (in the optimal solution) of the machine is at least ℓ_k

2. Claim that the cost of the optimal assignment is at least $\frac{\sum_{j=1}^{n} \ell_j}{m}$

   (Hint: Use the similar argument for bin packing optimal cost lower bound.)

   To minimize the highest load, the best thing an algorithm can do is to put the load as even as possible. If there is an algorithm which can place the total load evenly on each of the machines (without worrying about if we have to cut a job into pieces), the cost of this algorithm is

   $$\frac{\text{total load}}{\text{number of machines}} = \frac{\sum_{j=1}^{n} \ell_j}{m}.$$

   The optimal algorithm has to assign the jobs without cutting them, so it cannot do better than this assignment. Hence,

   $$\text{the cost of OPT} \geq \frac{\sum_{j=1}^{n} \ell_j}{m} .$$

3. Show that the greedy algorithm is $2 - \frac{1}{m}$-competitive.

   Assume the highest load machine M_i in the algorithm assignment has load $\ell_k + s$, where ℓ_k is the load of the last job J_k assigned to M_i and $s \geq 0$. Since the algorithm uses a greedy strategy, it always assigns the job to the current lowest-load machine. Hence, when the job J_k arrives, the load of each of the machines is at least s (otherwise, the job J_k will be assigned to another machine with a lower load instead of to the machine M_i). Therefore, the total load of the jobs is at least $(m-1) \cdot s + (\ell_k + s) = m \cdot s + \ell_k$.

   By part (b), the cost of optimal is at least $\frac{\text{total load}}{m} \geq \frac{m \cdot s + \ell_k}{m} = s + \frac{\ell_k}{m}$. Hence, $s \leq \text{OPT} - \frac{\ell_k}{m}$ (where OPT denotes the cost of the optimal solution).

   The cost of the algorithm

   $$\ell_k + s \leq \ell_k + (\text{OPT} - \frac{\ell_k}{m}) \leq \text{OPT} + \ell_k \cdot (1 - \frac{1}{m}) = (2 - \frac{1}{m}) \cdot \text{OPT}$$

   (where the second inequality comes from part (a) that $\text{OPT} \geq \ell_k$).

6 Last-In-First-Out

Consider the algorithm `LIFO` (Last-In-First-Out) for the PAGING problem:

`LIFO` (Last-In-First-Out) Algorithm

When there is a page fault and the cache is full, replace the page that has been copied into the cache the most recently.

Answer the following questions:

1. Given a sequence of n page requests, show that `LIFO` is at most $O(\frac{n}{k})$-competitive.

   In the worst case, `LIFO` incurs one page fault for each of the page requests. And the optimal solution has to copy each of the k pages into the cache. Therefore, for any sequence of requests R, $\frac{\texttt{LIFO}(R)}{\text{OPT}(R)} \leq \frac{n}{k}$.

2. Show that the analysis in (a) is tight.

   Consider the sequence of requests $R = 1, 2, 3, \cdots, k$ followed by n pairs of requests $k+1, k$. An offline algorithm can evict page $k-1$ when the page $k+1$ is requested for the first time and the total number of page faults is $k+1$. However, `FirstFit` replaces page k when the page $k+1$ is requested and replaces page $k+1$ when page k is requested. Hence, it incurs a page fault for every request, and the total number of page faults is $k-1+n$. Therefore, when n is large enough, $\frac{\texttt{LIFO}(R)}{\text{OPT}(R)} \geq \frac{k-1+n}{k+1} \approx \Omega(\frac{n}{k})$.