DM582 Exercises - Sheet 11

Mads Anker Nielsen Tobias Samsøe Sørensen

April 22, 2025

This document contains exercises from the course DM582 (spring 2025). Most exercises are from the book *Introduction to Algorithms, 4th edition* by Cormen, Leiserson, Rivest, and Stein (CLRS), the book *Algorithm Design, 1st edition* by J. Kleinberg and E. Tardos (KT), and the book *Discrete Mathematics and its Applications, 8th edition* by K. Rosen.

The solutions given here might differ from the solutions discussed in class. In class, we place more emphasis on the intuition leading to the correct answer. Please do not consider reading these solutions an alternative to attending the exercise classes.

References to CLRS refer to the book *Introduction to Algorithms, 4th edition* by Cormen, Leiserson, Rivest, and Stein.

References to KT refer to the book *Algorithm Design, 1st edition* by J. Kleinberg and E. Tardos.

References to Rosen refer to the book *Discrete Mathematics and its Applications, 8th edition* by K. Rosen.

References to BG refer to the book *Computer Algorithms: Introduction to Design and Analysis, 3rd edition* by Sara Baase and Allen Van Gelder.

This document will inevitably contain mistakes. If you find some, please report them to your TA so that we can correct them.

Sheet 11

Exercise from course webpage

Exercise

A core problem for an online machine scheduling algorithm is whether to make a level or a skewed schedule. For instance, just considering $m = 2$ and the sequences 1, 1 and 1, 1, 2, we do not know what the best option is for the second job: placing it on the *same* machine as the first job *or not*. (Because this depends on the future: do we receive a third job of size 2 or not?) Try to randomize the decision as we did for the Treasure Hunt problem in the lecture. Assuming you get one of these two sequences, can you get an expected ratio smaller than $\frac{3}{2}$?

Suggested solution

Let $\mathcal{A}$ be the algorithm that places the first job on some machine 1 and then places the second job on machine 2 with probability p and on machine 1 with probability $1 - p$. $\mathcal{A}$ places the third job on machine 2 deterministically (if it arrives). We don't care what $\mathcal{A}$ does after the third job in this exercise. For σ_1, we have

$$\mathrm{E}[\mathcal{A}(\sigma_1)] = p + 2(1 - p) = 2 - p$$

and $\mathrm{OPT}(\sigma_1) = 1$, so $\overline{\mathcal{R}} \geq (2 - p)/1 = 2 - p$. For σ_2, we have

$$\mathrm{E}[\mathcal{A}(\sigma_2)] = 3p + 2(1 - p) = 2 + p$$

and $\mathrm{OPT}(\sigma_2) = 2$, so $\overline{\mathcal{R}} \geq (2 + p)/2 = 1 + p/2$. Since the competitive ratio of $\mathcal{A}$ is bounded from below by both $2 - p$ and $1 + p/2$, we should pick p such that the largest of the two ratios is minimized. Since $2 - p$ is decreasing in p and $1 + p/2$ is increasing in p, their maximum is minimized when they are equal. We find that

$$\begin{aligned} 2 - p &= 1 + \frac{p}{2} \\ \Leftrightarrow 1 &= \frac{3}{2}p \\ \Leftrightarrow p &= \frac{2}{3}. \end{aligned}$$

Setting $p = 2/3$, the lower bound provided by the sequences σ_1 and σ_2 is $2 - 2/3 = 4/3 < 3/2$.

Exercise from course webpage

Exercise

A Professor Larsen with extremely poor eyesight is at a long, long wall. He's told that there is a hole somewhere where he can get out, but he won't be able to see it unless he's right at it; and he doesn't know if it's to the left or to the right. The hole is a whole number of steps away from the professor. Devise an algorithm the professor can use to escape. What's the competitive ratio of your algorithm? (Of course, OPT is just the distance from the professor to the hole.)

Suggested solution

Let call our algorithm $\mathcal{A}$. Perhaps the simplest approach would be to first move 1 step in one direction, then 2 steps in the other, then 3 in the first and so on. However, this algorithm turns out not to be competitive at all. If the hole is n steps in one direction, then our algorithm makes at least $1+2+\cdots+n = n(n+1)/2$ steps before finding the hole whereas OPT makes only n. Thus, $\mathcal{R}(\mathcal{A}) \geq (n+1)/2$ which depends on n! Thus, there exist no constants b and c such that $\mathcal{A}(\sigma) \leq c \cdot \textsc{Opt}(\sigma) + b$.

A better idea is to take larger chunks of steps in each direction. The strategy presented here is not optimal, but is chosen for the simplicity of its analysis. The strategy we went through in class is better.

The strategy is as follows: in the k-th round, move 2^k steps right, 2^{k+1} steps left and 2^k steps right again such that we are back at the starting point. The number of steps taken in the k-th round is then $2^k + 2^{k+1} + 2^k = 2^{k+2}$. Using this strategy, we find the door in the k-th round where k is the least integer such that $2^k \geq n$. The least k for which $2^k \geq n$ is $k = \lceil \log_2(n) \rceil \leq \log_2(n) + 1$. The number of steps taken from round 0 through round k is $2^2 + 2^3 + \cdots + 2^{k+2} < 2^{k+3} \leq 2^{\log_2(n)+4} = 16 \cdot 2^{\log_2(n)} = 16n$. Thus, $\mathcal{R}(\mathcal{A}) < \frac{16n}{n} = 16$.

Exercise from course webpage

Exercise

Show an example where lazy Dc benefits from being lazy and one where it doesn't.

Suggested solution

In the trivial case of an empty input sequence or a sequence with requests only to points which already have servers, lazy Dc has no advantage over Dc. Figure 1 shows a simple example where lazy Dc benefits from being lazy.

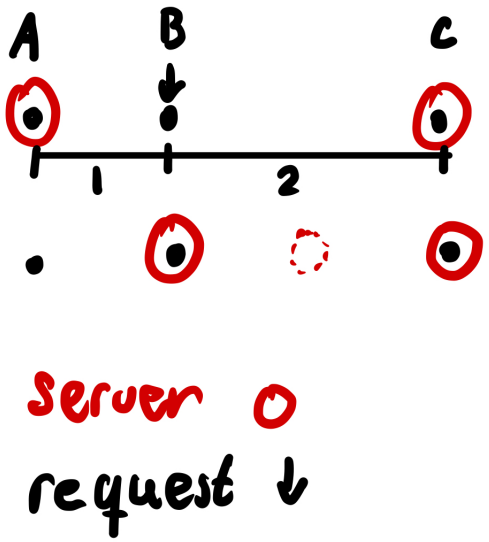

Figure 1: A sequence where lazy Dc benefits from being lazy.

Exercise from course webpage

Exercise

Show that if there exists an infinite family of sequences such that 1) for some fixed constant $b > 0$, we have that $\textsc{Alg}(\sigma) \geq k\textsc{Opt}(\sigma) - b$ for any σ in the family, and 2) for any d, there exists a σ in the family such that $\textsc{Opt}(\sigma) > d$, then $\textsc{Alg}$ cannot be $(k-\epsilon)$-competitive for any $\epsilon > 0$.

Suggested solution

The intuition here is that, based on a, b, and ϵ, we can always pick σ such that $\textsc{Opt}(\sigma)$ is so large that we obtain a contradiction to $\textsc{Alg}$ being $(k-\epsilon)$-competitve.

Fix an arbitrary $\epsilon > 0$. Suppose $\textsc{Alg}$ is $(k-\epsilon)$-competitive. Then there exists a such that

$$\textsc{Alg}(\sigma) \leq (k-\epsilon)\textsc{Opt}(\sigma) + a \tag{1}$$

for all σ. If we can find a σ such that (1) is violated, then $\textsc{Alg}$ cannot be $(k-\epsilon)$-competitive. We now show that such a σ exists. Indeed,

$$\begin{aligned} & k\textsc{Opt}(\sigma) - b > (k-\epsilon)\textsc{Opt}(\sigma) + a \\ \Leftrightarrow\ & k\textsc{Opt}(\sigma) - (k-\epsilon)\textsc{Opt}(\sigma) > a + b \\ \Leftrightarrow\ & \epsilon\textsc{Opt}(\sigma) > a + b \\ \Leftrightarrow\ & \textsc{Opt}(\sigma) > \frac{a+b}{\epsilon} \end{aligned}$$

and there exists a σ in the family such that $\textsc{Opt}(\sigma) > \frac{a+b}{\epsilon}$ by 2). Let σ be such a sequence. By 1) and the choice of σ,

$$\textsc{Alg}(\sigma) \geq k\textsc{Opt}(\sigma) - b > (k-\epsilon)\textsc{Opt}(\sigma) + a,$$

which is a contradiction to (1). Since ϵ was arbitrary, we conclude that $\textsc{Alg}$ cannot be $(k-\epsilon)$-competitive for any $\epsilon > 0$.

Exercise from course webpage

Exercise

Prove that there exists a minimum weight matching where the server DC moves (when it only moves one) is matched to the server OPT just moved.

Suggested solution

We consider the case where DC moves only the rightmost server. The proof for the case where DC moves the leftmost server is identical.

Let A_{OPT} and A_{DC} be the servers that OPT and DC move, respectively. Let M be a minimum weight matching between OPT's and DC's servers. If A_{DC} is not matched to A_{OPT} in M, we show that one can modify M such that A_{DC} is matched to A_{OPT} without increasing the weight of M.

Suppose A_{DC} is not matched to A_{OPT} in M and let B_{OPT} and B_{DC} be the servers that A_{DC} and A_{OPT} are matched to, respectively. Since DC moves only the rightmost server, B_{DC} is to the left of A_{DC} (see top of Figure 2). The claim follows by considering the 4 possible cases for the relative position of B_{OPT}. The 4 cases are illustrated in Figure 2.

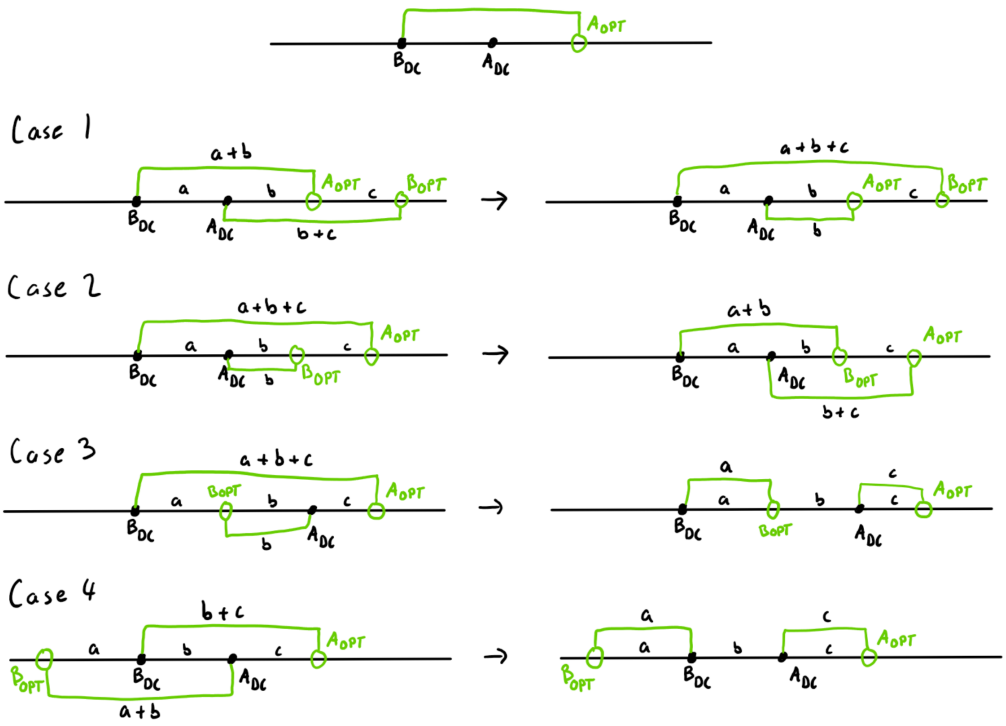

Figure 2: Illustration of the transformation of the matching M.

Exercise from course webpage

Exercise

Discuss whether or not one should assume that the adversary knows the coin flips of the algorithm - in the light that we want results in our model to say something about the real world.

Suggested solution

Probably not (no pun intended). When measuring the quality of an online algorithm in terms of its competitive ratio, we are assuming a worst-case scenario for the algorithm, in which the input is constructed by an evil adversary knowing the implementation of the algorithm. One could argue that this is already not a very realistic model, as input in most real world scenarios is not constructed by an adversary. Allowing the adversary to know the coin flips of the algorithm would correspond to analyzing a scenario in which the input depends not only the implementation of the algorithm, but also on random events. This is arguably even less realistic.

Exercise from course webpage

Exercise

Consider how one might generalize Dc to working on trees. No proofs are expected.

Suggested solution

There are several ways to generalize Dc to trees. We discussed a few ideas in class. The following generalization can be shown to be k-competitive, although we do not provide a proof here.

When we get a request for a point, start moving all servers with an unobstructed (by other servers) path to the requested point towards the request at the same speed. Whenever the path to a server becomes obstructed by another server, stop moving that server. When one server reaches the point, stop moving all servers.

If the tree is a path, this algorithm is equivalent to Dc on the line.