Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

HS 2023
Prof. R. Wattenhofer / J. Mathys

Discrete Event Systems

Solution to Exercise Sheet 10

1 Competitive Analysis

Competitive Analysis

The *competitive analysis* of an algorithm ALG consists of two separate steps. First, we show that for an arbitrary problem instance, the result of ALG is asymptotically at most a factor r worse than the the optimal *offline* result. This yields an upper bound on ALG's result, that is $\text{cost}_{\text{ALG}} \leq r \cdot \text{cost}_{\text{OPT}} + c$. If the task is to show that ALG is constant-competitive for a constant r, then we are done. If we are interested in a tight analysis, we have to show that there is a problem instance where the result of ALG is a factor r worse than the optimal *offline* result. This gives a matching lower bound on the objective value of the algorithm, $\text{cost}_{\text{ALG}} \geq r \cdot \text{cost}_{\text{OPT}}$.

Naturally, the second step is easier than the first one because we *just* have to find a "bad instance". The first step is often much more involved. A pattern that works quite often is the following.

1. Consider an arbitrary input sequence for ALG.
2. Partition the input sequence into *suitable* parts.
3. Show that $\text{cost}_{\text{ALG}} \leq r \cdot \text{cost}_{\text{OPT}}$ for each part.

The tricky part here is to find a *suitable* partition in step 2.

a) Recall that calls have infinite duration. Therefore, once a cell accepts a call, no neighboring cell can accept a call thereafter. The natural greedy algorithm GREEDY accepts a call, whenever this is possible. That is, a call in cell C is accepted if no neighboring cell of C has previously accepted a call.

We will assign colors to cells according to the following strategy:

- blue for calls that are accepted by OPT but rejected by GREEDY
- red for calls that are accepted by GREEDY but rejected by OPT

Note that cells that were accepted or rejected by both algorithms do not contribute to the competitive ratio and are therefore not colored. Assume, we have a red cell, i.e. OPT rejects a call in some cell A that GREEDY accepts (see Figure 1). This cell will have at least one blue neighbor, since GREEDY cannot accept any calls in the neighborhood and OPT would have accepted a call in cell A if it had not accepted any call in a neighboring cell. So, we have at least as many blue cells as red cells.

Next, assume that there are four calls, the first one in A, then three non-interfering ones in neighboring cells B, C, and D. GREEDY accepts the first call while OPT rejects it and

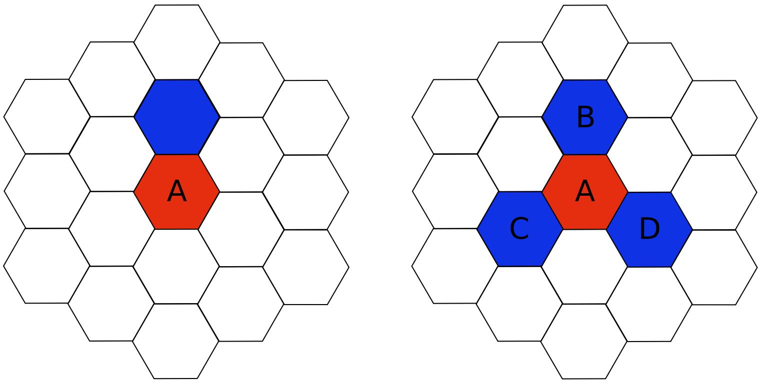

Figure 1: Coloring of Greedy and Opt

accepts the remaining three (see Figure 1). This leads to a situation where a red cell has three blue neighbors. In this case, Greedy has cost 3 while Opt hast cost 1. Since the sequence can be iterated ad infinitum, there is no $r < 3$ s.t. the algorithm is r-competitive. It follows that Greedy has a competitive ratio of 3.

b) There is no constant-competitive algorithm if calls can have arbitrary durations. We show this, again, by designing a "cruel" input sequence. Assume that a sequence of unit duration calls arrives in cell A. Clearly, Alg must accept a call at some point as otherwise Opt will accept all the calls and competitive ratio will be arbitrarily large.

After Alg accepts the first unit duration call in the input sequence, there will be a sequence of $1/\varepsilon$ calls each of duration ε in cell B, which Alg will have to reject. However, Opt will reject the call accepted by Alg and accept all the others. Thus, we have $\text{cost}_{\text{OPT}} = 1$ and $\text{cost}_{\text{ALG}} = 1/\varepsilon + x$, where x is the number of calls rejected by Alg before it accepts the first call. For any given constant r, we can choose ε so that $\text{cost}_{\text{ALG}} > r \cdot \text{cost}_{\text{OPT}}$ and competitive ratio will be larger than r.

Note: At first sight, it seems that there is no better algorithm than the natural greedy algorithm from part a) of the exercise. After all, the algorithm must accept the first call in order to stay competitive. Accepting the first call, on the other hand, leads to a competitive ratio of 3. However, it can be shown that there exists a *randomized algorithm* with competitive ratio 2.97. This algorithm accepts every call with a certain probability.

2 Competitive Christmas

In the lecture, we defined the competitive ratio in terms of costs. The smaller the costs, the better is our algorithm. In this task it does not make sense to talk about costs, since a larger tree also means a better result for the algorithm. Therefore, we will use a different concept of defining the competitive ratio. We will consider *benefits* instead of costs: the optimal algorithm reaches the largest benefit; the closer an algorithm gets to this value from below, the closer is the competitive ratio to 1. The following inequality defines the competitiveness in terms of benefits:

$$\text{benefit}_{\text{ALG}}(I) \geq \frac{1}{r} \cdot \text{benefit}_{\text{OPT}}(I) - c$$

and the competitive ratio is defined as

$$r = \inf_{I \in \mathcal{I}} \frac{\text{benefit}_{\text{OPT}}(I)}{\text{benefit}_{\text{ALG}}(I)}$$

Now we can start developing the algorithm: If our algorithm tells Roger to buy a tree of size c, then it is followed in the worst-case by a tree of size b which yields a competitive ratio of $\rho_A = b/c$. To make ρ_A as small as possible, Roger should only buy a tree if its size c is as large as possible. If he ignores a tree of size $c - \varepsilon$ in the hope that he finds a larger tree, he might in the worst-case only encounter trees of size a yielding a competitive ratio of $\rho_B = (c - \varepsilon)/a$. In these two cases ρ should be as small as possible such that Roger gets a tree that is as large as possible. Hence, we have to set the value c at (and above) which Roger buys a tree such that the maximum of ρ_A and ρ_B is minimised. This is the case for $\rho_A = \rho_B$ which yields $c = \sqrt{ab}$. The corresponding competitive ratio is $\sqrt{b/a}$.

3 The Best Secretary

a) To avoid a potentially bad ordering of the candidates, Roger invites them into his room in a random order. Within the first half ($n/2$ candidates), Roger only rates the applicants and memorizes the best quality G. In the second half, Roger accepts a candidate if its quality exceeds G. Because every candidate has a unique rating, the algorithm will output the best candidate is the second-best one is in the first half, and the best one is in the second half.

The probability that the second-best candidate is in the first half is $1/2$. Conditioned on this event, the probability that the best candidate is in the second half is at least $1/2$. If both these events happen, Roger actually hires the best candidate. This case occurs with probability of at least $1/2 \cdot 1/2 = 1/4$.

Note that this is for the case of even n, the odd case can be shown analogously.

b) The analysis for the above algorithm clearly is only a rough estimate. We only considered the special case where the second-best secretary is in the first half and the best one in the second half. Roger also would have chosen the best one if the third-best is in the first half and the best and the second-best candidate in the second half but in this order. Furthermore, we assumed without justification that the algorithm rates the *first half* of the candidates and then selects one from the second half.

For an exact analysis, assume that Roger first rates $x \cdot n$ candidates (for $x \in [0, 1]$) and then chooses the candidate that is better than the best one from these $x \cdot n$ applicants. Let $\mathbf{1}$ and $\mathbf{2}$ denote the set of the candidates in the first and second part, respectively.

We denote by C_i the i-th best candidate and define the probability p_i for the event that $C_i \in \mathbf{1}$ and all better ones are in $\mathbf{2}$ with the best candidate first. The case (special case from above) that $C_2 \in \mathbf{1}$ and $C_1 \in \mathbf{2}$ occurs with probability

$$\begin{aligned} p_2 &= \Pr[C_2 \in \mathbf{1}] \cdot \Pr[C_1 \in \mathbf{2} \mid C_2 \in \mathbf{1}] \\ &= x \cdot \frac{(1-x) \cdot n}{n-1} \quad . \end{aligned}$$

Note that the conditional probability for C_1 being in $\mathbf{2}$ given that C_2 is in $\mathbf{1}$ is bigger than $(1 - x)$ since one "slot" in $\mathbf{1}$ is already taken by C_2.

Similiarly, we can calculate the probability p_3 but we now also have to consider the probability that C_1 appears before C_2 in $\mathbf{2}$.

$$\begin{aligned} p_3 &= \Pr[C_3 \in \mathbf{1}] \cdot \Pr[C_2 \in \mathbf{2} \mid C_3 \in \mathbf{1}] \cdot \Pr[C_1 \in \mathbf{2} \mid C_3 \in \mathbf{1} \text{ and } C_2 \in \mathbf{2}] \cdot \Pr[C_1 \text{ before } C_2] \\ &= x \cdot \frac{(1-x) \cdot n}{n-1} \cdot \frac{(1-x) \cdot n - 1}{n-2} \cdot \frac{1}{2} \end{aligned}$$

Note again that we have to use conditional probabilities here.

Analogous to p_3, we can derive a formula for p_i.

$$\begin{aligned} p_i &= x \cdot \left(\prod_{j=0}^{i-2} \frac{(1-x)\cdot n - j}{n-j-1} \right) \cdot \frac{1}{i-1} \\ &= x \cdot \left(\prod_{j=0}^{i-2} \left[(1-x) \cdot \frac{n}{n-j-1} - \frac{j}{n-j-1} \right] \right) \cdot \frac{1}{i-1} \end{aligned}$$

Since we are interested in the probabilities for large n, we can consider the limit instead.

$$\begin{aligned} p_i' = \lim_{n\to\infty} p_i &= \lim_{n\to\infty} x \cdot \left(\prod_{j=0}^{i-2} \left[(1-x) \cdot \frac{n}{n-j-1} - \frac{j}{n-j-1} \right] \right) \cdot \frac{1}{i-1} \\ &= x \cdot \left(\prod_{j=0}^{i-2} (1-x) \right) \cdot \frac{1}{i-1} \\ &= x \cdot (1-x)^{i-1} \cdot \frac{1}{i-1} \end{aligned}$$

Note that this result corresponds to ignoring the dependence between the candidates in both parts by calculating without conditional probabilities.

Now we can calculate the success probability p_{succ} for hiring the best candidate.

$$\begin{aligned} p_{\text{succ}}(x) &= \sum_{i=2}^{\infty} p_i' \\ &= \sum_{i=1}^{\infty} \frac{x}{i} \cdot (1-x)^i \\ &= x \cdot \sum_{i=1}^{\infty} \frac{(1-x)^i}{i} \\ &= -x \ln(x) \end{aligned}$$

Differentiating yields that $p_{\text{succ}}(x)$ attains its maximum for $x = 1/e$ and we have further $p_{\text{succ}}(1/e) = 1/e$. Hence, Roger should only rate a fraction of $1/e \approx 37\,\%$ of the candidates and then start with the selection as described above. Then, the probability to get the best secretary is $1/e$.