Exercise XIII, Algorithms 2024-2025

These exercises are for your own benefit. Feel free to collaborate and share your answers with other students. There are many problems on this set, solve as many as you can and ask for help if you get stuck for too long. Problems marked * are more difficult but also more fun :).

1 Online Algorithms

1 Imagine the cost to buy skis (B) is 120 CHF and the cost to rent skis (R) is 30 CHF per season. Consider the "Deterministic Break-Even" algorithm: rent for $B/R-1$ seasons, then buy the skis. (So, rent for $120/30-1=3$ seasons, then buy).

(a) If you go skiing for exactly 2 seasons, what is the cost of this algorithm? What would have been the optimal cost?

(b) If you go skiing for exactly 4 seasons, what is the cost of this algorithm? What would have been the optimal cost?

(c) If you go skiing for 6 seasons, what is the cost of this algorithm? What would have been the optimal cost?

Solution:

(a) The cost incurred by the "Deterministic Break-Even" algorithm would be $30 \cdot 2 = 60$ CHF, which is optimal.

(b) The cost incurred by the "Deterministic Break-Even" algorithm would be $30 \cdot 3 + 120 = 210$ CHF, while the optimal solution is to buy skis immediately for a cost of 120 CHF.

(c) Same as (b).

2 Consider the Rent or Buy (Ski Rental) algorithm that we saw in class, where we assume renting costs one unit, i.e., $R = 1$. Suppose you modify the rule to "rent until you've paid $c \cdot B$, then buy." Express the competitive ratio as a function of c.

Solution: Let n be the number of times we go skiing. The cost of the optimal solution is always $\min\{n, B\}$. If $n \le c \cdot B$, then the cost we pay is exactly n, otherwise we pay $c \cdot B + B$. Hence, we can write our competitive ratio as a function of $c > 0$ as

$$CR(c) = \max\left\{CR_{low}(c), CR_{high}(c)\right\} ,$$

where

$$CR_{low}(c) = \max_{n=1,\dots,c\cdot B} \frac{n}{\min\{n, B\}}$$

and

$$CR_{high}(c) = \max_{n \ge c\cdot B+1} \frac{(1+c)\cdot B}{\min\{n, B\}}$$

We first look at $CR_{low}(c)$, and note that for every $n \le B$, one has

$$\frac{n}{\min\{n, B\}} = 1\,.$$

This means that if $c \le 1$, we get

$$\max_{n=1,\dots,c\cdot B} \frac{n}{\min\{n, B\}} = 1\,,$$

while if $c > 1$, we get

$$\max_{n=1,\dots,c\cdot B} \frac{n}{\min\{n, B\}} = c\,.$$

Hence, we can say that for any $c > 0$ we have

$$CR_{low}(c) = \max\{1, c\}\,.$$

Now we analyze $CR_{high}(c)$. If $n \ge B$, we have

$$\frac{(1+c)\cdot B}{\min\{n, B\}} = 1 + c\,.$$

If instead n is such that $c \cdot B + 1 \le n \le B$, we have

$$\frac{(1+c)\cdot B}{\min\{n, B\}} = \frac{(1+c)\cdot B}{n}\,,$$

but note that there can be such an n if and only if $c \le 1 - 1/B$, in which case we get

$$\max_{n=c\cdot B+1,\dots,,B} \frac{(1+c)\cdot B}{\min\{n, B\}} = \frac{(1+c)B}{(c+1/B)B} = \frac{1+c}{1/B+c}\,.$$

Hence, we can say that for any $c > 0$ we have

$$CR_{high}(c) = \frac{1+c}{\min\{1, 1/B + c\}}\,.$$

Combining CR_{low} and CR_{high} we get

$$CR(c) = \max\left\{\max\{1, c\},\, \max\left\{1+c, \frac{1+c}{1/B+c}\right\}\right\} = \max\left\{1+c, \frac{1+c}{1/B+c}\right\}\,.$$

3 Prove that no deterministic online algorithm for the ski-rental problem can achieve a competitive ratio strictly less than 2 when the cost of buying $B \to \infty$ (i.e. show 2 is optimal) by constructing an adversarial number of ski trips.

Solution: Let $\mathcal{A}$ be any deterministic online algorithm for the ski-rental problem. We denote with $n^*(\mathcal{A})$ the adversarial number of ski trips for $\mathcal{A}$, and define it to be exactly the number of ski trips at which $\mathcal{A}$ decides to buy skis. Then, the cost paid by $\mathcal{A}$ is $(n^*(\mathcal{A}) - 1) + B$ (since it buys skis at the $n^*(\mathcal{A})$-th ski trip), while the cost of the optimal solution is $\min\{n^*(\mathcal{A}), B\}$. Then, the competitive ratio of $\mathcal{A}$ is at least the ratio

$$\frac{(n^*(\mathcal{A}) - 1) + B}{\min\{n^*(\mathcal{A}), B\}}\,. \tag{1}$$

If $n^*(\mathcal{A}) > B$, then the ratio in (1) can be lower-bounded as

$$\frac{(n^*(\mathcal{A})-1)+B}{\min\{n^*(\mathcal{A}),B\}} = 1 + \frac{n^*(\mathcal{A})-1}{B} > 2 - \frac{1}{B}\,.$$

If $n^*(\mathcal{A}) \le B$, then the ratio in (1) can be lower-bounded as

$$\frac{(n^*(\mathcal{A})-1)+B}{\min\{n^*(\mathcal{A}),B\}} = 1 + \frac{B-1}{n^*(\mathcal{A})} \ge 1 + \frac{B-1}{B} = 2 - \frac{1}{B}\,.$$

Hence, the competitive ratio of $\mathcal{A}$ is at least $2 - 1/B$. This means that in the limit of $B \to \infty$, we get that a competitive ratio of 2 is the best possible[1].

4 Consider a cache with size $k = 3$ (meaning it can hold 3 pages). Initially, the cache is empty. Process the following sequence of page requests: A, B, C, D, A, B, E, D, A, B, C

(a) For the LRU (Least Recently Used) algorithm, list the cache contents after each request and count the total number of cache misses. A cache miss occurs if the requested page is not in the cache. If a miss occurs and the cache is full, LRU evicts the page that has been unused for the longest time.

(b) For the FIFO (First-In, First-Out) algorithm, list the cache contents after each request and count the total number of cache misses. If a miss occurs and the cache is full, FIFO evicts the page that has been in the cache the longest (like a queue).

(c) Based on your results for this sequence, which algorithm performed better?

Solution:

(a) The table below shows the evolution of the cache using the LRU algorithm, where the right-most element in the list is the next page to evict. The total number of cache misses is 11.

Request	Miss/Hit	Cache After Request	Evicted
Initial	–	`[]`	–
A	Miss	`[A]`	–
B	Miss	`[B, A]`	–
C	Miss	`[C, B, A]`	–
D	Miss	`[D, C, B]`	A
A	Miss	`[A, D, C]`	B
B	Miss	`[B, A, D]`	C
E	Miss	`[E, B, A]`	D
D	Miss	`[D, E, B]`	A
A	Miss	`[A, D, E]`	B
B	Miss	`[B, A, D]`	E
C	Miss	`[C, B, A]`	D

[1]Note that if we optimize $CR(c)$ in Problem 2, we get a competitive ratio of $2 - 1/B$ (by setting $c = 1 - 1/B$). While this is slightly better than the 2-competitive algorithm seen in class, this does not contradict the lower bound we found in Problem 3 and it has the same performance as the algorithm from class in the limit of $B \to \infty$.

(b) The table below shows the evolution of the cache using the FIFO algorithm, where the right-most element in the list is the next page to evict. The total number of cache misses is 11.

Request	Miss/Hit	Cache After Request	Evicted
Initial	–	`[]`	–
A	Miss	`[A]`	–
B	Miss	`[B, A]`	–
C	Miss	`[C, B, A]`	–
D	Miss	`[D, C, B]`	A
A	Miss	`[A, D, C]`	B
B	Miss	`[B, A, D]`	C
E	Miss	`[E, B, A]`	D
D	Miss	`[D, E, B]`	A
A	Miss	`[A, D, E]`	B
B	Miss	`[B, A, D]`	E
C	Miss	`[C, B, A]`	D

(c) The two algorithms have the same performance. This is because, with this sequence of requests, the least recently used page is also the one that has been in the cache the longest, and therefore the two algorithms perform identically.

5 Imagine you are a hiring manager tasked with selecting the single best candidate for a job out of n total applicants. The hiring process has a peculiar structure:

1. **Group A (The Random Sample):** From the total pool of n applicants, a group of $n/2$ candidates is **chosen uniformly at random** to form Group A. These $n/2$ candidates arrive first, one by one, in a **uniformly random permutation** (i.e., their internal order within Group A is random).

2. **Group B (The Adversary's Play):** The remaining $n/2$ candidates (those not chosen for Group A) form Group B. These candidates also arrive one by one, but their arrival order is determined by an **adversary** who knows your hiring strategy for this second group and wants to prevent you from hiring the overall best candidate.

As in the classic problem:

- When a candidate arrives, you learn their "score" or rank relative to those already seen.
- You must decide immediately whether to hire them or reject them.
- The decision is **irrevocable**. If you hire someone, the process stops. If you reject them, you cannot hire them later.
- You can hire at most one candidate.
- Your goal is to maximize the probability of hiring the **single overall best candidate** from the entire pool of n applicants.

Your Task: Show that you can devise a strategy that ensures you hire the overall best candidate with a probability of at least $1/4$, regardless of the adversary's actions in Group B (for $n \geq 2$).

Solution: The strategy is as follows:

1. **Observe Group A:** Allow all $n/2$ candidates in Group A (random sample, random internal order) to pass. Identify M, the best candidate seen in Group A. Do not hire from Group A.
2. **Select from Group B:** Hire the *first* candidate from Group B (adversarial order) who is better than M. If none, hire no one.

(Assume $n \geq 2$ and n is even).

Analysis of the Strategy. Let C_{best} be the overall best candidate and C_{second_best} be the overall second-best. The strategy successfully hires C_{best} if $C_{best} \in$ Group B and $C_{second\_best} \in$ Group A.

The probability of this joint event is:

$$\begin{aligned} P(\text{Success Conditions Met}) &= P(C_{best} \in \text{Group B AND } C_{second\_best} \in \text{Group A}) \\ &= P(C_{best} \in \text{Group B}) \times P(C_{second\_best} \in \text{Group A} \mid C_{best} \in \text{Group B}) \\ &= \frac{1}{2} \times \frac{n/2}{n-1} = \frac{n}{4(n-1)} \end{aligned}$$

The first term $P(C_{best} \in \text{Group B}) = \frac{n/2}{n} = \frac{1}{2}$. The second term $P(C_{second\_best} \in \text{Group A} \mid C_{best} \in \text{Group B}) = \frac{n/2}{n-1}$ because Group A must select C_{second_best} from the $n-1$ candidates remaining (excluding C_{best}) to fill its $n/2$ slots.

Adversary's Role. If $C_{best} \in$ Group B and $C_{second\_best} \in$ Group A:

- The benchmark M (best of Group A) becomes C_{second_best}.
- The strategy seeks the first candidate in Group B better than $M = C_{second\_best}$.
- The only candidate in Group B better than C_{second_best} is C_{best} itself.
- The adversary, controlling Group B's order, cannot present a decoy X from Group B such that $M < X < C_{best}$, as no such X exists in Group B.

Thus, if these conditions on the locations of C_{best} and C_{second_best} hold, the strategy guarantees hiring C_{best}.

Conclusion: Probability of Success. The probability of success is $P(\text{Success}) = \frac{n}{4(n-1)}$. We need to show this is at least $1/4$ for $n \geq 2$:

$$\frac{n}{4(n-1)} \geq \frac{1}{4} \quad \iff \quad \frac{n}{n-1} \geq 1$$

Since $n \geq 2$, $n-1 \geq 1$ (and is positive). Also, $n \geq n-1$. Dividing by $n-1$ preserves the inequality: $\frac{n}{n-1} \geq 1$. Thus, $P(\text{Success}) = \frac{n}{4(n-1)} \geq \frac{1}{4}$.

2 Weighted Majority and Hedge

6 Recall from the lecture that the number of mistakes that Weighted Majority makes is at most $2(1+\epsilon)\cdot(\#\text{ of } i\text{'s mistakes}) + O(\log N/\epsilon)$, where i is any expert and N is the number of experts.

Give an example that shows that the factor 2 is tight in the above bound. The simplest such example only uses two experts, i.e., $N = 2$, and each of the experts is wrong roughly half of the time. Finally, note how your example motivates the use of a random strategy (as in the Hedge strategy).

Solution: If there are two experts, then the Weighted Majority strategy boils down to following the prediction of the expert who was wrong fewer times in the past. Assume a deterministic implementation of this strategy – i.e., if the two experts are tied, then listen to the first one.

Our example will consist of (arbitrarily many) identical phases; each phase consists of two days and it looks as follows. On the first day, we are going to follow the first expert. The first expert is wrong and the second one is right. Therefore we "suffer". On the second day, we are going to follow the second expert. But the first expert is right and the second one is wrong. We suffer again. In total, we suffered twice and each expert suffered only once.

7 In this exercise we consider the Hedge algorithm and use the same notation as in the lecture notes. The average [external] regret of Hedge is defined as

$$\frac{\sum_{t\le T} \vec{p}^{(t)}\cdot \vec{m}^{(t)} - \min_i \sum_{t\le T} m_i^{(t)}}{T}$$

i.e., how much we "regret", on average over the days, compared to the best single strategy i.

7a If you knew the number of days T in advance, how would you set the parameter ϵ of Hedge to minimize the average external regret?

7b *(*)* Even if you do not know T in advance, describe a strategy that achieves roughly the same average external regret as in the case when T is known.

Hint: There is no need to redo the analysis from scratch. For example, you could consider restarting the algorithm each time you get to a day t of the form 4^i.

Solution:

1a From the lecture notes, the average external regret is at most

$$R(\epsilon) := \frac{\ln N}{\epsilon T} + \epsilon.$$

Minimizing this over ϵ, we get

$$\epsilon = \sqrt{\frac{\ln N}{T}} \quad\text{and}\quad R(\epsilon) = 2\sqrt{\frac{\ln N}{T}}.$$

1b We proceed as in the hint. More precisely, we divide the time period $[1,T]$ into phases where the i-th phase has length 4^i (except the last phase, which could be shorter) – that is, into phases $[1,4]$, $[5,20]$, $[21,84]$, etc. We restart the algorithm at the beginning of each such phase, by resetting all weights to 1 and setting $\epsilon = \epsilon_i = \frac{\sqrt{\ln N}}{2^i}$.

Now let us analyze the regret. (We assume for simplicity that the last, k-th phase also has full length 4^k.) For each phase i we have that the average regret is at most $2\frac{\sqrt{\ln N}}{2^i}$. Thus the total regret in phase i is at most $2\cdot 2^i\cdot\sqrt{\ln N}$. The average regret over all phases (i.e. over the time $T = \sum_{i=1}^k 4^i \approx 4^{k+1}/3$) is thus at most

$$\frac{\sum_{i=1}^k 2\cdot 2^i\cdot\sqrt{\ln N}}{T} \approx \frac{2\cdot 2^{k+1}\cdot\sqrt{\ln N}}{T} \approx \frac{2\cdot\sqrt{3}\cdot\sqrt{T}\cdot\sqrt{\ln N}}{T} = 2\sqrt{3}\cdot\sqrt{\frac{\ln N}{T}}.$$