Advanced algorithms

Exercise sheet #7 (Solutions) — Online algorithms

November 16, 2022

Exercice 1 (Online scheduling)**.** Consider m identical machines and a sequence of n tasks of respective duration $p_1, \dots, p_n$. We must assign each task to a machine, in an online way (the ith task is assigned without knowledge of what comes next, and an assignment is not revocable), aiming at minimizing the termination time of the last task.

Design a greedy algorithm that is $(2 - \frac{1}{m})$-competitive. Recall from the exercise sheet #1 that the optimal scheduling takes a time at least equal to $\max\left(\max_{1 \leqslant i \leqslant n} p_i, \frac{1}{m}\sum_{i=1}^n p_i\right)$.

Solution — When a task arrives, we assign it to the machine with the least load.

Let T_{opt} be the time of the optimal scheduling. Note that $T_{opt} \geqslant \max_i p_i$ and $T_{opt} \geqslant \frac{1}{m}\sum_i p_i$.

Consider the jth step. Before the task is assigned, the machine with the smallest load has smaller load than average, that is at most $\frac{1}{m}\sum_{i<k} p_i$. So after the jth step, the total load T_j of the machine to which the jth task is assigned satisfies

$$T_j \leqslant \frac{1}{m}\sum_{i<j} p_i + p_j = \frac{1}{m}\sum_{i \leqslant j} p_i + \left(1 - \frac{1}{m}\right) p_j \leqslant \left(2 - \frac{1}{m}\right) T_{opt}.$$

To conclude, we note that the final duration of the scheduling is $\max_j T_j$.

Exercice 2 (Memory management, (pâle 2011))**.** We consider a memory model with a cache, which can hold only one page, and a slower external memory. Accessing the page in cache is free, accessing a page in external memory costs 1. After accessing a page in external memory, we can choose to load it in cache, this costs D. We are interested in a good online strategy to address a stream of requests while minimizing the cost.

We study the following algorithm. Each page p has a counter $c(p)$ that is initially set to 0. Then the memory management performs the following loop.

```
1: while true do
2:     $p \leftarrow$ next requested page
3:     access $p$ (cost $\leqslant 1$)
4:     $c(p) \leftarrow c(p) + 1$
5:     if $c(p) = D$ then
6:         load $p$ in cache (cost $\leqslant D$)
7:         while $c(p) > 0$ do
8:             $q \leftarrow$ next requested page
9:             if $q \neq p$ then
```

```
                $c(p) \leftarrow c(p) - 1$
                access $q$ (cost 1)
            else
                access $q$ (cost 0)
            end if
        end while
        (end of phase for $p$)
    end if
end while
```

We consider that there is no concurrency issue: When there is no request to deal with, the memory management system holds. Notice that in the inner **while** loop, all the counters $c(q)$ for $q \neq p$ are locked.

(a) Consider the case $D = 3$ and the requests $s_1, s_2, \dots$ below. Complete the table with the values taken by the counters along the algorithm.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	...
s_i	1	1	4	6	4	1	4	6	1	1	4	1	4	6	1	4	6	6	...
$c(1)$	1	2	2	2	2	3	2												
$c(4)$	0	0	1	1	2	2	2												
$c(6)$	0	0	0	1	1	1	1												

Solution —

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	...
s_i	1	1	4	6	4	1	4	6	1	1	4	1	4	6	1	4	6	6	...
$c(1)$	1	2	2	2	2	3	2	1	1	1	0	1	1	1	1	1	1	1	
$c(4)$	0	0	1	1	2	2	2	2	2	2	2	2	3	2	1	1	0	0	
$c(6)$	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	

To analyze the performance of the algorithm, let's split the stream $s = (s_1, s_2, \dots)$ into subsets of requests called *phases*, related to each reset of a counter: The phase I_j^p countains the indexes i of the requests s_i received between resets j and $j+1$ of the counter $c(p)$, and such that

- Either s_i increments $c(p)$ (Access to p);
- Or, s_i is received after the request that made $c(p)$ reach the limit D for the $(j+1)$-th time.

In the previous example, we have

$$\begin{aligned} I_0^1 &= (1,2,6,7,8,9,10,11), \quad I_1^1 = (12,\dots), \quad \dots \\ I_0^4 &= (3,5,13,14,15,16,17), \quad \dots \\ I_0^6 &= (4,18,\dots), \quad \dots \end{aligned}$$

Notice that two phases corresponding to distinct pages can be interleaved and that the request I_j^p begins with a sequence of requests to page p and ends with a sequence of at least $D+1$ consecutive requests.

(b) Show that the cost of handling the requests of a single phase is at most $3D$.

Solution — The cost of the requests before loading p is 1, and there are at most D of them. The cost of loading p in cache is D. The cost of requests to pages other than p is one, but there are at most D of them before a new phase begins.

Let $\mathcal{A}$ be any other algorithm for handling the given requests.

(c) Given the sequence of requests s, show that we can split the cost paid by $\mathcal{A}$ for each operation (accessing and loading pages) on the different phases such that each phase is assigned a cost at least D.

Hint: Assign the cost paid by $\mathcal{A}$ to load a page that removes q from cache to the phase I_j^q currently open, and analyse the cost paid by $\mathcal{A}$ to handle all other request in each phase.

Solution —

In this question, we want to estimate a lower bound on the total cost paid by $\mathcal{A}$. Since the phases I_j^q form a partition of the stream, it is enough to virtually assign part of the cost paid by $\mathcal{A}$ to the different phases.

Consider the following procedure:

Upon receiving a request to page p:

- *if $\mathcal{A}$ has p in cache, there is no cost to assign;*
- *if $\mathcal{A}$ does not have p in cache, then the cost 1 is assigned to the phase I_j^q which contains the request;*
- *if moreover $\mathcal{A}$ loads p in cache (and unloads q), we assign the cost D (paid by $\mathcal{A}$ to load p) is assigned to I_j^q.*

Consider a phase I_j^p.

- *If the page p is* never *in the cache of $\mathcal{A}$ during the first D requests of I_j^p (which are requests to p), then $\mathcal{A}$ pays a cost 1 for each of theese D requests, and we can assign a cost at least D to I_j^p.*
- *Otherwise, p is* loaded *in cache during those first D requests. Now, there are two cases to consider:*
  - *If p is* unloaded *by $\mathcal{A}$ somewhere between the first request in I_j^p and the last, then I_j^p is assigned a cost at least D.*
  - *On the other hand, if p is* never *unloaded during I_j^p, it is in particular* always *in cache during the last consecutive requests of I_j^p (which correspond to requests that decrease $c(p)$ in the online algorithm), and $\mathcal{A}$ pays 1 to access each of the pages which are not in cache. There are at least D of them and therefore I_j^p is again assigned a cost at least D.*

In any case, the cost assigned to each phase is at least D.

(d) Deduce the competitive ratio of the algorithm.

Solution — Any other algorithm $\mathcal{A}$ (including an optimal one) assigns a cost at least D to each phase, and our algorithm pays $3D$ by phase. Therefore, the competitive ratio is 3.

Exercice 3 (Online boxing with advice)**.** A sequence σ of n objects, of size $x_1, \dots, x_n \in (0, 1]$ is given as input. They must be put into boxes of capacity 1, that is to say, the sum of the sizes of the objects inside a box cannot exceed 1. We aim at minimizing the number of nonempty boxes. We consider here the *online* variant of the problem: when the ith object comes in, it must be given a location immediately and permanently.

The objects are classified into four categories:

- the *tiny* objects, with a size in $(0, \frac{1}{3}]$;
- the *small* objects, with a size in $(\frac{1}{3}, \frac{1}{2}]$;
- the *critical* objects, with a size in $(\frac{1}{2}, \frac{2}{3}]$;
- and the *enormous* objects, with a size in $(\frac{2}{3}, 1]$.

The number m of critical objects is known from the beginning (this is called the advice), and we assume that there are a lot of tiny objects. Consider the following algorithm. The m first boxes are call *critical boxes* and $\frac{2}{3}$ of their capacity is reserved for critical objects. Then the objects are handled depending on their category:

Enormous: the object is put into a new box;
Critical: the object is put into one of the critical boxes;
Small: if possible, the object is put into a box that already contains a small object, otherwise it is put into a new box;
Tiny: if possible, the object is put into the first critical box that can contain the object in its nonreserved third, otherwise it is put into a box that already contains a tiny object, or, as a last possibility, into a new box.

(a) The level of a box is the sum of the sizes of the objects that are in it. At the end of the algorithm, show that the level of boxes that contains enormous or small objects is at least $\frac{2}{3}$ (except maybe for one box).

Solution —

- *The level of all boxes with enormous item is at least $\frac{2}{3}$ by definition.*
- *The level of all boxes with small items is at least $\frac{2}{3}$ (if we put 2 small items in it), except possibly the last box opened for small items which may only contain one.*

(b) Assuming that there is a box containing only tiny objects, show that the level of critical boxes (except maybe one) and boxes containing only tiny objects (expect maybe one) is at least $\frac{2}{3}$.

Solution —

- *Each critical box contains one critical object of size at least $\frac{1}{2}$, and then is filled by tiny objects. For the sake of contradiction, assume that the space filled by tiny objects is less than $\frac{1}{6}$ in at least 2 critical boxes B_i and B_j, with $i < j$. Upon packing the tiny objects in B_j, there is still enough room in B_i for them, so they should have been put in B_i according to the strategy. Therefore, it means that there is no tiny object in B_j. However, there is a box T with only tiny objects. When a tiny object is put in T, it means that there is no critical box with enough room for it. But B_j contains no tiny object. Contradiction. Therefore, the level of all critical boxes is at least $\frac{1}{2} + \frac{1}{6} = \frac{2}{3}$, except maybe for one.*

- *When no critical box can contain a tiny object anymore, they are put in the box containing only tiny objects until it is full. Therefore, except maybe for the last box opened for tiny objects only, they all have a level at least $\frac{2}{3}$.*

(c) Assume now that no box contains only tiny objects. Assign a weight to objects (depending on the category) so that: in the online algorithm, the boxes (except maybe one) carry a total weight $\geqslant 2$; in any strategy, the boxes contain a weight $\leqslant 3$.

Solution — Give weight 2 to enormous and critical objects, weight 1 to small objects and 0 to tiny objects. In the online algorithm, the total weight of items in any given bin, except possibly one bin which contain a single small item, is at least 2. Moreover, in any strategy, the maximum weight that a box can have is 3, which occurs when a critical and a small object are placed in the same box.

(d) Deduce the competitive ratio of the online algorithm.

Solution — Assume our algorithm uses $A(\sigma)$ non empty boxes, and let $OPT(\sigma)$ be the optimal number of non empty boxes.

We consider two cases:

(i) If there is a box dedicated to packing tiny objects, then all boxes (except maybe 3 of them) have a level of at least $\frac{2}{3}$.

- *For each box B_i, let l_i be its level and let $L(\sigma)$ be the sum of the sizes of all items in σ. Then, by partitioning into all the non empty boxes,*

$$L(\sigma) = \sum_{i=1}^{A(\sigma)} l_i \geq \frac{2}{3}(A(\sigma) - 3)$$

i.e.

$$A(\sigma) \leq \frac{3}{2}L(\sigma) + 3.$$

- *On the other hand, in any strategy, all boxes have level at most 1 by definition. So, $L(\sigma) \leq OPT(\sigma)$.*

Therefore,

$$A(\sigma) \leq \frac{3}{2}OPT(\sigma) + 3.$$

(ii) Otherwise we use the technique of the previous question and introduce the weights.

- *For each box B_i, let w_i be its weight and let W be the total weight of items in σ. Then*

$$W(\sigma) = \sum_{i=1}^{A(\sigma)} w_i \geq 2A(\sigma).$$

- *On the other hand, in any strategy, all boxes have a weight at most 3:*

$$W(\sigma) = \sum_{i=1}^{OPT(\sigma)} w_i^* \leq 3OPT(\sigma).$$

Therefore,

$$A(\sigma) \leq \frac{3}{2}OPT(\sigma).$$

In any case, we have $A(\sigma) \leq \frac{3}{2}OPT(\sigma) + cst$ i.e. *the online algorithm is* $\frac{3}{2}$*−competitive.*