2 Randomization

Now that we have established some basic ideas about online algorithms and competitive analysis, we introduce the concept of randomized online computation. Here, we basically allow an online algorithm to "flip a coin" from time to time and to continue its computation based on the outcome. The algorithms we studied so far were *deterministic*; this means that their actions were completely determined by the instance they were given. To distinguish these two different kinds of algorithms, we will now speak of deterministic online algorithms and *randomized online algorithms*. The output computed by a randomized online algorithm is not fully determined by the instance anymore, but there are different possible outputs, and therefore also different output qualities, when dealing with the same input. For a fixed input, we thus study the expected quality of a given randomized online algorithm. We again study worst-case inputs that are constructed by an adversary; the adversary model we use is called an *oblivious adversary*. Such an adversary is weak in the sense that it does not have any information about the random decisions made by the online algorithm.

We start by formally defining this model, and then continue by making some important observations on how to think about these algorithms. After that, we study randomized online algorithms for paging. An important result is that we can obtain an output that is exponentially better (in expectation) when computing using randomness instead of computing deterministically. Next, we learn a very central technique, known as *Yao's principle*, that allows us to prove lower bounds on the solution quality of randomized online algorithms by arguing about deterministic ones. First, we prove the statement for the special case where we have both a finite number of "coin tosses" and a finite class of instances; then, we generalize the results for an infinite setting. After that, we elaborate some interesting connections between online algorithms and two-person zero-sum games. Using what we have learned so far, we show a lower bound on the solution quality of randomized online algorithms for paging that asymptotically matches the preceding upper bound. Furthermore, as we are

D. Komm, *An Introduction to Online Computation*,
Texts in Theoretical Computer Science. An EATCS Series,
DOI 10.1007/978-3-319-42749-2_2

particularly interested in the number of random bits a randomized online algorithm has to use, we study so-called *barely random algorithms*, that is, algorithms that only use a constant number of random bits to create their output, independent of the input length. Last, we investigate deterministic and randomized online algorithms for a very generic online minimization problem that is encountered in many practical situations, namely the famous *ski rental problem*.

2.1 Introduction

The results of the previous chapter foreshadow the dilemma we are facing when computing online and considering worst-case instances; deterministic online algorithms may perform very poorly. Also, we have seen a possible way out by giving algorithms more power, for instance, by allowing resource augmentation or lookahead. However, for the latter idea, it turned out that it does not really help for paging with respect to competitive analysis. Of course, one might ask how realistic it is to perform such a worst-case analysis the way we do, or whether it is not sufficient to design algorithms that perform well for most instances. However, we want guarantees in the following sense. Our worst-case instances may seem artificial from a practical point of view, but maybe they are actually very natural for certain environments. In such a situation, there may exist a few hard inputs that always cause a given online algorithm to fail, although it performs a lot better on all other instances. The way we measure the solution quality of algorithms, such an algorithm is considered bad. In other words, we do not want that there are some instances of the given problem that *always* cause an online algorithm to perform poorly; even if our feeling is that these inputs do not occur very often.

However, in a randomized setting, it is acceptable if we design online algorithms for which we can guarantee that, for every given input, they perform well "on average." With this in mind, we use the following approach to enable online algorithms to obtain a higher output quality, that is, to overcome the drawback of not knowing the future, at least to some extent. We allow the online algorithm at hand to base its computations on randomness. So far, the output of a fixed algorithm was fully determined by its strategy and the input, which is why we call such algorithms *deterministic online algorithms*. We may think of *randomized online algorithms* as online algorithms that toss a coin from time to time and use the outcome of this coin flip to produce the output.

Formally, we need to introduce a *random source* which the algorithm may use; we neglect the potential difficulties of obtaining truly random numbers and simply suppose we have access to "real" randomness. In accordance with the model of Turing machines, we suppose that the random bits are read from a *tape* in a sequential manner. More precisely, the algorithm has access to a tape that is of unbounded length and that has an infinite binary string ψ written on it. Each bit of ψ is either 1 or 0 with a probability of $1/2$ each. Let us give a formal definition.

Definition 2.1 (Randomized Online Algorithm). Let Π be an online problem and let $I = (x_1, x_2, \ldots, x_n)$ be an input of Π. A *randomized online algorithm* RAND for Π computes the output $\text{RAND}^{\psi}(I) = (y_1, y_2, \ldots, y_n)$, where y_i depends on $\psi, x_1, x_2, \ldots, x_i$ and $y_1, y_2, \ldots, y_{i-1}$; ψ denotes a *binary string*, where every bit is chosen with probability $1/2$ to be either 0 or 1, and each choice is independent of all other bits.

So the output created by randomized online algorithms does not simply depend on the input, but also on ψ. In other words, for a fixed randomized online algorithm and a fixed input, there may be different outputs. Of course, this needs to be reflected in the way we measure the solution quality of such an algorithm and we cannot simply apply Definition 1.6. To this end, $\text{cost}(\text{RAND}(I))$ ($\text{gain}(\text{RAND}(I))$, respectively) is now a random variable that corresponds to the cost (gain, respectively) of RAND on I. We measure the solution quality of a randomized online algorithm by comparing its expected cost (expected gain, respectively), that is, $\mathbb{E}[\text{cost}(\text{RAND}(I))]$ ($\mathbb{E}[\text{gain}(\text{RAND}(I))]$, respectively), to that of an optimal offline solution for I. It is important to note that the optimal cost (gain, respectively) is a fixed value and not a random variable in this setting. Also, we are still interested in worst-case analysis when it comes to the input. This means that we do not consider a probability distribution over the inputs, but over contents of the random tape ψ. A randomized online algorithm is consistent for an online problem if it computes a feasible solution for every ψ and every input. In the remainder of this book, we omit ψ for the sake of an easier notation. These ideas are formalized in the following definition.

Definition 2.2 (Expected Competitive Ratio). Let Π be an online problem, let RAND be a consistent randomized online algorithm for Π, and let OPT be an optimal offline algorithm for Π. For $c \geq 1$, RAND is *c-competitive in expectation* for Π if there is a non-negative constant α such that, for every instance $I \in \mathcal{I}$,

$$\text{gain}(\text{OPT}(I)) \leq c \cdot \mathbb{E}[\text{gain}(\text{RAND}(I))] + \alpha$$

if Π is a maximization problem, or

$$\mathbb{E}[\text{cost}(\text{RAND}(I))] \leq c \cdot \text{cost}(\text{OPT}(I)) + \alpha$$

if Π is a minimization problem. If these inequalities hold with $\alpha = 0$, we call RAND *strictly c-competitive in expectation*. The *expected competitive ratio* of RAND is defined as

$$c_{\text{RAND}} := \inf\{c \geq 1 \mid \text{RAND is } c\text{-competitive in expectation for } \Pi\}\,.$$

If the expected competitive ratio of RAND is constant and the best that is achievable, we call RAND *strongly* c_{RAND}*-competitive in expectation*.

When proving lower bounds on the non-strict expected competitive ratio of some competitive randomized online algorithm, we can proceed analogously to deterministic online algorithms, that is, as we did in Theorems 1.2 and 1.3. More precisely, we again construct a set $\mathcal{I} = \{I_1, I_2, \ldots\}$ of instances and make sure that

$$\lim_{i \to \infty} \text{cost}(\textsc{Opt}(I_i)) = \infty$$

when dealing with a minimization problem, and

$$\lim_{i \to \infty} \text{gain}(\textsc{Opt}(I_i)) = \infty$$

when speaking about a maximization problem. All the other remarks we made in Section 1.2 about α can also be immediately transferred to the randomized setting.

For sure, one can easily think of alternative approaches to measure the solution quality of a randomized online algorithm; for instance, we could look at the competitive ratio that is achieved with some certain probability (which we will do in Section 2.7). Asking for an online algorithm to perform well in expectation is justified by the following reasoning.

- As mentioned above, if we consider a deterministic online algorithm Alg to be bad, there are infinitely many inputs for which Alg always produces some output of low quality, for instance, with large cost, compared to an optimal solution.
- For a good randomized online algorithm Rand, we require that it performs well on average. It may very well be the case that Rand produces a bad output for some instances, but only for some random decisions it makes. If the expected competitive ratio is small, however, we may hope that this does not happen too often for a fixed instance, or, if it does happen often, that Rand performs very well for the remaining random decisions. Thus, there are no hard instances that always cause Rand to fail, but, on each instance, Rand only fails sometimes. Of course, this "sometimes" comes in different flavors.

Similarly to deterministic online algorithms, we introduce an adversary that constructs the input in a malicious way. In accordance with the two points above, this adversary does not foresee the random decisions that are made by the algorithm; such an adversary is called an *oblivious adversary*. It knows all deterministic steps a given randomized online algorithm Rand makes. Thus, the adversary is aware of when Rand reads random bits, how many of them are used, and what is done depending on their values; but it has no clue about what these values will be. In the deterministic setting, we could think of the adversary as preparing the input in advance or as "reacting" to the algorithm's concrete answers. Here, we have to be more careful, because the adversary is not allowed to base the requests on the answers of the randomized online algorithm as they may depend on the values of the random bits; thus, the input has to be constructed in advance.

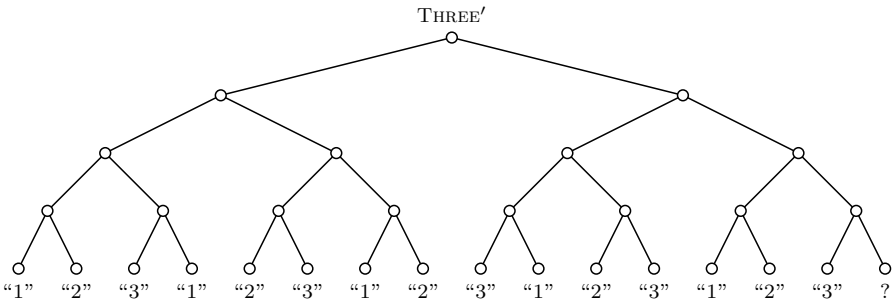


Figure 2.1. The randomized online algorithm THREE′.

A very important point which we need to consider when studying randomized online algorithms is the number of random decisions they make. An immediate problem is that we cannot always give an upper bound on the number of random bits used with absolute certainty. As an example, consider the simple randomized online algorithm THREE that does not get any input and only picks a number 1, 2, or 3 uniformly at random, that is, each with probability 1/3. How do we achieve this? Well, as THREE is a randomized online algorithm, it may read two bits from its random tape and simply map the outcome to “1,” “2,” or “3.” Thus, as a straightforward implementation, THREE outputs

$$\boxed{0}\boxed{0}\boxed{} \ldots \rightarrow \text{“1,”} \quad \boxed{0}\boxed{1}\boxed{} \ldots \rightarrow \text{“2,”} \quad \text{and} \quad \boxed{1}\boxed{0}\boxed{} \ldots \rightarrow \text{“3”}$$

and all these decisions are made with probability 1/4 as the bits on THREE's random tape are 0 or 1 with probability 1/2 each. Obviously, this is not sufficient, because every outcome is supposed to have a probability of 1/3. In other words, what do we do if the two random bits are both 1? Clearly, we cannot map this event to any of the three outputs; actually, there only seems to be one meaningful option. If this case occurs, we read the next two bits from the random tape and follow the same strategy as above.

Then, however, the outcome might again be that both bits are 1. In this case, THREE has to read another two bits, which happens with probability 1/16, and so on. In general, the probability that we still do not have a result after n tries is $1/4^n$, which means that it tends to 0 exponentially fast; still, this does not suffice to say that THREE terminates with absolute certainty after n tries.

As a matter of fact, it is easy to see that we can never give any guarantee in such a case. For a contradiction, suppose there is a natural number n' such that there is a randomized algorithm THREE′ that uses n' random bits and outputs “1,” “2,” or “3” uniformly at random and with absolute certainty. THREE′ also does not get any input and therefore its behavior is fully determined by the values of the first n' bits on its random tape. This means that THREE′ may act in at most $2^{n'}$ ways. If there are three outcomes and each is supposed to be chosen with the same probability, it

must be possible to distribute these outputs evenly among the $2^{n'}$ possible values of the random string. Clearly, this is not possible as no power of two is divisible by 3; this idea is sketched in Figure 2.1.

As we will see in Exercise 2.1, it also does not help to increase the alphabet size of the random tape of THREE′ as we would run into the same problems. So if we want to be really exact, we even need to model such a simple randomized algorithm as THREE′ as an infinitely branching tree, because, as Figure 2.1 suggests, if we only have a finite number of leaves, we cannot label them accurately.

Exercise 2.1. Suppose we change the model of randomized online algorithms such that the random tape does not contain bits, but symbols from an alphabet Σ with $|\Sigma| = \sigma$. Prove a result similar to the one for THREE′.

This behavior contradicts our notion as computer scientists of *algorithms*. As mentioned before, algorithms are those Turing machines that always halt, but here we are dealing with algorithms for which we cannot guarantee that they do. A possible way to cope with this problem is to analyze how many random bits we need in expectation; then we can apply Markov's inequality to derive a bound on the probability of taking t times as long, which is $1/t$. We can thus bound the number of random bits from above such that the probability that the considered randomized online algorithm uses more bits is very small. Then we will ignore this last uncertainty in a similar way that we ignore the probability of some hardware error during computation. Alternatively, we might be satisfied if the probabilities deviate very slightly from the ideal ones we assume theoretically. In what follows, we therefore assume that every randomized online algorithm that reads a finite input only uses a finite number of random bits.

To prove both lower and upper bounds on the expected competitive ratio of randomized online algorithms, we will often take the following point of view. Let $b\colon \mathbb{N} \to \mathbb{N}$ be a function that gives the maximum number of random binary decisions (figuratively speaking, the "coin tosses") of some given randomized online algorithm RAND for a given input length. As we have just discussed, for any natural number n, $b(n)$ is well defined, that is, $b(n)$ is some natural number as well. Therefore, we may say that RAND behaves in at most $2^{b(n)}$ different ways when reading some fixed input of length n. If we know RAND, we can compute $b(n)$ for every n. Furthermore, for every given instance of length n, we can simply simulate RAND for every possible random string of length $b(n)$, and thus study the behavior of the deterministic online algorithms we get as a consequence. The following observation is based on this idea and allows us to treat a randomized online algorithm as a set of deterministic algorithms.

Observation 2.1. *Every randomized online algorithm* RAND *that uses at most $b(n)$ random bits for inputs of length n can be viewed as a set* $\mathrm{strat}(\textsc{Rand}, n) = \{A_1, A_2, \ldots, A_{\ell(n)}\}$ *of $\ell(n) \leq 2^{b(n)}$ (not necessarily distinct) deterministic online algorithms on inputs of length n, from which* RAND *randomly chooses one.*

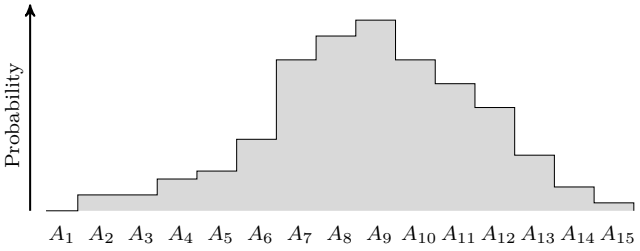


Figure 2.2. Schematic view of a randomized algorithm.

This point of view is shown schematically in Figure 2.2. To emphasize the context, we always write the names of deterministic strategies from the set $\mathrm{strat}(\textsc{Rand}, n)$ in italic letters. The probability distribution, on which RAND is based, is arbitrary; however, it must be possible to implement it with $b(n)$ random bits. We make another observation to again allow for an easier analysis. To this end, we note that every deterministic strategy is chosen with a probability that is a multiple of $1/2^{b(n)}$. Now let $a \in \{1, 2, \ldots, 2^{b(n)}\}$; instead of choosing a deterministic algorithm with a probability of $a/2^{b(n)}$, we can just choose a identical deterministic algorithms with a probability of $1/2^{b(n)}$ each. This leads to the following observation.

Observation 2.2. *Every randomized online algorithm* RAND *that uses at most $b(n)$ random bits for inputs of length n can be viewed as a set* $\mathrm{strat}(\textsc{Rand}, n) = \{A_1, A_2, \ldots, A_{2^{b(n)}}\}$ *of $2^{b(n)}$ (not necessarily distinct) deterministic online algorithms on inputs of length n, from which* RAND *chooses one uniformly at random with probability $1/2^{b(n)}$.*

Exercise 2.2. In Observation 2.2, we speak about "at most $b(n)$ random bits." But this means that the randomized online algorithm RAND may also use fewer bits. Don't we have to incorporate this fact? If so, the behavior of RAND can be different for all binary strings up to a length of $b(n)$. As a consequence, there would be

$$\sum_{i=0}^{b(n)} 2^i = 2^{b(n)+1} - 1$$

possibilities and not just $2^{b(n)}$. Argue why our reasoning is nevertheless correct, and why it is wrong to treat RAND as a set of $2^{b(n)+1} - 1$ deterministic algorithms.

Note that the point of view taken in Observations 2.1 and 2.2 is somewhat easier to see when speaking about offline algorithms instead of online algorithms. If we use the model of Turing machines, we can just modify a given randomized Turing machine RTM to another randomized Turing machine RTM$'$ in the following way. At the very beginning, RTM$'$ computes $b(n)$ and copies $b(n)$ bits from its random tape to an additional working tape. After that, RTM$'$ works exactly like RTM while

using its working tape the same way as RTM uses its random tape whereas RTM′ does not access its random tape anymore. In this case, no action was taken before the random bits were copied, and clearly a deterministic strategy is drawn randomly from a set of deterministic strategies.[1] For online algorithms, we cannot use such a constructive argument since the algorithm itself simply cannot compute $b(n)$ at the beginning (because n is not known in advance). However, we can compute this number when analyzing the algorithm, and this is all that is needed in this case. Moreover, there are randomized algorithms that make a constant number (with respect to the input length) of random decisions (so-called *barely random algorithms*, which we will consider shortly). These algorithms can indeed read all random bits before starting the computation. For such a randomized algorithm RAND that reads at most b random bits, we simply define the set of deterministic strategies by $\mathrm{strat}(\textsc{Rand}) = \{A_1, A_2, \ldots, A_{2^b}\}$.

Before we proceed to design randomized online algorithms for the paging problem, let us make some general statements. In Definition 2.2, we did not speak about *optimal* randomized online algorithms. In principle, there does not seem to be any immediate reason why we did not. So this would be a randomized online algorithm that, for *every* choice of random bits, produces an optimal solution for *every* input. However, if such an algorithm exists, we can immediately make an even stronger statement.

Theorem 2.1. *If, for some online problem* Π*, there is an optimal randomized online algorithm, then there also is an optimal deterministic online algorithm for* Π*.*

Proof. Suppose that RAND is an optimal randomized online for Π. Now suppose we simply design a deterministic online algorithm ALG that, whenever RAND reads a bit from its random tape, assumes that this bit is, without loss of generality, 1. Since RAND is optimal, it also has to be optimal in this (admittedly rather unlikely) case of "random" choices. Therefore, ALG is also optimal; moreover, we note that, for any fixed input length n, ALG is one of the strategies in $\mathrm{strat}(\textsc{Rand}, n)$. □

As a consequence, if we show that there is no optimal deterministic online algorithm for some online problem, we can conclude that there also is no optimal randomized one. It is very important to note that the above argument only holds for optimality, and that there is no analogous argument for 1-competitive randomized online algorithms.

Example 2.1. Consider the following online minimization problem, for which we have an *almost* optimal randomized online algorithm, but for which any deterministic online algorithm is very bad. Let $\varepsilon > 0$ be arbitrary but fixed such that $1/\varepsilon \in \mathbb{N}^+$. The input $I = (x_1, x_2, \ldots, x_n)$ starts with a request $x_1 = n$, where n is a multiple of $1/\varepsilon$, and we have $n \gg 1/\varepsilon$. Every online algorithm has to give an answer y_1 when reading x_1 such that $1 \leq y_1 \leq n$. The second request is x_2 with $1 \leq x_2 \leq n$. If

[1] Note that computing $b(n)$ may increase the running time. Another issue is that such a construction does not necessarily work if the memory RTM uses is bounded.

$y_1 = x_2$, the algorithm has to pay a penalty of 1 in time step T_2 and every subsequent time step that is a multiple of $1/\varepsilon$, independent of its further answers. If, however, $y_1 \neq x_2$, the algorithm only pays 1 in total. Therefore, OPT always has cost 1; both the cost and the competitive ratio of any deterministic online algorithm are exactly εn.

Conversely, we can easily design a randomized online algorithm RAND that gives an answer y_1 with $1 \leq y_1 \leq n$ after the first request uniformly at random, that is, every value is given as an answer with probability $1/n$. The expected cost of RAND is therefore

$$\frac{1}{n} \cdot \varepsilon n + \left(1 - \frac{1}{n}\right) \cdot 1 \leq 1 + \varepsilon \; .$$

It directly follows that RAND is 1-competitive in expectation according to Definition 2.2 for $\alpha = \varepsilon$. ♢

Of course, our argument only works because the input length n is revealed to the algorithm in the first time step.

Note that there is a natural restriction on how much randomization can help in an online setting that we do not encounter in offline problems. When we deal with randomized offline algorithms, we can make use of the principle of *amplification*. The idea is to run a randomized algorithm multiple times on the same input to increase the probability of obtaining a correct answer for some decision problem. Let us stick to offline problems for a second and consider an easy example.

Example 2.2. In Section 1.1, we briefly discussed the decision problem of determining whether a given number is prime. The *Solovay-Strassen algorithm* SOST answers this question with "yes" or "no" as follows. If the input I is prime, the correct output "yes" is given with probability 1; if the input is composite, however, the correct output is given with probability at least $1/2$. This is, of course, unsatisfying. If the output is "no," I must be composite; if the output is "yes," however, it may very well be the case that I is also composite.

Then again, we have an additional promise, namely SOST only gives the wrong answer "yes" with a probability of at most $1/2$ if I is composite. Now suppose we run SOST t times on the same input. The probability that I is composite and "yes" is still given as an answer all t times is at most $1/2^t$. On the other hand, if during these t runs, an answer "no" is observed once, we know that I is composite.

Hence, a correct answer is given with probability at least $1 - 1/2^t$ if I is composite, and with probability 1 if I is prime. For any $\varepsilon > 0$, we can choose the number t of runs of SOST such that

$$1 - \frac{1}{2^t} > 1 - \varepsilon \; .$$

To this end, it suffices to set

$$t := \left\lfloor \log_2\left(\frac{1}{\varepsilon}\right) \right\rfloor + 1$$

to only allow an arbitrarily small (constant) error in the worst case. ♢

When dealing with approximation algorithms for optimization problems, there is no simple "yes" or "no" and thus no right or wrong answer. Similarly to the expected competitive ratio of online algorithms, here, the expected approximation ratio is taken as a measurement. The nice thing is that through amplification a statement about some solution quality that is obtained in expectation also allows for a statement about which solution quality is obtained with an arbitrarily large (constant) probability. Suppose we are given a randomized offline r-approximation algorithm RAND for, say, a minimization problem Π. For any instance I of Π, let $d_I := r \cdot \text{cost}(\text{OPT}(I))$ where $\text{OPT}(I)$ is an optimal solution for I. Then, we have

$$\mathbb{E}[\text{cost}(\text{RAND}(I))] \leq d_I \ ,$$

for every I. For any given $\delta > 0$, we can bound the probability that RAND has a cost that is a factor of $1 + \delta$ larger than d_I by

$$\Pr[\text{cost}(\text{RAND}(I)) \geq (1+\delta) \cdot d_I] \leq \frac{1}{1+\delta}$$

due to Markov's inequality. If we again run RAND t times and choose the solution with minimum cost at the end, we can bound the probability that the result has a cost that is at least $(1+\delta) \cdot d_I$ in all t iterations by

$$\Pr[\text{cost}(\text{RAND}(I)) \geq (1+\delta) \cdot d_I \text{ in all } t \text{ tries}] \leq \left(\frac{1}{1+\delta}\right)^t \ ,$$

and we note that this upper bound does not depend on d_I. As in Example 2.2, we can choose the number of runs t of RAND such that

$$1 - \left(\frac{1}{1+\delta}\right)^t > 1 - \varepsilon \ .$$

A possible value is

$$t := \left\lfloor \log_{\frac{1}{1+\delta}} \left(\frac{1}{\varepsilon}\right) \right\rfloor + 1 \ .$$

In online computation, the technique of amplification unfortunately cannot be applied as online algorithms only have one shot to compute the output. Thus, a plain statement about the expected competitive ratio cannot be used in general to argue about the probabilities to compute a good or bad solution; for such statements we have to take a closer look.

2.2 A Randomized Online Algorithm for Paging

So far, we have learned that deterministic online algorithms are k-competitive at best for the paging problem. A general strategy to really obtain such a competitive

ratio is that followed by marking algorithms. These algorithms only differ in the way unmarked pages are replaced. In this section, we design a randomized marking algorithm RMark, which follows this concept and replaces the unmarked pages uniformly at random; the idea is shown in Algorithm 2.1.

```
mark all pages in the cache;                          // First page fault starts new phase
for every request x do
    if x is in the cache
        if x is unmarked
            mark x;
        output "0";
    else
        if there is no unmarked page
            unmark all pages in the cache;                          // Start new phase
        p := randomly chosen page among all unmarked cached pages;
        remove p and insert x at the old position of p;
        mark x;
        output "p";
end
```

Algorithm 2.1. Randomized online algorithm RMark for paging.

Our hope is that no oblivious adversary can force RMark to create an output that is too bad, because which pages reside in the cache at some given point in time partly depends on random decisions. For any $m \in \mathbb{N}^+$, let H_m denote the mth *harmonic number*, defined as

$$H_m := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{m} = \sum_{i=1}^{m} \frac{1}{i} ,$$

which we need to prove the following theorem.

Theorem 2.2. RMark *is strictly* $2H_k$*-competitive in expectation for paging.*

Proof. Let I denote the given input, and let there be N phases (defined by the k-phase partition by both RMark and Definition 1.8) in total. Note that the phases only depend on I and not on the random decisions of RMark. Recall that, before the first request is processed, RMark marks all pages in its cache; thus, the first page fault starts a new phase. For now, assume that all phases are complete.

Let us analyze a single phase P_j with $1 \leq j \leq N$. In this phase, exactly k distinct pages are requested; without loss of generality, we assume that no page is requested more than once during P_j. We call all pages that are already in the cache of RMark at the beginning of P_j "old." Conversely, pages that are not old and that are requested during P_j are called "new." For every new page, an unmarked old page is removed from the cache. Since new pages cause page faults anyway, it is

certainly a best strategy for the adversary to first request new pages and then old ones to increase the probability that some of the latter were removed before. Thus, we assume such an adversary for our analysis. Let l_j denote the number of new pages that are requested during P_j, leading to l_j page faults made by RMark. Since P_j is complete, after the first l_j requests, there are $k - l_j$ unmarked old pages requested during P_j, and each one is in the cache of RMark with a certain probability.

Consider the request for the first old page. In total, there are k unmarked old pages, some of which are still in the cache and some of which are removed at this point. More precisely, at the $(l_j + 1)$th time step of P_j, in which the first old page is requested, $k - l_j$ old pages have not yet been removed. Therefore, the probability that the page that is requested is still in the cache is

$$\frac{k - l_j}{k} .$$

After that, there are $k - 1$ unmarked old pages and $k - l_j - 1$ unmarked old pages that have not yet been removed from the cache. The probability that the page that is requested in the $(l_j + 2)$th time step is still in the cache is therefore

$$\frac{k - l_j - 1}{k - 1} ,$$

and in general we get that the ith requested old page is still in the cache of RMark with a probability of

$$\frac{k - l_j - (i - 1)}{k - (i - 1)} .$$

Conversely, this page is not in the cache with a probability of

$$1 - \frac{k - l_j - (i - 1)}{k - (i - 1)} = \frac{l_j}{k - (i - 1)} ,$$

which then also denotes the probability that RMark has cost 1 in the corresponding time step. Together with the cost l_j for the new pages which are requested at the beginning of P_j, we obtain a total expected cost for this phase of

$$\begin{aligned}
l_j + \sum_{i=1}^{k-l_j} \frac{l_j}{k - (i - 1)} &= l_j + l_j \left(\frac{1}{k} + \frac{1}{k-1} + \ldots + \frac{1}{l_j + 1} \right) \\
&= l_j + l_j \Bigg(\underbrace{\frac{1}{k} + \frac{1}{k-1} + \ldots + 1}_{H_k} - \Bigg(\underbrace{\frac{1}{l_j} + \frac{1}{l_j - 1} + \ldots + 1}_{H_{l_j}} \Bigg) \Bigg) \\
&= l_j (H_k - H_{l_j} + 1) \\
&\leq l_j H_k . \\
&\quad \textit{(since } l_j \geq 1 \textit{ and thus } H_{l_j} \geq 1 \textit{ (every phase starts with a new page))}
\end{aligned}$$

Now let $\text{cost}(\text{RMark}(P_j))$ denote a random variable that is equal to the cost of RMark in phase P_j. Due to linearity of expectation, the total cost of RMark is

$$\begin{aligned}\mathbb{E}[\text{cost}(\text{RMark}(I))] &= \mathbb{E}\left[\sum_{j=1}^{N} \text{cost}(\text{RMark}(P_j))\right] \\ &= \sum_{j=1}^{N} \mathbb{E}[\text{cost}(\text{RMark}(P_j))] \\ &\leq \sum_{j=1}^{N} H_k l_j \ .\end{aligned}$$

Next, we need to compute a lower bound on the cost of an optimal algorithm Opt. If we consider two consecutive phases P_{j-1} and P_j, at least $k + l_j$ distinct pages were requested. Hence, Opt has to make at least l_j page faults in these two phases. We can partition the phases in two different ways, either starting with phase P_1 or P_2, that is, we obtain

$$\underbrace{P_1,\ P_2}_{l_2 \text{ faults}},\ \underbrace{P_3,\ P_4}_{l_4 \text{ faults}},\ P_5,\ \ldots \quad \text{or} \quad P_1,\ \underbrace{P_2,\ P_3}_{l_3 \text{ faults}},\ \underbrace{P_4,\ P_5}_{l_5 \text{ faults}},\ \ldots \ .$$

Moreover, note that Opt causes l_1 page faults in P_1 since both RMark and Opt start with the same cache content. We get

$$\text{cost}(\text{Opt}(I)) \geq \sum_{j=1}^{\lfloor N/2 \rfloor} l_{2j} \quad \text{and} \quad \text{cost}(\text{Opt}(I)) \geq \sum_{j=1}^{\lceil N/2 \rceil} l_{2j-1}$$

and therefore

$$\begin{aligned}\text{cost}(\text{Opt}(I)) &\geq \max\left\{\sum_{j=1}^{\lfloor N/2 \rfloor} l_{2j},\ \sum_{j=1}^{\lceil N/2 \rceil} l_{2j-1}\right\} \\ &\geq \frac{1}{2}\left(\sum_{j=1}^{\lfloor N/2 \rfloor} l_{2j} + \sum_{j=1}^{\lceil N/2 \rceil} l_{2j-1}\right) \\ &= \sum_{j=1}^{N} \frac{1}{2} l_j \ .\end{aligned}$$

Consequently, we obtain an upper bound on the strict expected competitive ratio of RMark of

$$\sum_{j=1}^{N} H_k l_j \Bigg/ \sum_{j=1}^{N} \frac{1}{2} l_j = H_k \sum_{j=1}^{N} l_j \Bigg/ \frac{1}{2} \sum_{j=1}^{N} l_j = 2H_k \ .$$

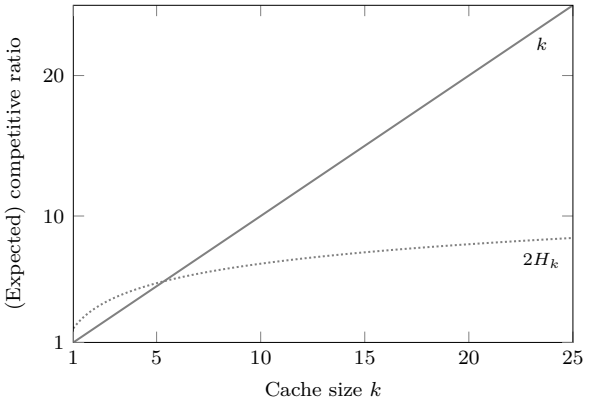


Figure 2.3. Comparison of the reachable competitive ratios of deterministic (solid) and randomized (dotted) online algorithms for paging.

To finish the proof, let us discuss the case where the last phase P_N is not complete. This means that, during P_N, there are $l_N \geq 1$ new pages requested and fewer than $k - l_N$ old pages. For the lower bound on $\text{cost}(\text{OPT}(I))$, nothing changes since it only incorporates the new pages of P_N. On the other hand, our upper bound on $\text{cost}(\text{RMARK}(P_N))$ decreases. □

To better understand how much randomization helps us for paging, we note that

$$H_m = \sum_{i=1}^{m} \frac{1}{i} \leq 1 + \int_1^m \frac{1}{i} \, \mathrm{d}i = \ln m + 1 \in \mathcal{O}(\log m) \, .$$

To sum up, we showed that no deterministic online algorithm is better than k-competitive, but there is a randomized online algorithm, which is $\mathcal{O}(\log k)$-competitive in expectation; see Figure 2.3. Hence, randomization allows for an exponential improvement of the asymptotic output quality in expectation.

In Section 2.5, we will show that, asymptotically, this is the best we can hope for when considering randomized online algorithms. However, before we show a lower bound on the expected competitive ratio of any randomized online algorithm for paging, we introduce a very general technique which often allows us to easily prove such bounds.

2.3 Yao's Principle

In this section, we study a connection between randomized and deterministic online algorithms. More precisely, the theorems below allow us to derive lower bounds on the expected competitive ratio of randomized online algorithms from bounds on the solution quality of deterministic online algorithms. It is of course not sufficient to

show that all deterministic strategies are bad for a given problem. In general, we can prove such a claim using different worst-case instances. However, since these deterministic algorithms may perform well for many other instances, there still might be a "good" randomized online algorithm that chooses between them (basically, that is the point of randomized computation). We have to show something stronger; namely, we have to take into account all possible randomized online algorithms, that is, all distributions over all deterministic online algorithms, and for any of these distributions, we have to find a hard instance. Obviously, this can be very challenging since we generally do not know anything about these distributions. Yao's principle offers a way to significantly simplify this task. It essentially says that it is sufficient to show that, for a fixed probability distribution over some instances, all deterministic strategies give bad results. Intuitively speaking, we will show that we can use a distribution over instances to derive statements for distributions over deterministic online strategies. Surprisingly, we are allowed to pick *one* distribution over instances and obtain a bound on *all* distributions over deterministic algorithms.

2.3.1 Finite Problems

In this subsection, we take a first step by proving a restricted version of the claim; more specifically, we assume that both the number of different instances of the given problem Π and the number of different deterministic algorithms that are consistent for Π are finite; we call such problems *finite online problems*. In this context, by "deterministic algorithms" we mean *generic algorithms*; that is, two algorithms that produce the same output for every instance are considered to be the same algorithm. The adversary chooses the instances according to a probability distribution $\Pr_{\text{ADV}}$ over a finite set $\mathcal{I}$ of size μ of instances while the given randomized online algorithm implements a probability distribution $\Pr_{\text{RAND}}$ over strat(RAND). We denote the expected values with respect to $\Pr_{\text{ADV}}$ or $\Pr_{\text{RAND}}$ by $\mathbb{E}_{\text{ADV}}$ or $\mathbb{E}_{\text{RAND}}$, respectively. To have a notation that is consistent with that for deterministic algorithms, we write $\mathbb{E}_{\text{ADV}}[\text{cost}(\text{ALG}(\mathcal{I}))]$ when we analyze the expected cost of ALG with respect to $\Pr_{\text{ADV}}$ for online minimization problems; likewise, we write $\mathbb{E}_{\text{ADV}}[\text{gain}(\text{ALG}(\mathcal{I}))]$ for online maximization problems. Let $A_1, A_2, \ldots, A_\ell$ denote *all* deterministic strategies that are consistent for Π; without loss of generality, we consider an online algorithm RAND with $\text{strat}(\text{RAND}) = \{A_1, A_2, \ldots, A_\ell\}$.

Note that, since there is only a finite number of instances, any deterministic online algorithm for Π is 1-competitive since we can choose the additive constant α from Definition 1.6 such that it is equal to the worst possible cost (best possible gain, respectively) over all algorithms and instances from $\mathcal{I}$; this has to be a fixed constant (assuming that there are no infinite costs or gains, which is true for all problems considered in this book). As a consequence, we will assume that α is 0 in the following statements, that is, we consider strict competitiveness.

We start by proving two claims in terms of minimizing some cost; combining them yields Yao's theorem for finite minimization problems.

Lemma 2.1. *Let Π be a finite online minimization problem, and let $\mathcal{I} = \{I_1, I_2, \ldots, I_\mu\}$ be a set of instances of Π; let $\Pr_{\text{ADV}}$ be a probability distribution over $\mathcal{I}$. If there is some constant $c \geq 1$ such that, for every deterministic online algorithm* ALG*, we have*

$$\mathbb{E}_{\text{ADV}}[\text{cost}(\text{ALG}(\mathcal{I}))] \geq c \cdot \mathbb{E}_{\text{ADV}}[\text{cost}(\text{OPT}(\mathcal{I}))] \,,$$

then, for every randomized online algorithm RAND*, there is an instance $I \in \mathcal{I}$ such that*

$$\mathbb{E}_{\text{RAND}}[\text{cost}(\text{RAND}(I))] \geq c \cdot \text{cost}(\text{OPT}(I)) \,.$$

Proof. Following Observations 2.1 and 2.2, RAND can be seen as a probability distribution $\Pr_{\text{RAND}}$, according to which one deterministic algorithm is chosen from a finite set strat(RAND). Now let us fix some arbitrary probability distribution $\Pr_{\text{ADV}}$ over $\mathcal{I}$, such that every instance $I \in \mathcal{I}$ gets chosen with probability $\Pr_{\text{ADV}}[I]$. We can then compute the expected cost of every deterministic online algorithm A_j with respect to $\Pr_{\text{ADV}}$ as

$$\mathbb{E}_{\text{ADV}}[\text{cost}(A_j(\mathcal{I}))] = \sum_{i=1}^{\mu} \Pr_{\text{ADV}}[I_i] \cdot \text{cost}(A_j(I_i)) \,. \tag{2.1}$$

The expected cost of RAND with respect to $\Pr_{\text{ADV}}$ is

$$\begin{aligned}
&\mathbb{E}_{\text{ADV}}[\mathbb{E}_{\text{RAND}}[\text{cost}(\text{RAND}(\mathcal{I}))]] \\
&\quad= \sum_{i=1}^{\mu} \Pr_{\text{ADV}}[I_i] \cdot \mathbb{E}_{\text{RAND}}[\text{cost}(\text{RAND}(I_i))] \\
&\quad= \sum_{i=1}^{\mu} \left(\Pr_{\text{ADV}}[I_i] \sum_{j=1}^{\ell} \Pr_{\text{RAND}}[A_j] \cdot \text{cost}(A_j(I_i)) \right) \\
&\quad= \sum_{i=1}^{\mu} \left(\sum_{j=1}^{\ell} \Pr_{\text{ADV}}[I_i] \cdot \Pr_{\text{RAND}}[A_j] \cdot \text{cost}(A_j(I_i)) \right) \\
&\quad= \sum_{j=1}^{\ell} \left(\sum_{i=1}^{\mu} \Pr_{\text{ADV}}[I_i] \cdot \Pr_{\text{RAND}}[A_j] \cdot \text{cost}(A_j(I_i)) \right) \\
&\qquad \textit{(by changing the order of summation)} \\
&\quad= \sum_{j=1}^{\ell} \left(\Pr_{\text{RAND}}[A_j] \sum_{i=1}^{\mu} \Pr_{\text{ADV}}[I_i] \cdot \text{cost}(A_j(I_i)) \right) \\
&\quad= \sum_{j=1}^{\ell} \Pr_{\text{RAND}}[A_j] \cdot \mathbb{E}_{\text{ADV}}[\text{cost}(A_j(\mathcal{I}))] \,. \qquad (2.2) \\
&\qquad \textit{(due to (2.1))}
\end{aligned}$$

Now suppose that every deterministic online algorithm has an expected cost that is at least c times larger than the expected optimal cost. Then we obtain

$$\begin{aligned}\mathbb{E}_{\text{ADV}}[\mathbb{E}_{\text{RAND}}[\text{cost}(\text{RAND}(\mathcal{I}))]] &= \sum_{j=1}^{\ell} \Pr_{\text{RAND}}[A_j] \cdot \mathbb{E}_{\text{ADV}}[\text{cost}(A_j(\mathcal{I}))] \\ &\geq \sum_{j=1}^{\ell} \Pr_{\text{RAND}}[A_j] \cdot c \cdot \mathbb{E}_{\text{ADV}}[\text{cost}(\text{OPT}(\mathcal{I}))] \\ &= c \cdot \mathbb{E}_{\text{ADV}}[\text{cost}(\text{OPT}(\mathcal{I}))] \,. \qquad (2.3) \\ &\quad \textit{(since the sum of all probabilities is 1)}\end{aligned}$$

At this point, we are not done yet as we considered a probability distribution over the set of inputs; this corresponds to a randomized adversary and this conflicts with our model where the adversary chooses one particular instance. However, we can easily "derandomize" the adversary by observing that (2.3) implies that there is a particular instance $I \in \mathcal{I}$ such that

$$\mathbb{E}_{\text{RAND}}[\text{cost}(\text{RAND}(I))] \geq c \cdot \text{cost}(\text{OPT}(I)) \,.$$

To see this, assume the contrary; that is, for every instance $I_i \in \mathcal{I}$ with $1 \leq i \leq \mu$, we have $\mathbb{E}_{\text{RAND}}[\text{cost}(\text{RAND}(I_i))] < c \cdot \text{cost}(\text{OPT}(I_i))$. Then it follows that $\Pr_{\text{ADV}}[I_i] \cdot \mathbb{E}_{\text{RAND}}[\text{cost}(\text{RAND}(I_i))] < c \cdot \Pr_{\text{ADV}}[I_i] \cdot \text{cost}(\text{OPT}(I_i))$, for every i, and hence

$$\sum_{i=1}^{\mu} \Pr_{\text{ADV}}[I_i] \cdot \mathbb{E}_{\text{RAND}}[\text{cost}(\text{RAND}(I_i))] < c \sum_{i=1}^{\mu} \Pr_{\text{ADV}}[I_i] \cdot \text{cost}(\text{OPT}(I_i)) \,,$$

which is equivalent to

$$\mathbb{E}_{\text{ADV}}[\mathbb{E}_{\text{RAND}}[\text{cost}(\text{RAND}(\mathcal{I}))]] < c \cdot \mathbb{E}_{\text{ADV}}[\text{cost}(\text{OPT}(\mathcal{I}))] \,,$$

and therefore directly contradicts (2.3). As a consequence, it follows that there must be at least one instance $I \in \mathcal{I}$ such that

$$\frac{\mathbb{E}_{\text{RAND}}[\text{cost}(\text{RAND}(I))]}{\text{cost}(\text{OPT}(I))} \geq c \,,$$

which gives the claimed lower bound on RAND's strict expected competitive ratio.□

Lemma 2.1 makes a statement on the ratio of the expected cost of online algorithms to the expected optimal cost. This does not immediately allow us to speak about the expectation of the ratio of the costs. Let ALG be some arbitrary deterministic online algorithm for some online problem, and suppose we are given some probability distribution $\Pr_{\text{ADV}}$ over a set $\mathcal{I}$ of inputs of this problem. Note that, in general, we have

$$\frac{\mathbb{E}_{\text{ADV}}[\text{cost}(\text{ALG}(\mathcal{I}))]}{\mathbb{E}_{\text{ADV}}[\text{cost}(\text{OPT}(\mathcal{I}))]} \neq \mathbb{E}_{\text{ADV}}\left[\frac{\text{cost}(\text{ALG}(\mathcal{I}))}{\text{cost}(\text{OPT}(\mathcal{I}))}\right] .$$

With little effort, however, we now prove the following lemma similarly to Lemma 2.1.

Lemma 2.2. *Let Π be a finite online minimization problem, and let $\mathcal{I}$ and $\Pr_{\text{ADV}}$ be as described above. If there is some constant $c \geq 1$ such that, for every deterministic online algorithm* ALG*, we have*

$$\mathbb{E}_{\text{ADV}}\left[\frac{\text{cost}(\text{ALG}(\mathcal{I}))}{\text{cost}(\text{OPT}(\mathcal{I}))}\right] \geq c\,,$$

then, for every randomized online algorithm RAND*, there is an instance $I \in \mathcal{I}$ such that*

$$\mathbb{E}_{\text{RAND}}[\text{cost}(\text{RAND}(I))] \geq c \cdot \text{cost}(\text{OPT}(I))\,.$$

Proof. Let us consider a randomized online algorithm RAND that implements a probability distribution $\Pr_{\text{RAND}}$ over a finite set strat(RAND) of deterministic online algorithms. Suppose that, for every deterministic online algorithm A_j, we have

$$\mathbb{E}_{\text{ADV}}\left[\frac{\text{cost}(A_j(\mathcal{I}))}{\text{cost}(\text{OPT}(\mathcal{I}))}\right] \geq c\,. \tag{2.4}$$

According to the definition of the expected value $\mathbb{E}_{\text{ADV}}$, we further have

$$\mathbb{E}_{\text{ADV}}\left[\frac{\text{cost}(A_j(\mathcal{I}))}{\text{cost}(\text{OPT}(\mathcal{I}))}\right] = \sum_{i=1}^{\mu} \Pr_{\text{ADV}}[I_i] \cdot \frac{\text{cost}(A_j(I_i))}{\text{cost}(\text{OPT}(I_i))}\,. \tag{2.5}$$

Likewise, for RAND we get

$$\begin{aligned}
&\mathbb{E}_{\text{ADV}}\left[\frac{\mathbb{E}_{\text{RAND}}[\text{cost}(\text{RAND}(\mathcal{I}))]}{\text{cost}(\text{OPT}(\mathcal{I}))}\right] \\
&\quad= \mathbb{E}_{\text{ADV}}\left[\frac{\sum_{j=1}^{\ell} \Pr_{\text{RAND}}[A_j] \cdot \text{cost}(A_j(\mathcal{I}))}{\text{cost}(\text{OPT}(\mathcal{I}))}\right] \\
&\quad= \sum_{i=1}^{\mu}\left(\Pr_{\text{ADV}}[I_i] \cdot \frac{\sum_{j=1}^{\ell} \Pr_{\text{RAND}}[A_j] \cdot \text{cost}(A_j(I_i))}{\text{cost}(\text{OPT}(I_i))}\right) \\
&\quad= \sum_{i=1}^{\mu}\left(\Pr_{\text{ADV}}[I_i] \sum_{j=1}^{\ell} \Pr_{\text{RAND}}[A_j] \cdot \frac{\text{cost}(A_j(I_i))}{\text{cost}(\text{OPT}(I_i))}\right) \\
&\quad= \sum_{i=1}^{\mu}\left(\sum_{j=1}^{\ell} \Pr_{\text{ADV}}[I_i] \cdot \Pr_{\text{RAND}}[A_j] \cdot \frac{\text{cost}(A_j(I_i))}{\text{cost}(\text{OPT}(I_i))}\right) \\
&\quad= \sum_{j=1}^{\ell}\left(\Pr_{\text{RAND}}[A_j] \sum_{i=1}^{\mu} \Pr_{\text{ADV}}[I_i] \cdot \frac{\text{cost}(A_j(I_i))}{\text{cost}(\text{OPT}(I_i))}\right)\,. \\
&\qquad\textit{(by again changing the order of summation)}
\end{aligned}$$

In the last line, we can plug in (2.5), which yields

$$\mathbb{E}_{\text{ADV}}\left[\frac{\mathbb{E}_{\text{RAND}}[\text{cost}(\text{RAND}(\mathcal{I}))]}{\text{cost}(\text{OPT}(\mathcal{I}))}\right] = \sum_{j=1}^{\ell} \Pr_{\text{RAND}}[A_j] \cdot \mathbb{E}_{\text{ADV}}\left[\frac{\text{cost}(A_j(\mathcal{I}))}{\text{cost}(\text{OPT}(\mathcal{I}))}\right]$$

and, due to (2.4), it follows that

$$\mathbb{E}_{\text{ADV}}\left[\frac{\mathbb{E}_{\text{RAND}}[\text{cost}(\text{RAND}(\mathcal{I}))]}{\text{cost}(\text{OPT}(\mathcal{I}))}\right] \geq \sum_{j=1}^{\ell} \Pr_{\text{RAND}}[A_j] \cdot c = c\ .$$

As before (in the proof of Lemma 2.1), from the definition of the expected value $\mathbb{E}_{\text{ADV}}$, we can immediately conclude that there is an instance $I \in \mathcal{I}$ such that

$$\frac{\mathbb{E}_{\text{RAND}}[\text{cost}(\text{RAND}(I))]}{\text{cost}(\text{OPT}(I))} \geq c$$

as we claimed. □

Combining Lemmata 2.1 and 2.2 results in Yao's principle for finite online minimization problems. Recall that $A_1, A_2, \ldots, A_\ell$ are all the deterministic online algorithms for Π, and that every randomized online algorithm for Π can be thought of as choosing between these algorithms.

Theorem 2.3 (Yao's Principle for Finite Min. Problems). *Let Π be a finite online minimization problem, and let $\mathcal{I}$, $\Pr_{\text{ADV}}$, and $\Pr_{\text{RAND}}$ be as described above. If there is some constant $c \geq 1$ such that*

$$\max\left\{\frac{\min_j(\mathbb{E}_{\text{ADV}}[\text{cost}(A_j(\mathcal{I}))])}{\mathbb{E}_{\text{ADV}}[\text{cost}(\text{OPT}(\mathcal{I}))]}, \min_j\left(\mathbb{E}_{\text{ADV}}\left[\frac{\text{cost}(A_j(\mathcal{I}))}{\text{cost}(\text{OPT}(\mathcal{I}))}\right]\right)\right\} \geq c\ ,$$

then, for every randomized online algorithm RAND *for Π, there is an instance $I \in \mathcal{I}$, such that*

$$\mathbb{E}_{\text{RAND}}[\text{cost}(\text{RAND}(I))] \geq c \cdot \text{cost}(\text{OPT}(I))\ .$$ □

Obviously, we would like to have a similar statement for online maximization problems. This is indeed possible; we leave the proof of the following statement as an exercise for the reader.

Theorem 2.4 (Yao's Principle for Finite Max. Problems). *Let Π be a finite online maximization problem, and let $\mathcal{I}$, $\Pr_{\text{ADV}}$, and $\Pr_{\text{RAND}}$ be as described above. If there is some constant $c \geq 1$ such that*

$$\max\left\{\frac{\mathbb{E}_{\text{ADV}}[\text{gain}(\text{OPT}(\mathcal{I}))]}{\max_j(\mathbb{E}_{\text{ADV}}[\text{gain}(A_j(\mathcal{I}))])}, \min_j\left(\left(\mathbb{E}_{\text{ADV}}\left[\frac{\text{gain}(A_j(\mathcal{I}))}{\text{gain}(\text{OPT}(\mathcal{I}))}\right]\right)^{-1}\right)\right\} \geq c\ ,$$

then, for every randomized online algorithm RAND *for Π, there is an instance $I \in \mathcal{I}$ such that*

$$\text{gain}(\text{OPT}(I)) \geq c \cdot \mathbb{E}_{\text{RAND}}[\text{gain}(\text{RAND}(I))]\ .$$

Exercise 2.3. Prove Theorem 2.4.

Hint. For the second argument of the maximum function, consider the reciprocal of the strict competitive ratio as a measurement.

2.3.2 Infinite Problems

So far, our proofs only work if both strat(RAND) and $\mathcal{I}$ are finite. Indeed, if we assume they are infinitely large, we run into some problems. In this case, it is not justified anymore to ignore the additive constant α when computing a bound on the competitive ratio of the given online algorithms. This means that we now do not speak about the strict competitive ratio, and therefore we cannot simply show that there is one hard instance, but we have to prove the existence of infinitely many. Second, the expected values cannot necessarily be expressed as finite sums anymore. However, for many online problems, the number of instances of a given length is finite (or at least we can use finite subsets of instances of a given length for our lower-bound argument); for instance, for paging, for a fixed n, there is just a finite number of different sequences that request one of m pages in every time step, namely m^n. The number of possible answers to a given request is usually also finite, and thus, for a fixed n, there is also a finite number of *generic algorithms*. We again assume that strat(RAND, n) consists of all such algorithms. This way, we are able to apply Yao's principle for finite problems from the last subsection, yet implicitly define an infinite set of instances $\mathcal{I}$ and speak about an infinite set of algorithms. Moreover, if we can prove that the expected gain (when considering maximization problems) or the expected cost (when considering minimization problems) of an optimal offline algorithm tends to infinity, we can cover the case that α is positive; this is basically the same idea as used for Theorems 1.2 and 1.3. Subsequently, $\mathcal{I}_1, \mathcal{I}_2, \ldots$ denote infinitely many finite sets of instances of the given problem such that all instances in $\mathcal{I}_i$ have the same length, and, for every $I \in \mathcal{I}_i$ and $I' \in \mathcal{I}_{i+1}$, we have $|I| < |I'|$. Furthermore, for every input length n considered, there is only a finite number of deterministic online algorithms $A_1, A_2, \ldots, A_{\ell(n)}$. The adversary chooses the instances from $\mathcal{I}_i$ according to a probability distribution $\Pr_{\text{ADV},i}$; the expected value is denoted by $\mathbb{E}_{\text{ADV},i}$. Now suppose we can construct $\mathcal{I}_1, \mathcal{I}_2, \ldots$ as above for an online minimization problem Π and show that

$$\max\left\{\frac{\min_j(\mathbb{E}_{\text{ADV},i}[\text{cost}(A_j(\mathcal{I}_i))])}{\mathbb{E}_{\text{ADV},i}[\text{cost}(\text{OPT}(\mathcal{I}_i))]}, \min_j\left(\mathbb{E}_{\text{ADV},i}\left[\frac{\text{cost}(A_j(\mathcal{I}_i))}{\text{cost}(\text{OPT}(\mathcal{I}_i))}\right]\right)\right\} \geq c\,, \quad (2.6)$$

for every $i \in \mathbb{N}^+$. Applying either Lemma 2.1 or Lemma 2.2, there is an infinite set $\mathcal{I} = \{I_1, I_2, \ldots\}$ with $I_i \in \mathcal{I}_i$ and $|I_i| < |I_{i+1}|$ such that

$$\frac{\mathbb{E}_{\text{RAND}}[\text{cost}(\text{RAND}(I_i))]}{\text{cost}(\text{OPT}(I_i))} \geq c\,.$$

If $c = c(n)$ is an unbounded increasing function, then, by the same reasoning as for the deterministic case (see Section 1.2), RAND cannot be c'-competitive for any c' with $c'(n) \in o(c(n))$.

But what happens if c is constant? Suppose we can also show

$$\lim_{i\to\infty} \text{cost}(\textsc{Opt}(I_i)) = \infty \ ,$$

for any choice of $I_i \in \mathcal{I}_i$. This implies that no randomized online algorithm for Π is $(c-\varepsilon)$-competitive in expectation, for any $\varepsilon > 0$; the arguments are exactly the same as in the proof of Theorem 1.2.

Analogous statements can easily be made for infinite maximization problems, which is left as an exercise for the reader. Next, we formulate an infinite version of Yao's principle where we bound from below the expected cost of an optimal offline algorithm. However, the proof (which uses a contradiction) only works for the first argument of the max-expression of (2.6).

Theorem 2.5 (Yao's Principle for Infinite Min. Problems). *Let Π be an online minimization problem, and let $\mathcal{I}_1, \mathcal{I}_2, \ldots$ and $\Pr_{\text{Adv},i}$ be as described above. If there is some constant $c \geq 1$ such that*

(i) $\dfrac{\min_j(\mathbb{E}_{\text{Adv},i}[\text{cost}(A_j(\mathcal{I}_i))])}{\mathbb{E}_{\text{Adv},i}[\text{cost}(\textsc{Opt}(\mathcal{I}_i))]} \geq c$, *for every $i \in \mathbb{N}^+$, and*

(ii) $\lim_{i\to\infty} \mathbb{E}_{\text{Adv},i}[\text{cost}(\textsc{Opt}(\mathcal{I}_i))] = \infty$,

then there is no randomized online algorithm for Π that is $(c-\varepsilon)$-competitive in expectation, for any $\varepsilon > 0$.

Proof. For a contradiction, suppose that both conditions (i) and (ii) are true, but there still is a randomized online algorithm Rand that is $(c-\varepsilon)$-competitive in expectation for Π, where $\varepsilon > 0$. In particular, there is a constant α such that, for every $i \in \mathbb{N}^+$ and every instance $I \in \mathcal{I}_i$, we have that

$$\mathbb{E}_{\text{Rand}}[\text{cost}(\textsc{Rand}(I))] \leq (c-\varepsilon) \cdot \text{cost}(\textsc{Opt}(I)) + \alpha \ .$$

Since this inequality holds for any $I \in \mathcal{I}_i$, we can immediately speak about the expected value with respect to $\Pr_{\text{Adv},i}$, yielding

$$\mathbb{E}_{\text{Adv},i}[\mathbb{E}_{\text{Rand}}[\text{cost}(\textsc{Rand}(\mathcal{I}_i))]] \leq (c-\varepsilon) \cdot \mathbb{E}_{\text{Adv},i}[\text{cost}(\textsc{Opt}(\mathcal{I}_i))] + \alpha \ . \tag{2.7}$$

The same calculations as in the proof of Lemma 2.1 (more precisely, (2.2)) yield

$$\begin{aligned}\mathbb{E}_{\text{Adv},i}[\mathbb{E}_{\text{Rand}}[\text{cost}(\textsc{Rand}(\mathcal{I}_i))]] &= \sum_{j=1}^{\ell} \Pr_{\text{Rand}}[A_j] \cdot \mathbb{E}_{\text{Adv},i}[\text{cost}(A_j(\mathcal{I}_i))] \\ &= \mathbb{E}_{\text{Rand}}[\mathbb{E}_{\text{Adv},i}[\text{cost}(\textsc{Rand}(\mathcal{I}_i))]] \ ,\end{aligned}$$

and thus it follows that

$$\mathbb{E}_{\text{Rand}}[\mathbb{E}_{\text{Adv},i}[\text{cost}(\textsc{Rand}(\mathcal{I}_i))]] \leq (c-\varepsilon) \cdot \mathbb{E}_{\text{Adv},i}[\text{cost}(\textsc{Opt}(\mathcal{I}_i))] + \alpha \tag{2.8}$$

has to be satisfied.

Since $\Pr_{\text{Adv},i}$ is fixed, there is a "best" random choice for RAND, that is,

$$\min_j(\mathbb{E}_{\text{Adv},i}[\text{cost}(A_j(\mathcal{I}_i))]) \leq \mathbb{E}_{\text{Rand}}[\mathbb{E}_{\text{Adv},i}[\text{cost}(\text{RAND}(\mathcal{I}_i))]] \; . \tag{2.9}$$

The following steps are essentially the same as in the proof of Theorem 1.2. With (2.8) and (2.9), we get

$$\min_j(\mathbb{E}_{\text{Adv},i}[\text{cost}(A_j(\mathcal{I}_i))]) \leq (c-\varepsilon) \cdot \mathbb{E}_{\text{Adv},i}[\text{cost}(\text{OPT}(\mathcal{I}_i))] + \alpha \; ,$$

which is equivalent to (assuming that the expected optimal cost is not zero)

$$\frac{\min_j(\mathbb{E}_{\text{Adv},i}[\text{cost}(A_j(\mathcal{I}_i))])}{\mathbb{E}_{\text{Adv},i}[\text{cost}(\text{OPT}(\mathcal{I}_i))]} - \frac{\alpha}{\mathbb{E}_{\text{Adv},i}[\text{cost}(\text{OPT}(\mathcal{I}_i))]} \leq c-\varepsilon \; . \tag{2.10}$$

Due to (i), the first term of (2.10) is at least c. Additionally, (ii) implies that there are infinitely many sets of instances such that the second term of (2.10) is smaller than ε. Thus, for infinitely many $\mathcal{I}_i$, we get a contradiction. □

The proof of the complementing statement for infinite maximization problems is also left as an exercise.

Theorem 2.6 (Yao's Principle for Infinite Max. Problems). *Let Π be an online maximization problem, and let $\mathcal{I}_1, \mathcal{I}_2, \ldots$ and $\Pr_{\text{Adv},i}$ be as described above. If there is some constant $c \geq 1$ such that*

(i) $\dfrac{\mathbb{E}_{\text{Adv},i}[\text{gain}(\text{OPT}(\mathcal{I}_i))]}{\max_j(\mathbb{E}_{\text{Adv},i}[\text{gain}(A_j(\mathcal{I}_i))])} \geq c$, *for every $i \in \mathbb{N}^+$, and*

(ii) $\lim\limits_{i\to\infty} \mathbb{E}_{\text{Adv},i}[\text{gain}(\text{OPT}(\mathcal{I}_i))] = \infty$,

then there is no randomized online algorithm for Π that is $(c-\varepsilon)$-competitive in expectation, for any $\varepsilon > 0$.

Exercise 2.4. For online maximization problems, we can make observations similar to those preceding Theorem 2.5. Discuss them.

Exercise 2.5. Prove Theorem 2.6.

⋆2.3.3 Unbounded Problems

In this subsection, we even go a step further and allow infinitely many sets of infinitely many instances and consistent algorithms. We even allow uncountably many instances and algorithms in each set, although this is somewhat "too general" as there is only a countable number of algorithms (to have an easier notation, we still speak of the jth algorithm A_j also in this case). As already mentioned in the

previous subsection, in contrast to the finite case, the expected values may have to be expressed as infinite sums; even worse, we may be dealing with continuous random variables, whose expected values are given by integrals rather than sums. To study such a setting, we use the following variation of Tonelli's theorem, which we state without a proof; it will allow us to prove the "unbounded version" of Yao's principle by changing the order of taking the expectation (which so far simply meant to change the order of summation).

Theorem 2.7 (Tonelli's Theorem). *Let $\mathcal{X}$ and $\mathcal{Y}$ be probability spaces, and let $f\colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$ be a non-negative measurable function. Then, we have*

$$\int_{\mathcal{Y}} \left(\int_{\mathcal{X}} f(x,y)\,\mathrm{d}x \right) \mathrm{d}y = \int_{\mathcal{X}} \left(\int_{\mathcal{Y}} f(x,y)\,\mathrm{d}y \right) \mathrm{d}x \ . \qquad \square$$

Since we are speaking about infinite sets, in the following we need to replace the maximum and minimum functions by supremum and infimum functions, respectively. Unlike in the previous two subsections, we use the limit inferior or limit superior of the fractions.

Moreover, we prove a more general claim by not grouping instances by their lengths, but by any suitable parameter. Consider (countably) infinitely many sets $\mathcal{I}_1, \mathcal{I}_2, \ldots$ of instances of an online minimization problem Π; each $\mathcal{I}_i$ may be of (even uncountably) infinite size. As always, let $\Pr_{\text{Adv},i}$ be a probability distribution over $\mathcal{I}_i$. The following proof is obtained by slightly modifying that of Theorem 2.5.

Theorem 2.8 (Yao's Principle for Unbounded Min. Problems). *Let Π be an online minimization problem, and let $\mathcal{I}_1, \mathcal{I}_2, \ldots$ and $\Pr_{\text{Adv},i}$ be as described above. If there is some constant $c \geq 1$ such that*

(i) $\displaystyle\liminf_{i\to\infty} \left(\frac{\inf_j(\mathbb{E}_{\text{Adv},i}[\text{cost}(A_j(\mathcal{I}_i))])}{\mathbb{E}_{\text{Adv},i}[\text{cost}(\text{Opt}(\mathcal{I}_i))]} \right) \geq c$ *and*

(ii) $\displaystyle\limsup_{i\to\infty} (\mathbb{E}_{\text{Adv},i}[\text{cost}(\text{Opt}(\mathcal{I}_i))]) = \infty$,

then there is no randomized online algorithm for Π that is $(c-\varepsilon)$-competitive in expectation, for any $\varepsilon > 0$.

Proof. For a contradiction, suppose that both conditions (i) and (ii) are true, but there still is a randomized online algorithm Rand that is $(c-\varepsilon)$-competitive in expectation for Π, where $\varepsilon > 0$. By the same arguments as in the proof of Theorem 2.5, we get

$$\mathbb{E}_{\text{Adv},i}[\mathbb{E}_{\text{Rand}}[\text{cost}(\text{Rand}(\mathcal{I}_i))]] \leq (c-\varepsilon) \cdot \mathbb{E}_{\text{Adv},i}[\text{cost}(\text{Opt}(\mathcal{I}_i))] + \alpha \ ,$$

and, together with Theorem 2.7 (the cost and the probabilities are always non-negative and measurable), it follows that

$$\mathbb{E}_{\text{Rand}}[\mathbb{E}_{\text{Adv},i}[\text{cost}(\text{Rand}(\mathcal{I}_i))]] \leq (c-\varepsilon) \cdot \mathbb{E}_{\text{Adv},i}[\text{cost}(\text{Opt}(\mathcal{I}_i))] + \alpha \ . \quad (2.11)$$

Again using the same arguments as in the proof of Theorem 2.5 gives (assuming that the expected optimal cost is not zero)

$$\frac{\inf_j(\mathbb{E}_{\text{ADV},i}[\text{cost}(A_j(\mathcal{I}_i))])}{\mathbb{E}_{\text{ADV},i}[\text{cost}(\text{OPT}(\mathcal{I}_i))]} - \frac{\alpha}{\mathbb{E}_{\text{ADV},i}[\text{cost}(\text{OPT}(\mathcal{I}_i))]} \leq c - \varepsilon \ . \tag{2.12}$$

As a consequence of (i), there is an i_0 such that the first term of (2.12) is larger than c, for any $i \geq i_0$. Additionally, (ii) implies that there is an infinite increasing sequence $i_1, i_2, \ldots$, where $i_1 \geq i_0$, such that

$$\lim_{l\to\infty}\left(\frac{\alpha}{\mathbb{E}_{\text{ADV},i_l}[\text{cost}(\text{OPT}(\mathcal{I}_{i_l}))]}\right) = 0 \ .$$

Thus, for infinitely many $\mathcal{I}_i$, we get a contradiction. □

One possible way to apply Theorem 2.8 is to define the sets to contain instances of the same length; however, we will use another partitioning of instances in Section 2.5, where we use Yao's principle to obtain a lower bound on the expected competitive ratio of every randomized online algorithm for paging.

Note that, for the last step of our argumentation in the proof above, we needed the fact that (i) speaks about the limit inferior. If, instead, we had used the limit superior (which is a weaker assumption), we could not argue the same way. In this case, we only know that the first term of (2.12) is larger than c for infinitely many i and the second one tends to 0 for a sequence of infinitely many i; however, the intersection of these two sets of input lengths is not necessarily infinite.

For maximization problems, we can prove an analogous statement. We again leave the proof to the reader.

Theorem 2.9 (Yao's Principle for Unbounded Max. Problems). *Let Π be an online maximization problem, and let $\mathcal{I}_1, \mathcal{I}_2, \ldots$ and $\text{Pr}_{\text{ADV},i}$ be as described above. If there is some constant $c \geq 1$ such that*

(i) $\liminf_{i\to\infty}\left(\frac{\mathbb{E}_{\text{ADV},i}[\text{gain}(\text{OPT}(\mathcal{I}_i))]}{\sup_j(\mathbb{E}_{\text{ADV},i}[\text{gain}(A_j(\mathcal{I}_i))])}\right) \geq c$ *and*

(ii) $\limsup_{i\to\infty}(\mathbb{E}_{\text{ADV},i}[\text{gain}(\text{OPT}(\mathcal{I}_i))]) = \infty$,

then there is no randomized online algorithm for Π that is $(c-\varepsilon)$-competitive in expectation, for any $\varepsilon > 0$.

Exercise 2.6. Prove Theorem 2.9.

Before we apply Yao's principle, we will have a look at the game between a randomized online algorithm and the adversary from a different perspective. Our main reason to do this is to present an alternative proof of Yao's principle for finite minimization problems.

2.4 Another Point of View: Game Theory

Again, we consider a randomized online algorithm RAND and an adversary that tries to force RAND into creating an output that is as bad as possible compared to an optimal solution. Throughout this section, we will only consider finite online minimization problems; let Π be such a problem. RAND makes $b \in \mathbb{N}^+$ binary random decisions and, according to Observation 2.1, therefore picks one out of ℓ deterministic algorithms $A_1, A_1, \ldots, A_\ell$ from a set strat(RAND), where $\ell \leq 2^b$. Conversely, the adversary can choose an arbitrary input $I_1, I_2, \ldots, I_\mu$ from a set $\mathcal{I}$, which RAND then has to work on. We assume that ℓ and μ are arbitrarily large, but finite. For the same reasons as in Subsection 2.3.1, we thus assume that the additive constant α is 0.

In what follows, we call $A_1, A_2, \ldots, A_\ell$ the *strategies* of RAND and $I_1, I_2, \ldots, I_\mu$ the *strategies* of the adversary. Moreover, for all i and j with $1 \leq i \leq \mu$ and $1 \leq j \leq \ell$, we define the *strict performance* $c_{i,j}$ of A_j on I_i as

$$c_{i,j} := \frac{\text{cost}(A_j(I_i))}{\text{cost}(\text{OPT}(I_i))} ,$$

where $\text{OPT}(I_i)$ is an optimal solution for I_i. With these values, we can construct the following matrix $\mathcal{M}$.

	A_1	A_2	A_3	$\ldots$
I_1	$c_{1,1}$	$c_{1,2}$	$c_{1,3}$	$\ldots$
I_2	$c_{2,1}$	$c_{2,2}$	$c_{2,3}$	
I_3	$c_{3,1}$	$c_{3,2}$	$c_{3,3}$	
$\vdots$	$\vdots$			$\ddots$

For the moment, let us stick to the deterministic case, that is, RAND chooses one strategy with probability 1. We may think of RAND and the adversary as two players in a game. RAND chooses a column j from $\mathcal{M}$, which corresponds to a strategy A_j, and the adversary chooses a row i corresponding to an input I_i. In particular, RAND wants to choose j such that, whatever row i is chosen by the adversary, the value $c_{i,j}$ is as small as possible. Equivalently, we may say that RAND wants to maximize the value $-c_{i,j}$. Conversely, the adversary obviously wants to maximize the value $c_{i,j}$. We call $c_{i,j}$ and $-c_{i,j}$ the *payoff* of the adversary or RAND, respectively. Since, for a fixed column and row, RAND has a payoff which is exactly the negated payoff of the adversary, we call this game a *zero-sum game*; more specifically, as there are two players involved, such a game is called a *two-person zero-sum game*. In what follows, we say that RAND wants to minimize the adversary's payoff, which is, as just noted, the same as if RAND wants to maximize its own payoff, but yields a more intuitive point of view for our investigations.

We call the adversary the *row player* and RAND the *column player*, and we may assume that both players know the set of strategies of the opponent and therefore $\mathcal{M}$.

Example 2.3. Suppose the players have two strategies each, and consider the following matrix $\mathcal{M}_1$.

	A_1	A_2
I_1	1	2
I_2	2	1

The players now study all possible courses and payoffs according to $\mathcal{M}_1$, and then pick a best strategy based on their investigations, and we assume that they both have to reveal the chosen strategy at the same time. However, it turns out that one of them would always want to change its strategy. Suppose, for instance, that both have chosen their first strategy, that is, A_1 and I_1. In this case, the adversary has a payoff of $c_{1,1} = 1$ and RAND has a payoff of -1. Since the adversary knows the matrix $\mathcal{M}_1$, it wants to change its strategy to I_2 instead. In this case, RAND has a payoff of $-c_{1,2} = -2$, which is why it wants to change its strategy to A_2. If it does this, the adversary has an *incentive* to again change its strategy, and we see that we go round in circles; no matter which strategies are chosen, one player always has an advantage if it changes its strategy. We can illustrate this situation as follows, where the arrows point to the maximum values of the columns and the minimum values of the rows.

	A_1	A_2
I_1	←	
I_2	↓	→ ↑

♢

Let us give a different example that leads to a completely different situation.

Example 2.4. Consider the following matrix $\mathcal{M}_2$.

	A_1	A_2	A_3	A_4
I_1	7	2	3	1
I_2	1	4	5	4
I_3	10	9	6	9
I_4	6	3	2	11
I_5	12	1	2	5

In this example, there is a pair of strategies which, if chosen, lead to a situation in which neither of the players has an incentive to change its strategy, given that the other player sticks to its strategy. Specifically, this is the case if RAND chooses A_3 and the adversary chooses I_3; for all other choices, at least one player would want to change its strategy, as the following illustration shows.

	A_1	A_2	A_3	A_4
I_1				→
I_2	←	↓		
I_3		↑	→	←
I_4			←	↓ ↑
I_5	↓	←		

In such a case, we say that the game is at an *equilibrium*, and we call $c_{3,3}$ the *value of the game*. We observe the following two facts.

1. If RAND chooses the strategy A_3, it can be sure that the adversary's payoff is not larger than 6; for instance, if RAND chose A_1 instead, the payoff of the adversary could be significantly larger, namely 12.
2. Conversely, if the adversary chooses the strategy I_3, it can be sure that its payoff is at least 6. If it chose I_5, its payoff could be a lot smaller, namely 1.

It follows that, in the above case, we can describe the strategies of both players that lead to an equilibrium as follows.

- RAND chooses its strategy A_{j^*} such that the maximum value over all entries in this column is as small as possible, that is,

  $$j^* = \arg\min_j \{\max_i \{c_{i,j}\}\} ,$$

  and we set

  $$v_{\text{RAND}} := \max_i \{c_{i,j^*}\} .$$

- On the other hand, the adversary chooses its strategy I_{i^*} such that the minimum entry over the columns in this row is as large as possible, that is,

  $$i^* = \arg\max_i \{\min_j \{c_{i,j}\}\} ,$$

  and we set

  $$v_{\text{ADV}} := \min_j \{c_{i^*,j}\} .$$

As we have just seen, we have

$$v_{\text{ADV}} = v_{\text{RAND}} = c_{3,3}$$

in this example. If i^* and j^* are played, neither player can obtain a larger payoff by changing its strategy (given that the other one does not change its strategy). ♢

As Example 2.4 suggests, the value of the game can always be computed as above if the given game has an equilibrium. However, as Example 2.3 shows, this does not hold in general, although the values v_{RAND} and v_{ADV} always exist; we merely know that

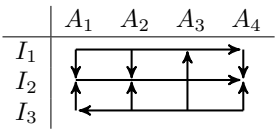
$$v_{\text{ADV}} \leq v_{\text{RAND}}$$

is always true. Let us give one last example.

Example 2.5. Consider the following matrix $\mathcal{M}_3$.

	A_1	A_2	A_3	A_4
I_1	6	5	8	4
I_2	7	6	2	7
I_3	1	3	3	2

We can illustrate $\mathcal{M}_3$ as follows.



Suppose both players act according to the principle from Example 2.4. RAND can guarantee that the payoff of the adversary is at most 6 by playing A_2. Obviously, for the adversary it would then be best to choose the strategy I_2. However, in this case, RAND would want to change its strategy and play A_3 instead. On the other hand, by choosing the strategy I_1, the adversary can always guarantee that its payoff is at least 4. In this case, RAND would want to choose A_4 and the adversary has an incentive to change its strategy. As a consequence, the game is not at an equilibrium if the strategies I_1 and A_2 are chosen. ♢

Such a procedure does not seem very promising for analyzing deterministic algorithms, but this is not what we want anyway.[2] We want to speak about randomized online algorithms and, in what follows, let us also assume that the adversary chooses a strategy at random. If we transfer this model to our game, we are dealing with the following situation. RAND uses a probability distribution $\Pr_{\text{RAND}}\colon \text{strat}(\text{RAND}) \to [0,1]$ over the columns of a given matrix $\mathcal{M}$. We denote the probability $\Pr_{\text{RAND}}[A_j]$ that RAND chooses the strategy A_j by $q_{\text{RAND},j}$; the sequence $q_{\text{RAND}} = (q_{\text{RAND},1}, q_{\text{RAND},2}, \ldots, q_{\text{RAND},\ell})$ is called the *mixed strategy* of RAND. Conversely, the adversary uses a probability distribution $\Pr_{\text{ADV}}\colon \mathcal{I} \to [0,1]$ over the rows of $\mathcal{M}$. The probability $\Pr_{\text{ADV}}[I_i]$ that the adversary chooses the strategy I_i is denoted by $q_{\text{ADV},i}$; the sequence $q_{\text{ADV}} = (q_{\text{ADV},1}, q_{\text{ADV},2}, \ldots, q_{\text{ADV},\mu})$ is called the mixed strategy of the adversary. To distinguish these strategies from the deterministic setting above, we call the former strategies *pure strategies*. Since we are now dealing with a randomized setting, the payoff is modeled by a random variable $C\colon \mathcal{I} \times \text{strat}(\text{RAND}) \to \mathbb{R}^+$, which has an expected value of

$$\mathbb{E}[C] := \sum_{i=1}^{\mu} \sum_{j=1}^{\ell} q_{\text{ADV},i} \cdot c_{i,j} \cdot q_{\text{RAND},j} = q_{\text{ADV}}^{\mathsf{T}} \cdot \mathcal{M} \cdot q_{\text{RAND}} \ .$$

Obviously, RAND wants to minimize this expected value by choosing its probability distribution accordingly, and conversely, the adversary wants to maximize it. But what do concrete choices for the two players look like? We can argue as we did previously for deterministic strategies.

- RAND knows $\mathcal{M}$ and further knows that the adversary wants to choose its mixed strategy q_{ADV} such that $\mathbb{E}[C]$ is maximized for the given choice of q_{RAND}. Thus, RAND can choose a strategy q^*_{RAND} such that
$$q^*_{\text{RAND}} = \arg\min_{q_{\text{RAND}}} \{\max_{q_{\text{ADV}}} \{q_{\text{ADV}}^{\mathsf{T}} \cdot \mathcal{M} \cdot q_{\text{RAND}}\}\} \ .$$

[2] If we consider a purely deterministic setting, the problem of Examples 2.3 and 2.5 does not appear anyway since the algorithm is fixed first, and the adversary chooses its strategy afterwards.

Therefore, the adversary has a payoff of at most

$$v_{\text{RAND}} := \max_{q_{\text{ADV}}} \{q_{\text{ADV}}^{\intercal} \cdot \mathcal{M} \cdot q_{\text{RAND}}^{*}\} .$$

- Conversely, the adversary can choose a mixed strategy q_{ADV}^{*} such that

$$q_{\text{ADV}}^{*} = \arg\max_{q_{\text{ADV}}} \{\min_{q_{\text{RAND}}} \{q_{\text{ADV}}^{\intercal} \cdot \mathcal{M} \cdot q_{\text{RAND}}\}\} .$$

This means that it can guarantee a payoff of at least

$$v_{\text{ADV}} := \min_{q_{\text{RAND}}} \{q_{\text{ADV}}^{*\intercal} \cdot \mathcal{M} \cdot q_{\text{RAND}}\} .$$

As with deterministic strategies, it is easy to see that we have $v_{\text{ADV}} \leq v_{\text{RAND}}$, but here, this holds with equality even if there is no equilibrium in pure strategies. The following theorem is one of the most important results from game theory and states that $v_{\text{ADV}} = v_{\text{RAND}}$. We will not prove it here.

Theorem 2.10 (Minimax Theorem). *For every two-person zero-sum game with finite strategies, we have*

$$\min_{q_{\text{RAND}}} \{\max_{q_{\text{ADV}}} \{q_{\text{ADV}}^{\intercal} \cdot \mathcal{M} \cdot q_{\text{RAND}}\}\} = \max_{q_{\text{ADV}}} \{\min_{q_{\text{RAND}}} \{q_{\text{ADV}}^{\intercal} \cdot \mathcal{M} \cdot q_{\text{RAND}}\}\} . \qquad \square$$

The minimax theorem states that, for any two-person zero-sum game, there is an equilibrium in mixed strategies q_{ADV}^{*} and q_{RAND}^{*} that are defined as above. Let us now go back to online algorithms and see what this implies.

We note that if one of the two mixed strategies q_{ADV} or q_{RAND} is fixed, the respective other player can maximize its payoff by choosing a deterministic strategy. Suppose that the strategy q_{RAND} is fixed; this corresponds to the situation we are dealing with in the context of randomized online algorithms, because the adversary chooses a deterministic strategy while knowing the probability distribution with which RAND chooses its deterministic strategies. Then

$$\mathcal{M} \cdot q_{\text{RAND}} = \begin{pmatrix} c_{1,1} & \dots & c_{1,\ell} \\ \vdots & \ddots & \vdots \\ c_{\mu,1} & \dots & c_{\mu,\ell} \end{pmatrix} \cdot \begin{pmatrix} q_{\text{RAND},1} \\ \vdots \\ q_{\text{RAND},\ell} \end{pmatrix} = \begin{pmatrix} c_1 \\ \vdots \\ c_\mu \end{pmatrix} ,$$

which means that the adversary can choose its strategy as a unit vector in which exactly that distinct entry is non-zero, for which the corresponding entry of the vector $(c_1, c_2, \dots, c_\mu)^{\intercal}$ is as large as possible, to maximize its payoff

$$\left(q_{\text{ADV},1}, \dots, q_{\text{ADV},\mu}\right) \cdot \begin{pmatrix} c'_1 \\ \vdots \\ c'_\mu \end{pmatrix} .$$

Analogously, we can argue the other way around. If we combine this fact with Theorem 2.10, we obtain the following lemma, where e_i with $1 \leq i \leq \mu$ (e_j with $1 \leq j \leq \ell$, respectively) denotes the corresponding unit vector for which the ith (jth, respectively) entry is 1 and all other entries are 0.

Lemma 2.3 (Loomis' Lemma). *For every two-person zero-sum game with finite strategies, we have*

$$\min_{q_{\text{RAND}}}\{\max_i\{e_i^\intercal \cdot \mathcal{M} \cdot q_{\text{RAND}}\}\} = \max_{q_{\text{ADV}}}\{\min_j\{q_{\text{ADV}}^\intercal \cdot \mathcal{M} \cdot e_j\}\} \; . \qquad \square$$

Since, for every fixed mixed strategy q'_{ADV} for the adversary,

$$\max_{q_{\text{ADV}}}\{\min_j\{q_{\text{ADV}}^\intercal \cdot \mathcal{M} \cdot e_j\}\} \geq q_{\text{ADV}}'^\intercal \cdot \min_j\{\mathcal{M} \cdot e_j\} \; ,$$

we can, using the minimax theorem (or rather Loomis' lemma), conclude that, for every q'_{ADV},

$$\min_{q_{\text{RAND}}}\{\max_i\{e_i^\intercal \cdot \mathcal{M} \cdot q_{\text{RAND}}\}\} \geq q_{\text{ADV}}'^\intercal \cdot \min_j\{\mathcal{M} \cdot e_j\} \; .$$

Now let q^*_{RAND} be a best mixed strategy for RAND; then we have

$$\max_i\{e_i^\intercal \cdot \mathcal{M} \cdot q^*_{\text{RAND}}\} \geq q_{\text{ADV}}'^\intercal \cdot \min_j\{\mathcal{M} \cdot e_j\} \; .$$

If we take a closer look, we observe that this inequality is exactly Yao's principle, which we introduced in Section 2.3. The statement is that, for a best randomized online algorithm (namely q^*_{RAND}), there is an input (namely $e_i^\intercal$) such that the expected performance is at least as large as that of a best deterministic online algorithm (namely e_j, which induces minimum cost) on an arbitrary but fixed distribution over inputs (namely $q_{\text{ADV}}'^\intercal$).

2.5 A Lower Bound for Randomized Online Algorithms for Paging

Theorem 2.2 states that the randomized online algorithm RMARK for paging obtains an expected competitive ratio of at most $2H_k$. The next question is whether this is all we can do when allowing randomized computations. The following result gives the answer; as a matter of fact, the bound on the competitive ratio of RMARK is only a multiplicative constant of 2 away from a bound on what a randomized online algorithm can achieve at best. To prove the claim, we make use of Yao's principle, which we introduced in Section 2.3.

Theorem 2.11. *No randomized online algorithm for paging is better than H_k-competitive in expectation.*

Proof. As in the proof of Theorem 1.5, it is sufficient to assume that there are only $m = k + 1$ pages in total. We construct instances of the following form.

- In the first time step, page p_{k+1} is requested; by definition, it is not in the cache of any algorithm.

- In every subsequent time step, an arbitrary page is requested other than the one that was requested right before; each of these k pages is requested with probability $1/k$.

We now show that every deterministic online algorithm has a large expected cost compared to an optimal solution on these instances. Applying Yao's principle, we can then derive a lower bound on the expected competitive ratio of a best randomized online algorithm for paging.

Let ALG be some deterministic online algorithm for paging; without loss of generality (see Exercises 1.5 and 1.6), we restrict ourselves to demand paging algorithms. Thus, ALG has exactly k pages in its cache in every time step. In other words, every page that is requested in some time step is not in the cache with a probability of $1/k$ (an exception is, of course, the first time step).

We subdivide the input into phases; the number of phases is denoted by N. This is done similarly to Definition 1.8; however, the phases are now *stochastic*. By $\mathcal{I}_N$ we denote the set of all instances that contain N complete phases. Phase P_1 starts with time step T_1 and ends after time step T_r, where r is chosen such that in time step T_{r+1} all of the $k+1$ different pages were requested. After that, phase P_2, which is defined analogously, begins, etc. P_N ends right after time step T_{n-1}, which means that there is one more request after the last phase ends. We observe that $\mathcal{I}_N$ is well defined and contains infinitely many instances (of infinitely many lengths). However, there are instances that are not contained in any set.

Recall that, in every time step (except for T_1), ALG causes a page fault with probability $1/k$. This is true for every instance in $\mathcal{I}_N$, because the probability of reaching the Nth phase starting with any sufficiently short prefix and continuing by requesting pages as described is 1. Now let $\mathbb{E}_{\text{ADV},N}[\text{cost}(\text{ALG}(P_j))]$ denote the expected cost of ALG in phase P_j, and let $|P_j|$ denote the expected length of P_j with $1 \leq j \leq N$. Then we obtain

$$\mathbb{E}_{\text{ADV},N}[\text{cost}(\text{ALG}(P_j))] \geq |P_j| \cdot \frac{1}{k} \,. \tag{2.13}$$

Conversely, there is an optimal algorithm OPT that does not make more than one page fault in every single phase plus one additional page fault in T_n; in phase P_1, OPT causes a page fault in time step T_1 and it replaces the page that is requested at the beginning of phase P_2, and so on (in P_N, it replaces the page that is requested in T_n). As a result, we have

$$\mathbb{E}_{\text{ADV},N}[\text{cost}(\text{OPT}(P_j))] = 1 \,, \tag{2.14}$$

for every phase P_j with $1 \leq j \leq N$.

The key point is to compute the expected length of a single phase; recall that all phases are complete. Phase P_j ends when the next page that will be requested is the first occurrence of the $(k+1)$th different page since the beginning of phase P_j. Thus, we have to compute how long it takes in expectation until all pages $p_1, p_2, \ldots, p_{k+1}$

have been requested (a combinatorial problem, which is closely related to the well-known *coupon collector's problem*); the expected number of time steps of phase P_j is this number minus 1. To this end, let X_i, for any i with $1 \leq i \leq k+1$, denote a random variable that counts the number of steps that pass until the ith page gets requested after $i-1$ different pages were already requested (within P_j).

Let us first consider the corresponding probability q_i that the ith new page is requested when in the preceding time steps of P_j already $i-1$ different pages were requested. Clearly, $q_1 = 1$ and, since in the second time step of P_j the first page is not requested again, also $q_2 = 1$; furthermore, $q_3 = (k-1)/k$, $q_4 = (k-2)/k$, etc. In general, we have

$$q_i = \frac{k-(i-2)}{k} \,, \tag{2.15}$$

for $2 \leq i \leq k+1$. This is true for every phase including P_1. Now we can compute the expected value of X_i as follows. Suppose the $(i-1)$th page was just requested, and now we count the time steps that are needed until the ith new page is requested. The probability that this takes t time steps is $q_i \cdot (1-q_i)^{t-1}$; in other words, for $t-1$ steps, a page is requested that was already requested before, and then, in the tth step, a page is requested that was not requested yet. We get

$$\begin{aligned}
\mathbb{E}_{\text{Adv},N}[X_i] &= \sum_{t=1}^{\infty} t \cdot q_i \cdot (1-q_i)^{t-1} \\
&= \sum_{t=0}^{\infty} (t+1) \cdot q_i \cdot (1-q_i)^{t} \\
&= \sum_{t=0}^{\infty} t \cdot q_i \cdot (1-q_i)^{t} + \sum_{t=0}^{\infty} q_i \cdot (1-q_i)^{t} \\
&\quad \textit{(since none of the summands are negative)} \\
&= (1-q_i) \cdot \sum_{t=1}^{\infty} t \cdot q_i \cdot (1-q_i)^{t-1} + q_i \cdot \sum_{t=0}^{\infty} (1-q_i)^{t} \\
&\quad \textit{(since the first part is 0 for } t = 0\textit{)} \\
&= (1-q_i) \cdot \mathbb{E}_{\text{Adv},N}[X_i] + \frac{q_i}{1-(1-q_i)} \,, \\
&\quad \textit{(using the closed form of the geometric series)}
\end{aligned}$$

and therefore

$$\mathbb{E}_{\text{Adv},N}[X_i] = \frac{1}{q_i} \,. \tag{2.16}$$

Now let $X = X_1 + X_2 + \ldots + X_{k+1}$ denote a random variable that counts all time steps until $k+1$ different pages were requested since the beginning of P_j. Then, we have

$$\begin{aligned}
\mathbb{E}_{\text{Adv},N}[X] &= \mathbb{E}_{\text{Adv},N}[X_1 + X_2 + \ldots + X_{k+1}] \\
&= \mathbb{E}_{\text{Adv},N}[X_1] + \mathbb{E}_{\text{Adv},N}[X_2] + \ldots + \mathbb{E}_{\text{Adv},N}[X_{k+1}] \\
&\quad \textit{(by linearity of expectation)} \\
&= 1 + \frac{1}{q_2} + \ldots + \frac{1}{q_{k+1}} \\
&\quad \textit{(as a consequence of (2.16))} \\
&= 1 + \sum_{i=2}^{k+1} \frac{1}{q_i} \\
&= 1 + \sum_{i=2}^{k+1} \frac{k}{k-(i-2)} \\
&\quad \textit{(due to (2.15))} \\
&= 1 + k \cdot \sum_{i=2}^{k+1} \frac{1}{k-(i-2)} \\
&= 1 + k \cdot \sum_{i=1}^{k} \frac{1}{k-(i-1)} \\
&= 1 + k \cdot \sum_{i=1}^{k} \frac{1}{i} \\
&= 1 + k \cdot H_k \ ,
\end{aligned}$$

and thus P_j ends after an expected number of kH_k steps. It is important to note that the above arguments are also true for the first and the last phase.

From (2.13), it follows that the expected cost of Alg in P_j is at least

$$\mathbb{E}_{\text{Adv},N}[\text{cost}(\text{Alg}(P_j))] \geq H_k \ . \tag{2.17}$$

As Alg is any deterministic online algorithm for paging, from (2.14) and (2.17) together with the fact that Opt causes a page fault in T_n, we obtain

$$\frac{\inf_j(\mathbb{E}_{\text{Adv},N}[\text{cost}(A_j(\mathcal{I}_N))])}{\mathbb{E}_{\text{Adv},N}[\text{cost}(\text{Opt}(\mathcal{I}_N))]} \geq \frac{\sum_{j=1}^{N} H_k}{1 + \sum_{j=1}^{N} 1} = \frac{NH_k}{N+1} \ ,$$

and since

$$\liminf_{N\to\infty}\left(\frac{NH_k}{N+1}\right) = H_k \ ,$$

it follows that (i) of Theorem 2.8 is satisfied. Moreover, note that the expected cost of OPT increases with N, that is,

$$\limsup_{N\to\infty}(\mathbb{E}_{\text{ADV},N}[\text{cost}(\text{OPT}(\mathcal{I}_N))]) = \infty \ ,$$

which means that also (ii) of Theorem 2.8 is true. Therefore, we can apply Yao's principle and conclude that no randomized online algorithm has an expected competitive ratio better than H_k. □

⋆2.6 A Barely Random Algorithm for Paging

We now have gained some insight into the paging problem. In particular, we know that there is a randomized online algorithm that is exponentially better than all deterministic ones. If we take a closer look, we also realize that the algorithms we have dealt with so far are clearly efficient (in terms of their time complexity). Now we ask how much randomization we need to be this good. The randomized marking algorithm RMARK that we discussed in Section 2.2 obviously uses a number of random bits that increases with the input length n. In this section, we show that this is not necessary; we only need a constant number (with respect to n) of random bits to obtain (asymptotically) the same competitive ratio. To this end, we design a so-called *barely random algorithm* RMARKBARELY that chooses randomly between some deterministic marking algorithms. Formally, we have

$$\text{strat}(\text{RMARKBARELY}) = \{Mark_1, Mark_2, \ldots, Mark_{2^b}\} \ ,$$

where the online algorithms $Mark_i$ with $1 \leq i \leq 2^b$ are marking algorithms that replace unmarked pages in the cache in such a way that if a page is requested, this page is in the cache for a large number of them. Let us first try to explain the idea with an example.

Example 2.6. Suppose we read two random bits and thus pick one out of four algorithms $Mark_1$, $Mark_2$, $Mark_3$, and $Mark_4$. Moreover, suppose the cache has a size of 7 and is initialized with the pages $p_1, p_2, \ldots, p_7$ such that we are facing the situation

p_1	p_2	p_3	p_4	p_5	p_6	p_7

at the beginning. All algorithms memorize these pages in this order. Now, the page p_8 is requested, which is not in the cache. Then $Mark_i$ replaces p_i with p_8, which leads to the following situation; we color marked pages gray.

$Mark_1$:

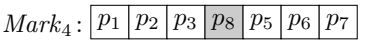
p_8	p_2	p_3	p_4	p_5	p_6	p_7

$Mark_2$:

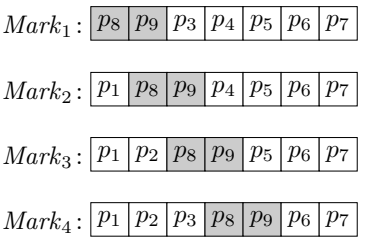
p_1	p_8	p_3	p_4	p_5	p_6	p_7

$Mark_3$:

p_1	p_2	p_8	p_4	p_5	p_6	p_7

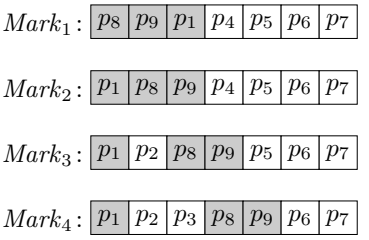


If the page p_9 is requested next, all algorithms remove the next page according to the above order of the pages that were in the cache initially, which leads to

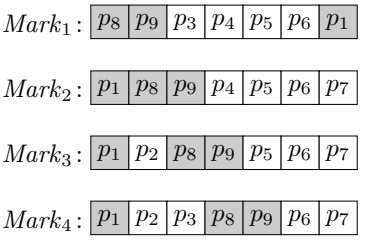


as a result.

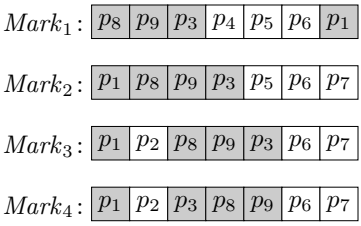
The idea is that if, for instance, the page p_2 is now requested, this page has only been removed from the cache by some algorithms; in this case $Mark_1$ and $Mark_2$. If one of these four algorithms is chosen uniformly at random, the page p_2 is still in the cache with a probability of 1/2. However, we run into problems if we continue with this strategy. It can happen that some algorithms act identically after some time step. Suppose the "old" page p_1 is requested. Then we get the following situation.



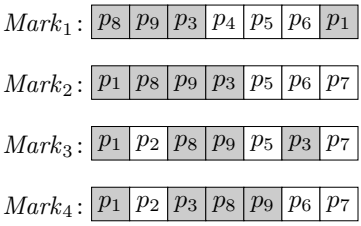
We note that the algorithms $Mark_1$ and $Mark_2$ now have the same marked and unmarked pages in their caches. From this point on, these two algorithms behave the same until the end of the current phase. Clearly, we want to avoid such situations whenever possible. A way out is to make sure that $Mark_1$ does not remove the page p_3, but some other page which no other algorithm removed so far; an example is p_7, which leads to the following result.



Suppose page p_3 is requested next. Again, if all algorithms follow our original idea, two algorithms have the same cache content afterwards; this time these are $Mark_3$ and $Mark_4$, and the cache contents look as follows.



This problem can also be fixed by making $Mark_3$ remove the page p_6 instead of p_5; as a consequence, we get the following situation.



So far, so good, but now we encounter a point where we cannot use this strategy again. The problem is that there is no page left to remove from the cache that was not yet removed by any algorithm. ♢

To analyze RMarkBarely, we again consider the k-phase partition of the input according to Definition 1.8. Moreover, we use the notion of "old" and "new" pages as in the proof of Theorem 2.2, where we showed that RMark is $2H_k$-competitive in expectation. Now suppose that, at the beginning, three new pages are requested. Every algorithm from strat(RMarkBarely) removes pages using the principle from Example 2.6. Afterwards, every algorithm is assigned a sequence of three removed unmarked old pages; for instance, $Mark_3$ used (p_3, p_4, p_5). This is done by ordering the pages that are in the cache at the beginning in some way (for instance, by their indices) and make every algorithm replace a unique subsequence as long as this is possible; see Figure 2.4. As the computation continues, we want to ensure that these subsequences stay unique for every algorithm, such that no two of them act identically. Since marked pages do not get evicted in the current phase, they need to be removed from these subsequences.

If a new page is requested in some time step, the subsequences are extended by one page, which means that this unmarked old page is removed to load the new page into the cache. However, if an unmarked old page p is requested, the situation is more complex. After this request, p is not unmarked anymore and therefore has to be removed from every subsequence including it. Algorithms that already removed

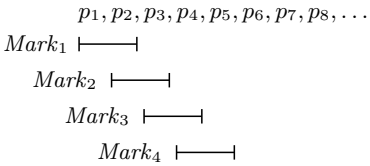


Figure 2.4. The pages that are in the cache at the beginning of a phase and the first two pages that are replaced by the deterministic strategies RMarkBarely chooses from.

p are now not allowed to remove the page p' that would be next according to the ordering. As we have seen in Example 2.6, we could not guarantee that all algorithms act differently afterwards. Instead, these algorithms use a page p'' which, so far, is in the cache of all algorithms, that is, it was not removed yet by any algorithm. We will soon investigate under what circumstances such a page exists, but for now just assume that it does. Then, we add p'' to all subsequences of all algorithms that do not have p in their caches, which means that in their caches p'' is replaced by p.

Example 2.7. We now take a closer look at the situation described in Example 2.6. The subsequence (p_1, p_2) was assigned to $Mark_1$ and (p_2, p_3) was assigned to $Mark_2$ after the first two requests p_8 and p_9. If p_1 is requested after that, the subsequence of $Mark_2$ is unchanged, since it has p_1 in its cache, and simply needs to mark it. The situation is different for $Mark_1$; here, p_1 is removed from the subsequence since the page is now marked. If now the page p_3 is removed to load p_1, both subsequences are (p_2, p_3). Thus, $Mark_1$ removes the unmarked page p_7 instead, which leads to a subsequence (p_2, p_7). Consequently, the two algorithms have different unmarked pages in their caches after this request. ♢

In general, after this strategy is repeated for some number of times, we will get to a point where we do not have any free (that is, unmarked old) pages left, and the adversary can force some of the algorithms to have the same marked and unmarked pages in their caches. RMarkBarely therefore works in rounds in which it partitions the algorithms into groups. In every round, two algorithms from different groups replace different pages. The idea is that the unavoidable problem of some algorithms acting the same from some point on is somewhat "controlled" by building these groups. Note that RMarkBarely must simulate all deterministic algorithms $Mark_1, Mark_2, \ldots, Mark_{2^b}$ at once in order to know how the randomly chosen one replaces pages.

Exercise 2.7. The instance in Example 2.6 is (p_8, p_9, p_1, p_3). It is easy to see that, after a fifth page is requested, the deterministic algorithms can still choose pages in their caches to replace such that all caches stay pairwise different.

For $b = 2$ and $k = 7$, give an instance of length 5 such that the last request forces two algorithms from strat(RMarkBarely) to have the same cache content afterwards.

Now that we have a rough idea about how the algorithm works, in what follows, we prove that RMarkBarely is indeed asymptotically as good as RMark. To this end, we formalize the ideas we just discussed, and describe the algorithm's behavior in more detail.

Theorem 2.12. RMarkBarely *uses b random bits, where $2^b < k$, and is strictly*

$$\left(3b + \frac{2(k+1)}{2^b}\right)\text{-competitive}$$

in expectation for paging.

Proof. Consider the deterministic algorithms $Mark_1, Mark_2, \ldots, Mark_{2^b}$ and the k-phase partition as given by Definition 1.8. The algorithms follow different strategies to remove unmarked pages. However, the set of marked pages is the same for every algorithm at any point in time, and thus all algorithms have the same pages in their caches at the beginning of any phase.

Let there be N phases in total, let P_j with $1 \leq j \leq N$ be an arbitrary phase, and let us denote pages by "new" and "old" as in the proof of Theorem 2.2; that is, all pages that are in the cache at the beginning of P_j are called old, and all pages that are not old and that are requested during P_j are called new. Moreover, we ignore all requests to pages that were already requested during P_j since they do not cause page faults for any of the considered marking algorithms. Recall that, during any phase, at least one new page is requested. For the sake of a simple notation, we assume that P_j consists of k requests $x_1, x_2, \ldots, x_k$ to unmarked old or new pages. These k requests are now processed in $b+1$ rounds that are defined as follows.

- Round R_0 starts with request x_1 and takes until request x_{k-2^b+1} is processed.
- For $z \geq 1$, round R_z consists of the $2^b/2^z$ requests $x_{k-2^{b-(z-1)}+2}$ to $x_{k-2^{b-z}+1}$.

In every round, the 2^b algorithms are divided into $2^b/2^z$ groups $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_{2^{b-z}}$ each of size 2^z. In round R_0, every such group consists of one unique algorithm, that is, $\mathcal{G}_i = \{Mark_i\}$. At the beginning of round R_z with $z \geq 1$, the two groups $\mathcal{G}_i$ and $\mathcal{G}_{i+2^{b-z}}$ are merged into a single group $\mathcal{G}_i$; for instance, if there are eight algorithms in total, $\mathcal{G}_1$ and $\mathcal{G}_5$ are merged to $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{G}_6$ are merged to $\mathcal{G}_2$, and so on, right before round R_1.

Now consider a request x_{t+1} processed in round R_z, and denote the number of new pages that are requested during $x_1, x_2, \ldots, x_t$ by l_t. Then there are $t - l_t$ old pages that were requested before and that are already marked in this time step. Conversely, there are

$$k - t + l_t \tag{2.18}$$

unmarked old pages, and, for every algorithm, some of them are in the cache and some are not. We denote these pages by $\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_{k-t+l_t}$, and set

$$S_t := (\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_{k-t+l_t}) \,.$$

In other words, S_t is a sequence of old pages that remain unmarked until the tth time step of P_j; right before the first time step of P_j, these are all pages that are in the cache at the beginning of this phase.

As already stated, every group $\mathcal{G}_i$ replaces pages following a certain strategy. More specifically, in the tth time step of P_j, every $\mathcal{G}_i$ is assigned a set of l_t unmarked old pages (determined by the ordering that is given by S_t). This set is

$$E_{i,t} := \{\overline{p}_i, \overline{p}_{i+1}, \ldots, \overline{p}_{i+l_t-1}\} ,$$

for $l_t \geq 1$ and $E_{i,t} = \emptyset$ for $l_t = 0$. Algorithms in $\mathcal{G}_i$ only replace pages in their caches that are in this set in the corresponding time step. Note that every unmarked old page is in at most l_t different sets $E_{i,t}$ since $|E_{i,t}| = l_t$ and by the construction of the sets. We first have to show that there are always enough unmarked old pages. By the definition of the sets $E_{i,t}$ with $1 \leq i \leq 2^b/2^z$, the unmarked old pages assigned to groups are

$$\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_{2^b/2^z+l_t-1} ,$$

which amounts to $2^b/2^z + l_t - 1$ pages. As noted in (2.18), there are $k - t + l_t$ unmarked old pages in total. Thus, there are

$$(k - t + l_t) - \left(\frac{2^b}{2^z} + l_t - 1\right) = k - t - \frac{2^b}{2^z} + 1 \tag{2.19}$$

unmarked old pages left. Next, we bound t from above. To this end, recall that round R_z with $z \geq 0$ of P_j ends after the $(k - 2^{b-z} + 1)$th time step of P_j. Consequently, we have

$$t + 1 \leq k - \frac{2^b}{2^z} + 1 . \tag{2.20}$$

From (2.19) and (2.20), it follows that the number of pages that are left is at least

$$k - \left(k - \frac{2^b}{2^z}\right) - \frac{2^b}{2^z} + 1 = 1 , \tag{2.21}$$

and thus the sets $E_{i,t}$ are well defined.

Now we describe how the sequences S_t are modified. As we assume that only unmarked pages are requested, we can simply distinguish the following two cases.

Case 1. Suppose x_{t+1} is a new page. In this case, we set $S_{t+1} = S_t$ (note that both t and l_t increase by one). Moreover, we add another element to all sets, that is, $E_{i,t+1} = E_{i,t} \cup \{\overline{p}_{i+l_t}\}$. This simply means that the page $\overline{p}_{i+l_t}$ gets replaced by x_{t+1} in the caches of the corresponding algorithms, which is always possible due to (2.21).

Case 2. Now suppose x_{t+1} is an unmarked old page. Then x_{t+1} is removed from S_t and replaced by some other page that is so far not assigned to any group.

Due to (2.21), there is at least one page that can take the place of x_{t+1}. Furthermore, since every unmarked old page is in at most l_t sets $E_{i,t}$, at most l_t sets $E_{i,t+1}$ are different from $E_{i,t}$.

We have just shown how the sets $E_{i,t}$, which are used to keep track of which pages are replaced when a page fault occurs, are created and maintained. Now let Mark_q with $1 \leq q \leq 2^b$ be some algorithm from a group $\mathcal{G}_i$, and let x_{t+1} be a request that causes a page fault. Then Mark_q loads this page into the cache by replacing a page from $E_{i,t+1}$. We show that such a page always exists. Mark_q has $k-t$ unmarked old pages in its cache so far. By definition, there are l_{t+1} unmarked old pages in the set $E_{i,t+1}$. In total (see (2.18)), after x_{t+1} is processed, there are $k + l_{t+1} - (t+1)$ unmarked old pages. Consequently, there are only $k-t-1$ unmarked old pages that are not in $E_{i,t+1}$, and hence the cache of Mark_q must contain at least one page from $E_{i,t+1}$. The algorithms are therefore consistent for paging, that is, they always produce a feasible output.

It remains to bound the expected number of page faults that are caused by RMarkBarely. This is done by bounding the average number of page faults made by the algorithms Mark_q. We begin with the total number of page faults of all algorithms combined. To this end, let l'_j denote the number of new pages that are requested during phase P_j.

- Every new page that is requested obviously causes one page fault for every algorithm, which sums up to

  $$l'_j 2^b \tag{2.22}$$

  page faults in total.

- It gets a little more tricky for unmarked old pages that are requested. As we have seen above, such a request x_{t+1} forces the algorithms of at most $l_t \leq l'_j$ groups to cause a page fault. Round R_0 consists of $k - 2^b + 1$ time steps, and since every group includes one single algorithm, this leads to at most $l'_j(k - 2^b + 1)$ page faults. In round R_z with $z \geq 1$, every group consists of 2^z algorithms and R_z consists of at most $2^b/2^z$ time steps; for each such round, at most l'_j groups are affected by this page fault. Since there are b such rounds in total, we get a maximum number of page faults of

  $$l'_j \frac{2^b}{2^z} 2^z \, b \, .$$

  All in all, the number of page faults in phase P_j due to unmarked old pages sums up to at most

  $$l'_j(k - 2^b + 1) + l'_j \, 2^b \, b \, . \tag{2.23}$$

- Last, we bound the number of page faults that are due to the merging of groups. When the algorithms from group $\mathcal{G}_{i+2^{b-z}}$ with $1 \leq i \leq 2^b/2^z$ are added to the ones from $\mathcal{G}_i$, we assume that, afterwards, they only replace pages according to the strategy of $\mathcal{G}_i$. However, it may be the case that the former algorithms replaced unmarked old pages that are still in the caches of the algorithms from group $\mathcal{G}_i$. For the sake of an easier analysis, we thus assume that the algorithms from $\mathcal{G}_{i+2^{b-z}}$ simply load the corresponding unmarked old pages into their caches without marking them; after that, all algorithms from $\mathcal{G}_i$ are assigned the same group $E_{i,t}$. For any affected algorithm, this causes an additional cost of at most $l_t \leq l'_j$. Every merging, independent of both the preceding and succeeding round, affects exactly half of the algorithms, that is, 2^{b-1} many; this happens exactly b times. Therefore, in the sum, this causes at most another

  $$l'_j 2^{b-1} b \tag{2.24}$$

  page faults.

As a direct consequence of (2.22) to (2.24), all algorithms in strat(RMarkBarely) together have at most a cost of

$$l'_j 2^b + l'_j(k - 2^b + 1) + l'_j 2^b b + l'_j 2^{b-1} b = l'_j \left(k + 1 + \frac{3}{2}\, 2^b\, b \right)$$

in each phase. This leads to an average cost of

$$l'_j \left(\frac{k+1}{2^b} + \frac{3}{2}\, b \right)$$

of a single algorithm per phase. Finally, as in the proof of Theorem 2.2, we can sum over all N phases and argue that Opt has to make at least

$$\sum_{j=1}^{N} \frac{1}{2}\, l'_j$$

page faults in total; furthermore, we can assume that all phases are complete. With this, we can bound the strict expected competitive ratio of RMarkBarely from above by

$$3b + \frac{2(k+1)}{2^b}$$

as we claimed. □

Now we can use Theorem 2.12 with $b = \lfloor \log_2 k \rfloor - 1$. It follows that RMarkBarely obtains a strict expected competitive ratio of at most

$$3\lfloor \log_2 k \rfloor - 3 + \frac{2(k+1)}{2^{\lfloor \log_2 k \rfloor - 1}} = 3 \log_2 k + \mathcal{O}(1) \in \mathcal{O}(\log k)\ .$$

Let us briefly interpret these results. No deterministic online algorithm can be better than k-competitive. However, this can be improved exponentially by picking a strategy uniformly at random from a set of constantly many ones. Such phenomena motivate the study of *advice complexity*, which we will introduce in the next chapter and which will subsequently accompany us throughout this book.

⋆2.7 Bounds with Probability Tending to One

So far, we always measured the quality of randomized online algorithms by means of the expected competitive ratio. In Section 2.1, we mentioned that, for offline problems, one is usually more interested in the concrete probabilities with which "good" or "bad" output is created. For a broad class of offline algorithms, statements about the expected value allow for statements about the concrete probability of success by using the amplification technique. For online algorithms, this argumentation does not work in general since they cannot repeat their computation.

However, for some online algorithms for paging, we can make statements that speak about the concrete probabilities of creating a good output. As for offline optimization problems, "good" means that the gain or cost of the computed solution is very close to the expected gain or cost. A desirable result would be that the probability that this happens is large. But what does "large" mean in this context? Preferably, it means "not constant," and we would be happy if the probability tends to 1, for instance, with increasing input length. In the following, we show that such a statement is not possible in general by using the fact that any input of length n can be extended to an input of length dn by repeating every request d times. The following theorem only speaks about strict competitiveness; a general statement is given in Exercise 2.8.

Theorem 2.13. *Let $f\colon \mathbb{N} \to \mathbb{R}^+$ be some unbounded increasing function. There is no randomized online algorithm for paging that is strictly c-competitive for inputs of length n with a probability of $1 - 1/f(n)$, where $c < k$.*

Proof. Let $c < k$, and suppose there is some $n_0 \in \mathbb{N}^+$ such that there is a randomized online algorithm Rand that, for any instance I of paging with $|I| = n \geq n_0$ is strictly c-competitive with a probability of $1 - 1/f(n)$, for some function f that tends to infinity with growing n. Without loss of generality, we assume that Rand is a demand paging algorithm (see Exercises 1.5 and 1.6).

Now let $n' \geq n_0$ be an arbitrary natural number that is a large multiple of k such that

$$\frac{1}{k^{k-1}} > \frac{1}{f(n')} \,. \tag{2.25}$$

We design a randomized online algorithm Rand$'$ that has a strict performance of c with a probability of $1 - 1/f(n')$ on inputs of length k. To this end, Rand$'$

simulates RAND on an input that is obtained by repeating every request n'/k times. By definition, RAND has a strict performance of c on this input with a probability of $1 - 1/f(n')$. Since RAND′ replaces the same pages as RAND, RAND′ also has a strict performance of c on the original instance of length k with a probability of $1 - 1/f(n')$.

Next, we show that the existence of RAND′ and thus RAND leads to a contradiction. Again, let there be $k + 1$ pages in total. Now consider the following instance I' of paging of length k. First, the page p_{k+1} is requested, and then some sequence of $k - 1$ pages such that the same page is never requested in two consecutive time steps. We already know that there is an optimal solution for I' that only makes a page fault in the first time step.

Now consider the solution of RAND′ for I'. In every time step in which the algorithm causes a page fault, RAND′ randomly chooses a page to replace in its cache. There is always a page that is chosen with probability at least $1/k$, and the advesary requests this page in the next time step. Thus, there is a sequence $p_{i_1}, p_{i_2}, \ldots, p_{i_{k-1}}$ of "bad" choices that causes RAND′ to have cost k. In the first time step, RAND′ chooses the bad page (that is, the page that is requested right after that) with probability at least $1/k$; with probability at least $1/k^2$, it chooses the bad pages in the first and the second time step, and so on. Clearly, the probability that it chooses the bad sequence is at least $1/k^{k-1}$. Thus, RAND′ has a strict performance of k on I' with at least this probability; but together with (2.25), this immediately contradicts our assumption that RAND′ has a strict performance of c on I' with a probability of $1 - 1/f(n')$. As a consequence, RAND′ cannot be strictly c-competitive with a probability of $1 - 1/f(n)$ for inputs of length n; thus, RAND cannot exist. □

Exercise 2.8.⋆ For Theorem 2.13, we assume that RAND is strictly c-competitive with the given probability, and from this conclude that RAND′ cannot be strictly c-competitive with this probability on short instances. Prove this result for the case that the additive constant α from Definition 2.2 is allowed to be positive.

Hint. Consider an input length that depends on α.

A consequence of Theorem 2.13 (and Exercise 2.8) is that we cannot hope for a randomized online algorithm that is better than (strictly) k-competitive with a probability that tends to 1 with growing n. As we know that there are deterministic online algorithms that reach this bound, this seems disappointing. Then again, the instances we used in the proof of this theorem appeared to be rather artificial, as they basically consist of a linear number of requests for the same page, and the optimal cost is always one. For paging, instances are of interest for which also the cost of an optimal solution grows with the input length.

In what follows, we therefore measure the probability of being as good as the expected value in the number N of phases according to the k-phase partition from Definition 1.8. Our goal is to again study the randomized marking algorithm RMARK, which was introduced in Section 2.2, and to show that it achieves a competitive

ratio that is close to its expected value $2H_k$ with a probability that tends to 1 as N increases. To prove this claim, we need the following technical lemma, which we state without a proof. Let $\exp\colon \mathbb{R} \to \mathbb{R}^+$ be the natural exponential function, that is, $\exp(x) = \mathrm{e}^x$ where $\mathrm{e} = 2.718\ldots$ is Euler's number.

Lemma 2.4 (Hoeffding's Inequality). *Let $X_1, X_2, \ldots, X_N$ be independent random variables such that $a_i \leq X_i - \mathbb{E}[X_i] \leq b_i$, for all i with $1 \leq i \leq N$. Then*

$$\Pr\left[\sum_{i=1}^{N}(X_i - \mathbb{E}[X_i]) \geq t\right] \leq \exp\left(-\frac{2t^2}{\sum_{i=1}^{N}(b_i - a_i)^2}\right),$$

for any $t > 0$. □

To prove the following theorem, we treat the cost of single phases as random variables. Note that the number of phases does not simply increase with the input length.

Theorem 2.14. *Let $\varepsilon > 0$. The probability that the competitive ratio of* RMark *is at least $2H_k + \varepsilon$ tends to 0 as the number of phases tends to infinity.*

Proof. Let I be any instance of paging, and let RMark be the randomized marking algorithm defined in Section 2.2; we consider the k-phase partition according to Definition 1.8. As in the proofs of Theorems 2.2 and 2.11, for any phase P_i, we denote the cost of RMark on I during P_i by $\mathrm{cost}(\mathrm{RMark}(P_i))$ and set

$$C_i := \mathrm{cost}(\mathrm{RMark}(P_i)),$$

for every i with $1 \leq i \leq N$, to get an easier notation. Consequently, we have

$$\mathrm{cost}(\mathrm{RMark}(I)) = \sum_{i=1}^{N} C_i \,. \tag{2.26}$$

Note that the C_i are independent random variables since the cache content at the beginning and the end of any phase does not depend on the random decisions made in any of the previous phases; indeed, it only depends on I, which is chosen deterministically.

Since at most k different pages are requested in one phase, we can trivially bound every random variable C_i by

$$0 \leq C_i \leq k\,,$$

and thus

$$\underbrace{-\mathbb{E}[C_i]}_{a_i} \leq C_i - \mathbb{E}[C_i] \leq \underbrace{k - \mathbb{E}[C_i]}_{b_i} \,. \tag{2.27}$$

Theorem 2.2 states that RMark is strictly $2H_k$-competitive in expectation, which means that

$$\mathbb{E}[\text{cost}(\text{RMark}(I))] \leq 2H_k \cdot \text{cost}(\text{Opt}(I)) \ . \tag{2.28}$$

Now we are interested in the probability that the cost of RMark is at least $(2H_k + \varepsilon) \cdot \text{cost}(\text{Opt}(I))$, for any $\varepsilon > 0$. Using (2.28), (2.26), and linearity of expectation in this order, we obtain

$$\begin{aligned}
&\Pr[\text{cost}(\text{RMark}(I)) \geq (2H_k + \varepsilon) \cdot \text{cost}(\text{Opt}(I))] \\
&\quad \leq \Pr\left[\text{cost}(\text{RMark}(I)) \geq \frac{(2H_k + \varepsilon) \cdot \mathbb{E}[\text{cost}(\text{RMark}(I))]}{2H_k}\right] \\
&\quad = \Pr\left[\text{cost}(\text{RMark}(I)) - \mathbb{E}[\text{cost}(\text{RMark}(I))] \geq \frac{\varepsilon \cdot \mathbb{E}[\text{cost}(\text{RMark}(I))]}{2H_k}\right] \\
&\quad = \Pr\left[\sum_{i=1}^{N} C_i - \mathbb{E}\left[\sum_{i=1}^{N} C_i\right] \geq \frac{\varepsilon \cdot \mathbb{E}[\text{cost}(\text{RMark}(I))]}{2H_k}\right] \\
&\quad = \Pr\left[\sum_{i=1}^{N} (C_i - \mathbb{E}[C_i]) \geq \frac{\varepsilon \cdot \mathbb{E}[\text{cost}(\text{RMark}(I))]}{2H_k}\right] .
\end{aligned}$$

We now apply Hoeffding's inequality and obtain

$$\begin{aligned}
&\Pr\left[\sum_{i=1}^{N} (C_i - \mathbb{E}[C_i]) \geq \frac{\varepsilon \cdot \mathbb{E}[\text{cost}(\text{RMark}(I))]}{2H_k}\right] \\
&\quad \leq \exp\left(-\frac{(\varepsilon \cdot \mathbb{E}[\text{cost}(\text{RMark}(I))])^2}{2H_k^2 \sum_{i=1}^{N} (b_i - a_i)^2}\right) .
\end{aligned}$$

Recall that every phase contains at least one request for a new page. Thus, RMark makes at least one page fault per phase, and we have

$$\mathbb{E}[\text{cost}(\text{RMark}(I))] \geq N \ .$$

Moreover, due to (2.27), we have

$$\sum_{i=1}^{N} (b_i - a_i)^2 = Nk^2 \ .$$

With this, we finally get

$$\Pr\left[\sum_{i=1}^{N} (C_i - \mathbb{E}[C_i]) \geq \frac{\varepsilon \cdot \mathbb{E}[\text{cost}(\text{RMark}(I))]}{2H_k}\right] \leq \exp\left(-\frac{\varepsilon^2}{2H_k^2 k^2} N\right) .$$

Since both ε and k are constant, this upper bound is monotonically decreasing in the number of phases N. To conclude, the probability that the actual cost of RMark on I is at most $(2H_k + \varepsilon) \cdot \text{cost}(\text{Opt}(I))$ is at least

$$1 - \frac{1}{e^{\Omega(N)}} ,$$

which tends to 1 as we claimed. □

Unfortunately, this approach does not work in general. The proof of Theorem 2.14 only works since we have independent phases induced by RMark.

Exercise 2.9. In the proof of Theorem 2.14, we also made use of the fact that RMark is strictly $2H_k$-competitive in expectation. Does the proof still work if we also assume that the additive constant α from Definition 2.2 is positive?

2.8 The Ski Rental Problem

In this section, we study a very simple online problem that is met in disguise in various situations in everyday life, namely the *ski rental problem*. The idea is simple and captures the dilemma we are facing when we need to decide whether it is cheaper to rent some resource on demand again and again for small cost or to buy it at once, without knowing how often we need to use it.

Suppose you want to go skiing for as long as possible, but you do not own any skis. You have two choices. Either you rent skis for a small amount, say, EUR 1, or you buy the skis for EUR k, where k is some natural number larger than 1. To simplify things, assume that the only thing that would prevent you from skiing is the weather, so you are in excellent shape and highly motivated. The problem is, however, that you only get a reliable weather forecast on the morning of the current day. Thus, at the beginning of every day with good weather, you have to decide whether you rent skis or buy them. In the latter case, you probably assume that buying pays off later as you expect there will be many more days with good weather. Let us model this situation as an online problem to see what can be achieved in terms of competitive analysis.

Definition 2.3 (Ski Rental Problem). The *ski rental problem* is an online minimization problem. An input consists of a sequence of n requests, where each request is either "good" or "bad," and the ith request represents whether it is possible to ski on the ith day. If the request is "bad," the output is always "⟨null⟩." If the request is "good," an online algorithm must either output "rent" or "buy." If an online algorithm answers "buy" in some time step, it can only answer "⟨null⟩" in all subsequent time steps. Let $k \in \mathbb{N}^+$ with $k \geq 2$; the cost of an output is the number of "rent" answers, plus k if there was a "buy" answer.

Definition 2.3 implies that, for any online algorithm for the ski rental problem, we simply need to specify what it does on days with good weather. Let us start with deterministic strategies. First of all, we note that it is a bad strategy to never buy the skis at all. If some algorithm never buys the skis no matter how many days with good weather there are, an adversary can easily force it to have an unbounded cost, namely n. But when should the skis be bought? To buy on the first day with good weather also seems to be a bad idea, because then the adversary will make all subsequent days have bad weather, and the online algorithm pays k times as much as is necessary.

Let us try another strategy, namely "break even." This strategy is to rent the skis for the first $k-1$ days with good weather and to buy them on the kth day with good weather. It turns out that this is the best deterministic strategy we can follow in such a situation; we denote the corresponding online algorithm by BreakEven.

Theorem 2.15. BreakEven *is strictly* $(2-1/k)$*-competitive for the ski rental problem and no deterministic online algorithm is better than strictly* $(2-1/k)$*-competitive.*

Proof. Let us first prove the lower bound. We consider an adversary that always starts with a sequence of "good" requests, and possibly switches to "bad" at some point; after such a switch, it never switches back to "good." As we have just discussed, never buying the skis is not a promising idea. So we simply distinguish between three cases depending on when the skis are bought by some online algorithm Alg; say, this happens on the ith day with good weather if such a day exists.

Case 1. Suppose $1 \leq i < k$. The adversary constructs the input such that, after the ith day, all days have bad weather and therefore an optimal strategy rents the skis for all i days with good weather. We can immediately bound the competitive ratio from below by

$$\frac{k+i-1}{i} > \frac{2k-1}{k}$$

in this case.

Case 2. Now suppose $i > k$. Again, the adversary causes all days after the ith to have bad weather. An optimal strategy in such a case is to buy the skis at the first day with good weather. We get a bound of

$$\frac{k+i-1}{k} > \frac{2k-1}{k}$$

on the competitive ratio also in the second case.

So far, we have basically covered all algorithms that have a strategy which deviates from that of BreakEven. Now we study what happens in this case. Of course, we now consider an arbitrary adversary.

Case 3. So finally, suppose $i = k$. If there are at least k days of good weather, Alg (more specifically, BreakEven) pays exactly $2k-1$. In this case, an optimal

algorithm pays at least k, no matter whether the skis are rented or bought. Therefore, the (strict) performance is

$$\frac{2k-1}{k} = 2 - \frac{1}{k} .$$

On the other hand, if there are fewer than k days with good weather, ALG is even optimal.

Summing up, no online algorithm can be better than strictly $(2-1/k)$-competitive, and only BREAKEVEN achieves this competitive ratio. □

According to Definition 1.6, BREAKEVEN is *strongly* strictly competitive. We chose to formulate the claim in terms of *strict* competitiveness, because the cost k of buying the skis is a fixed parameter of the problem. Indeed, BREAKEVEN's cost is never larger than $2k-1$ and therefore, for any I, if we plug

$$\text{cost}(\text{BREAKEVEN}(I)) \leq 2k-1, \quad \text{cost}(\text{OPT}(I)) \geq 1, \quad \text{and } c = 1$$

into

$$\text{cost}(\text{BREAKEVEN}(I)) \leq c \cdot \text{cost}(\text{OPT}(I)) + \alpha ,$$

we get

$$2k - 1 \leq 1 + \alpha .$$

Hence, for $\alpha = 2(k-1)$, BREAKEVEN is 1-competitive. Although there do exist online algorithms that are not competitive for the ski rental problem at all, it seems to be the case that all somewhat "serious" online algorithms are 1-competitive, but only for a value for α that depends on k. As a consequence, the interesting case is to consider strict competitiveness. As we have seen, we cannot be better than strictly $(2-1/k)$-competitive in a deterministic setting; with increasing k, this lower bound tends to 2.

Exercise 2.10. Suppose the cost of buying skis is not a parameter known to any online algorithm in advance, but is given with the first request; the subsequent requests are either "good" or "bad" as defined in Definition 2.3. What follows for the non-strict competitive ratio of BREAKEVEN?

Now we ask how much randomization helps for ski rental. In Section 2.2, we have seen that randomization allows for a rather drastic, namely exponential (with respect to the cache size), improvement for paging. As BREAKEVEN already has a competitive ratio that is better than 2, the improvement for ski rental has to be less significant. But does randomization help at all? From Theorem 2.1, we already know that there is no such thing as an optimal randomized online algorithm for this problem.

In the following, we describe the randomized online algorithm RSki that chooses deterministic strategies according to some probability distribution $\Pr_{\text{RSki}}$. These strategies are defined as follows; for every i with $1 \leq i \leq k$, let Buy_i be the deterministic online algorithm that rents skis until day $i-1$ of good weather and buys them on the ith such day (if it exists). We set

$$\text{strat}(\text{RSki}) := \{Buy_1, Buy_2, \ldots, Buy_k\}$$

and define a probability distribution $\Pr_{\text{RSki}} \colon \text{RSki} \to [0,1]$ in what follows. To this end, let

$$\delta := \frac{k}{k-1}$$

and

$$\gamma := \frac{\delta - 1}{\delta^k - 1} \, .$$

The probability that RSki chooses the strategy Buy_i is given by

$$\Pr_{\text{RSki}}[Buy_i] := \gamma \delta^{i-1} \, .$$

We immediately verify that

$$\sum_{i=1}^{k} \Pr_{\text{RSki}}[Buy_i] = \sum_{i=1}^{k} \gamma \delta^{i-1} = \gamma \sum_{i=0}^{k-1} \delta^i = \left(\frac{\delta - 1}{\delta^k - 1} \right) \left(\frac{\delta^k - 1}{\delta - 1} \right) = 1 \, ,$$

thus RSki is well defined. Now we compute an upper bound on its expected competitive ratio.

Theorem 2.16. RSki *is strictly*

$$\left(\frac{\delta^k}{\delta^k - 1} \right)\text{-}competitive$$

in expectation for the ski rental problem.

Proof. At first, we can apply two simplifications that allow us to ignore some instances that might be created by the adversary.

1. Since every deterministic online algorithm from strat(RSki) buys the skis at the latest on the kth day with good weather, we only need to consider inputs that contain at most k such days.
2. Moreover, we may neglect days with bad weather that lie between days with good weather, because they do not change the achieved competitive ratio. This is due to the fact that RSki's behavior does not depend on them (and neither does an optimal solution).

In other words, we can assume that all instances consist of j days with good weather and after that, there are only days with bad weather for $1 \leq j \leq k$. For such an instance, an optimal solution always has cost j. If there are j days with good weather, the online algorithm Buy_i has cost $i-1+k$ if $i \leq j$; conversely, it has cost j whenever $i > j$.

Therefore, the expected cost of RSki on such an instance I, which has exactly j days with good weather, is

$$\begin{aligned}
&\mathbb{E}[\text{cost}(\text{RSki}(I))] \\
&= \sum_{i=1}^{j}(i-1+k)\cdot \Pr_{\text{RSki}}[Buy_i] + \sum_{i=j+1}^{k} j \cdot \Pr_{\text{RSki}}[Buy_i] \\
&= \sum_{i=1}^{j}(i-1)\cdot \Pr_{\text{RSki}}[Buy_i] + \sum_{i=1}^{j} k \cdot \Pr_{\text{RSki}}[Buy_i] + \sum_{i=j+1}^{k} j \cdot \Pr_{\text{RSki}}[Buy_i] \,.
\end{aligned}$$

By the definition of $\Pr_{\text{RSki}}$, we get

$$\begin{aligned}
\mathbb{E}[\text{cost}(\text{RSki}(I))] &= \gamma \sum_{i=1}^{j}(i-1)\delta^{i-1} + \gamma k \sum_{i=1}^{j}\delta^{i-1} + \gamma j \sum_{i=j+1}^{k}\delta^{i-1} \\
&= \gamma \sum_{i=0}^{j-1} i\delta^{i} + \gamma k \sum_{i=0}^{j-1}\delta^{i} + \gamma j \sum_{i=j}^{k-1}\delta^{i} \\
&= \gamma \sum_{i=0}^{j-1} i\delta^{i} + \gamma k \sum_{i=0}^{j-1}\delta^{i} + \gamma j \sum_{i=0}^{k-1}\delta^{i} - \gamma j \sum_{i=0}^{j-1}\delta^{i} \,.
\end{aligned}$$

We can now use the closed forms of these geometric series together with

$$\delta = \frac{k}{k-1} \iff k = \frac{\delta}{\delta-1} \,,$$

and get

$$\begin{aligned}
&\mathbb{E}[\text{cost}(\text{RSki}(I))] \\
&= \gamma \frac{(j-1)\delta^{j+1} - j\delta^{j} + \delta}{(\delta-1)^2} + \gamma k \frac{\delta^{j}-1}{\delta-1} + \gamma j \frac{\delta^{k}-1}{\delta-1} - \gamma j \frac{\delta^{j}-1}{\delta-1} \\
&= \frac{\gamma}{\delta-1}\left(\frac{(j-1)\delta^{j+1} - j\delta^{j} + \delta}{\delta-1} + k(\delta^{j}-1) + j(\delta^{k}-\delta^{j})\right) \\
&= \frac{\gamma}{\delta-1}\big(k(j-1)\delta^{j} - kj\delta^{j-1} + k + k(\delta^{j}-1) + j(\delta^{k}-\delta^{j})\big) \\
&= \frac{\gamma}{\delta-1}\big(jk\delta^{j} - kj\delta^{j-1} + j\delta^{k} - j\delta^{j}\big) \\
&= \frac{\gamma}{\delta-1}\left(\delta^{j}\underbrace{\left(k - \frac{k}{\delta} - 1\right)}_{0} + \delta^{k}\right) j
\end{aligned}$$

$$= \frac{\delta^k \gamma}{\delta - 1} j \ .$$

We plug in $\text{cost}(\textsc{Opt}(I)) = j$ and $\gamma = (\delta - 1)/(\delta^k - 1)$, finally obtaining

$$\begin{aligned} \mathbb{E}[\text{cost}(\text{RSki}(I))] &= \frac{\delta^k(\delta - 1)}{(\delta - 1)(\delta^k - 1)} \cdot \text{cost}(\textsc{Opt}(I)) \\ &= \frac{\delta^k}{\delta^k - 1} \cdot \text{cost}(\textsc{Opt}(I)) \end{aligned}$$

as claimed. □

We are now interested in the asymptotic behavior of the expected competitive ratio of RSki if k tends to infinity. Thus, we consider

$$\lim_{k\to\infty} \delta^k = \lim_{k\to\infty} \left(\frac{k}{k-1}\right)^k = \mathrm{e} \ ,$$

where as before $\mathrm{e} = 2.718\ldots$ denotes Euler's number, and obtain

$$\lim_{k\to\infty} \frac{\delta^k}{\delta^k - 1} = \frac{\mathrm{e}}{\mathrm{e} - 1} = 1.582\ldots \ .$$

So, under which conditions is RSki better in expectation than BreakEven? To answer this question, we note that

$$\begin{aligned} & \frac{\delta^k}{\delta^k - 1} < 2 - \frac{1}{k} && (2.29) \\ \iff & \delta^k < \left(2 - \frac{1}{k}\right)\delta^k - 2 + \frac{1}{k} \\ \iff & -\left(1 + \frac{1}{k}\right)\delta^k < -\left(2 - \frac{1}{k}\right) \\ \iff & \left(\frac{k}{k-1}\right)^k > \frac{2k-1}{k+1} \ , \\ & \qquad \textit{(by the definition of } \delta\textit{)} \end{aligned}$$

which holds whenever

$$\left(\frac{k}{k-1}\right)^k \geq 2 \ . \qquad (2.30)$$

Since $(2/(2-1))^2 = 4$ and $(k/(k-1))^k$ is monotonically decreasing and tends to e for $k \geq 2$, (2.30) and thus (2.29) is true for any $k \geq 2$. We conclude that, already for $k \geq 2$ (which is always satisfied since k is a natural number larger than 1), the randomized online algorithm RSki is better than any deterministic online algorithm; see Figure 2.5.

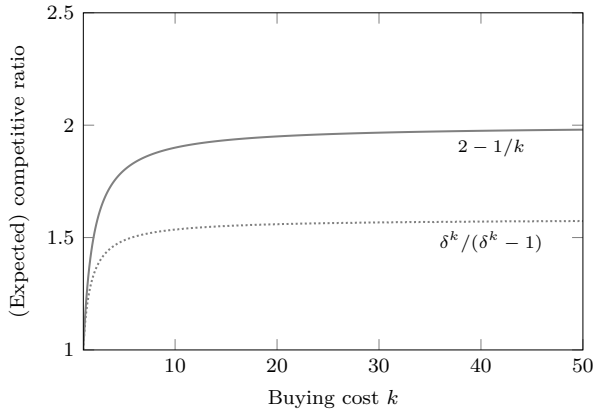


Figure 2.5. Comparison of the competitive ratios of BreakEven and RSki.

2.9 Historical and Bibliographical Notes

Introductions to randomized algorithms are given by, for instance, Hromkovič [81], Mitzenmacher and Upfal [118], and Motwani and Raghavan [120]. Also the textbook by Borodin and El-Yaniv [34] spends many chapters on randomized online algorithms. Knuth and Yao [96] discuss how to generate random numbers using random bits and remark "... we shall use the word 'algorithm' for such possibly infinite procedures, although strictly speaking we should be calling them 'computational methods' since algorithms are traditionally supposed to be finite in their worst case."

The Solovay-Strassen algorithm was published in 1977 by Solovay and Strassen [133]. A comprehensive analysis is, for instance, given in the aforementioned book by Hromkovič [81]. Note that this algorithm is a *one-sided-error Monte Carlo algorithm* for the inverse problem of deciding whether a given number is composite.

Yao's principle was first applied by Yao [145] in 1977. Since then, it was used for a large number of different online problems. The formulation and proof presented in this chapter follow the textbook by Borodin and El-Yaniv [34].

An introduction to game theory, including zero-sum games, is given by Straffin [136]. The minimax theorem is due to von Neumann [140]. Later, Nash [121] showed that all games with a finite number of strategies have equilibria in mixed strategies (they are therefore called *Nash equilibria*). Loomis' lemma was proven in 1946 by Loomis [111]. The application of games to online algorithms is discussed in detail by Borodin and El-Yaniv [35].

The randomized online algorithm RMark is from Fiat et al. [61]. The lower bound for randomized paging algorithms is also due to Fiat et al. [61]; the proof presented here is taken from Motwani and Raghavan [120]. Actually, the lower bound of H_k is not just asymptotically tight; McGeoch and Sleator [116] designed a randomized paging algorithm that is H_k-competitive in expectation. The barely

random algorithm RMarkBarely and its analysis are due to Böckenhauer et al. [30] and Komm and Královič [101, 102].

Hoeffding's inequality was first proven by Hoeffding [77]. Komm et al. [100] proved that if a given online problem satisfies some natural conditions, a similar statement to Theorem 2.14 is possible. In this analysis, the random variables are not independent, but form a bounded supermartingale. Instead of Hoeffding's inequality, the Azuma-Hoeffding inequality [12] is used to bound the cost of any randomized online algorithm.

The upper bound of RSki for the ski rental problem (see Theorem 2.16) is due to Karlin et al. [91], who also presented a matching lower bound. The presentation given in Section 2.8 follows Krumke and Thielen [109].

http://www.springer.com/978-3-319-42747-8

An Introduction to Online Computation
Determinism, Randomization, Advice
Komm, D.
2016, XV, 349 p. 58 illus., Hardcover
ISBN: 978-3-319-42747-8