ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

HS 2012 Prof. C. Stamm / K.-T. Foerster, T. Langner, J. Seidel

Discrete Event Systems

Solution to Exercise Sheet 12

1 Competitive Analysis

Competitiveness

In the script there is a definition for the competitiveness of an algorithm. However, this definition only holds if we want to evaluate an algorithm by means of its costs. Sometimes, we want to compare algorithms regarding their *benefit* rather than their costs. In this case, we have to be a bit careful with the definition of a c-competitive algorithm. We say that an algorithm ALG is c-competitive, if for all finite input sequences the solution of algorithm ALG is at most a factor c worse than the optimal algorithm, regardless of the algorithms being compared concerning costs or benefit.
According to whether we evaluate an algorithm based on costs or benefit, an algorithm ALG is c-competitive if for all finite input sequences I

$$\text{cost}_{\text{ALG}}(I) \leq c \cdot \text{cost}_{\text{OPT}}(I) + k \quad \text{or}$$

$$\text{benefit}_{\text{ALG}}(I) \geq \frac{1}{c} \cdot \text{benefit}_{\text{OPT}}(I) - k \quad \text{respectively.}$$

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 c worse than the the optimal *offline* result. This yields an upper bound on ALG's result, that is $\text{cost}_{\text{ALG}} \leq c \cdot \text{cost}_{\text{OPT}} + k$. If the task is to show that ALG is c-competitive for a constant c, 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 c worse than the optimal *offline* result. This gives a matching lower bound on the objective value of the algorithm, $\text{cost}_{\text{ALG}} \geq c \cdot \text{cost}_{\text{OPT}}$.

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 $c_{\text{ALG}} \leq c \cdot c_{\text{OPT}}$ 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.

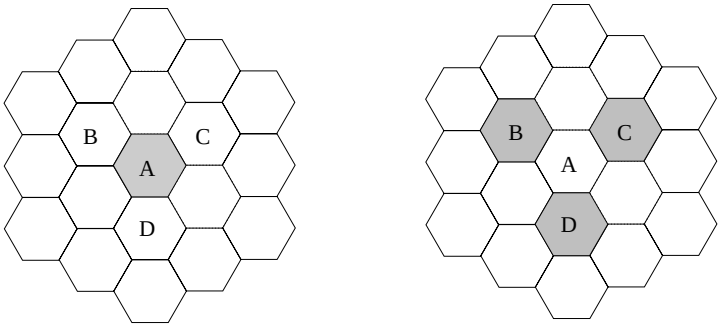

Figure 1: The solutions GREEDY (left) and OPT (right)

For every call accepted by GREEDY, there are two cases:

- OPT accepted the call as well
- OPT rejected the call which may enable it to accept at most three additional calls (see Figure 1)

Thus, OPT can accept at most three times as many calls as GREEDY. 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 and has benefit 1. OPT rejects the first call, but accepts the remaining three, resulting in a benefit of 3. The algorithm is 3-competitive.

b) There is no competitive algorithm if calls can have arbitrary durations. We show this, again, by designing a "cruel" input sequence. Assume that the first call arrives in A and has arbitrary duration. There are two possible actions for an algorithm ALG.

If ALG rejects this call, no further calls will arrive and therefore benefit(ALG) = 0. The optimal algorithm would have accepted the call, i.e., benefit(OPT) = 1. The competitive ratio is infinitely large.

On the other hand, if ALG accepts the call, there will be infinitely many calls coming in cell B, each of which has very short duration ε. While ALG cannot accept any of these calls (because the call in A has infinite duration), the optimal algorithm rejects the first call and accepts all subsequent calls. This yields benefit(ALG) = 1 as opposed to benefit(OPT) = β, for an arbitrarily large value of β.

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

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 above which Roger buys a tree such that ρ_A and ρ_B are 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 memorises the best quality G. In the second half, Roger accepts a candidate if its quality exceeds G.

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

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 **1** and **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 **2** with the best candidate first. The case (special case from above) that the $C_2 \in \mathbf{1}$ and $C_1 \in \mathbf{2}$ occurs with probability

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

Note that the conditional probability for C_1 being in **2** given that C_2 is in **1** is bigger than $(1-x)$ since one "slot" in **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 **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.

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

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

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

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 $1/e$.