Randomized Online Algorithms with High Probability Guarantees*

Dennis Komm[1], Rastislav Královič[2], Richard Královič[1,3], and Tobias Mömke[4]

1 **ETH Zurich, Switzerland,** dennis.komm@inf.ethz.ch
2 **Comenius University, Bratislava, Slovakia,** kralovic@dcs.fmph.uniba.sk
3 **Google Inc., Zurich, Switzerland,** richard.kralovic@dcs.fmph.uniba.sk
4 **Saarland University, Germany,** moemke@cs.uni-saarland.de

Abstract

We study the relationship between the competitive ratio and the tail distribution of randomized online problems. To this end, we define a broad class of online problems that includes some of the well-studied problems like paging, k-server and metrical task systems on finite metrics, and show that for these problems it is possible to obtain, given an algorithm with constant expected competitive ratio, another algorithm that achieves the same solution quality up to an arbitrarily small constant error with high probability; the "high probability" statement is in terms of the optimal cost. Furthermore, we show that our assumptions are tight in the sense that removing any of them allows for a counterexample to the theorem.

1998 ACM Subject Classification F.1.2 Modes of Computation, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Online Algorithms, Randomization, High Probability

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.470

1 Introduction

In online computation, we face the challenge of designing algorithms that work in environments where parts of the input are not known while parts of the output are already needed. The standard way of evaluating the quality of online algorithms is by means of *competitive analysis*, where one compares the outcome of an online algorithm to the optimal solution constructed by a hypothetical optimal offline algorithm. Since deterministic strategies are often proven to fail for the most prominent problems, randomization is used as a powerful tool to construct high-quality algorithms that outperform their deterministic counterparts against an oblivious adversary. These algorithms base their computations on the outcome of a random source; for a detailed introduction to online problems we refer the reader to the literature [5].

The most common way to measure the performance of randomized algorithms is to analyze the worst-case expected outcome and to compare it to the optimal solution. With offline algorithms, a statement about the expected outcome is also a statement about the *outcome with high probability* due to Markov's inequality and the fact that the algorithm may be executed many times to amplify the probability of success [11]. However, this amplification is not possible in online settings. As online algorithms only have one attempt to compute a

* The research is partially funded by the SNF grant 200021–146372, Deutsche Forschungsgemeinschaft grant BL511/10-1, and the VEGA 1/0671/11 grant.

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 470–481

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


reasonably good result, a statement with respect to the expected value of their competitive ratio may be rather unsatisfying. As a matter of fact, for a fixed input, it might be the case that such an algorithm produces results of a very high quality in very few cases (i. e., for a rather small number of random choices), but is unacceptably bad for the majority of random computations; still, the expected competitive ratio might suggest a better performance. Thus, if we want to have a certain guarantee that some randomized online algorithm obtains a particular quality, we must have a closer look at its analysis. In such a setting, we would like to state that the algorithm does not only perform well on average, but "almost always."

Besides a theoretical formalization of the above statement, the main contribution of this paper is to show that, for a broad class of problems, the existence of a randomized online algorithm that performs well in expectation immediately implies the existence of a randomized online algorithm that is virtually as good with high probability. Our investigations, however, need to be detailed in order to face the particularities of the framework. First, we show that it is not possible to measure the probability of success with respect to the input size, which might be considered the straightforward approach. Many of the known randomized online algorithms are naturally divided into some kind of *phases* (e. g., the algorithm for metrical task systems from Borodin et al. [6], the marking algorithm for paging from Fiat et al. [8], etc.) where each phase is processed and analyzed separately. Since the phases are independent, a high probability result (i. e., with a probability converging to 1 with an increasing number of phases) can be obtained. However, the definition of these phases is specific to each problem and algorithm. Also, there are other algorithms (e. g., the optimal paging algorithm from Achlioptas et al. [2] and many workfunction-based algorithms) that use other constructions and that are not divided into phases. As we want to establish results with high probability that are independent of the concrete algorithms, we thus have to measure this probability with respect to another parameter; we show that the cost of an optimal solution is a very reasonable quantity for this purpose. Then again it turns out that, if we consider general online problems, the notions of the expected outcome and an outcome with high probability are still not related in any way, i. e., we define problems for which these two measures are incomparable. Hence, we carefully examine both to which parameter the probability should relate and which properties we need the studied problem to fulfill to again allow a division into independent phases; finally, this allows us to construct randomized online algorithms that perform well with a probability tending to 1 with a growing size of the optimal cost. We show that this technique is applicable for a wide range of online problems.

Classically, results concerning randomized online algorithms commonly analyze their expected behavior; there are, however, a few exceptions, e. g., Leonardi et al. [14] analyze the tail distribution of algorithms for call control problems, and Maggs et al. [15] deal with online distributed data management strategies that minimize the congestion in certain network topologies.

Overview

In Section 2, we define the class of symmetric online problems and present the main result (Theorem 8). The theorem states that, for any symmetric problem that fulfills certain natural conditions, it is possible to transform an algorithm with constant expected competitive ratio r to an algorithm having a competitive ratio of $(1 + \varepsilon)r$ with high probability (with respect to the cost of an optimal solution). Section 3 is devoted to proving Theorem 8. We partition the run of the algorithm into phases such that the loss incurred by the phase changes can be amortized; however, to control the variance within one phase, we need to further subdivide the phases. Modelling the cost of single phases as dependent random variables, we obtain a

supermartingale that enables us to apply the Azuma-Hoeffding inequality and thus to obtain the result. After these investigations, we provide applications of the theorem in Section 4 where we show that our result is applicable for task systems. For the k-server problem on unbounded metric spaces and for makespan scheduling, we show that no comparable result can be obtained. We further elaborate the necessity of the conditions in Section 5. The outcome is that, even though the conditions may appear strong, a weakening prohibits a result of the same generality.

Due to space restrictions, many of the proofs are omitted and can be found in the technical report [12].

2 Preliminaries

In this section, we fix the notation for online algorithms that we use throughout the paper. Before we start, we need to briefly discuss the way in which online problems and instances are formally defined. For our investigations, we have to be very careful about these definitions. In particular, in the literature one often refers to "an online problem" when really a class of online problems is meant, which is parameterized by some problem-specific parameters. Let us give a few examples of problems that we study later in the paper. When speaking about the paging problem, we really mean the class of paging problems for, e. g., different cache sizes k. Note that there is some inconsistency in the literature as this problem is usually referred to as "paging" (and not "k-paging") while we speak of the "k-server problem." Here, k denotes the number of servers that are moved in a metric space. However, k alone is neither sufficient to specify a member from the class of paging problems nor of k-server problems. For paging, we also need the number of pages that may be requested in total, say N; for k-server the metric space (M, d) must be known, where M is a set of points and d is a distance function.

To define the above problems entirely, we still need to give even more information by speaking about how problem instances are initialized according to the parameters. For example, we need to specify how the cache is initialized for the paging problem or where the servers are located at the beginning when dealing with the k-server problem. We call this initialization the *initial situation*; for paging, the initial situation is a k-tupel of distinct integers between 1 and N, which formalizes which pages are in the cache at the beginning. Formally, we thus have to speak of an instance of the $((k, N), (s_1, \ldots, s_k))$-paging problem. In general, such a parameterized online problem is given by $(\mathcal{C}, \mathcal{I})$-$P$ where $\mathcal{C}$ is a sequence of problem-specific parameters, $\mathcal{I}$ is a set of valid initial situations, and P is the name of the union of all of theses problems. Formally, $\mathcal{I}$ is a set of valid assignments I to some of the parameters in $\mathcal{C}$ and the competitiveness guarantees of any algorithm for P must be satisfied for any $I \in \mathcal{I}$; note that, sometimes, I is also considered a part of the input instance. To end this discussion, note that in the literature, the initial situation is at times called the initial configuration; in this paper, we choose another name to distinguish it from the configuration of an algorithm (Turing machine). In the following, we will use the notation as used in the literature and omit $\mathcal{C}$; however, the initial situation I plays an important role for us and it is given together with the actual input sequence x. Let us emphasize that, if we say that some algorithm has some specific performance for a problem P, this means that this performance must be guaranteed for *all* feasible choices of $\mathcal{C}$, I, and x.

We are now ready to define online algorithms on initial situations and input instances. To keep the presentation concise, we focus on minimization problems. The same ideas translate to maximization problems by changing the point of view: we show that any optimal solution

has a low profit instead of showing that the algorithmic solution has low cost. An *online algorithm* $\mathtt{A}$ computes the output sequence $\mathtt{A}(I,x) = y = (y_1,\ldots,y_n)$, where I is an initial situation, $x = (x_1,\ldots,x_n)$ is an input sequence, and $y_i = f(I,x_1,\ldots,x_i)$ for some function f. The *cost* of the solution $\mathtt{A}(I,x)$ is denoted by $\mathrm{Cost}(I,x,y) = \mathrm{Cost}(I,x,\mathtt{A}(I,x))$. For the ease of presentation, we refer to the tuple that consists of the initial situation and the input sequence, i. e., (I,x), as the input of the problem; also, we abbreviate $\mathrm{Cost}(I,x,\mathtt{A}(I,x))$ by $\mathrm{Cost}(\mathtt{A}(I,x))$. As already mentioned, the notion of an initial situation plays an important role in the relationship between different variants of the competitive ratio; although it is usually omitted, our definition imposes no restriction on the studied problems and algorithms.

A *randomized online algorithm* $\mathtt{R}$ computes the output sequence $\mathtt{R}^{\phi}(I,x) = y = (y_1,\ldots,y_n)$ such that y_i is computed from $\phi, I, x_1,\ldots,x_i$, where ϕ is the content of a random tape. By $\mathrm{Cost}(\mathtt{R}(I,x))$ we denote the random variable (over the probability space defined by ϕ) expressing the cost of the solution $\mathtt{R}^{\phi}(I,x)$. When dealing with randomized online algorithms we compare the expected outcome to the one of an optimal algorithm. Note that, as usual in such a setting, we only consider computable problems. In the context of this paper we assume an *oblivious adversary* and say that a randomized algorithm is r-competitive if there exists a constant α such that, for every initial situation I and input sequence x, $\mathbb{E}[\mathrm{Cost}(\mathtt{R}(I,x))] \leq r \cdot \mathrm{Cost}(\mathtt{OPT}(I,x)) + \alpha$. For formal reasons, we define the competitive ratio of any (randomized) online algorithm to be 1 if both x and y are empty.

In the sequel, we analyze the notion of *competitive ratio with high probability*. Using paging, it can be shown that it does not make sense to measure the probability with respect to the input length [12]. Then again, for the practical use of paging algorithms, the instances where also the optimal algorithm makes faults are of interest. Hence, it seems reasonable to define the term *high probability* with respect to the cost of an optimal solution. In this paper, we use a strong notion of high probability requiring the error probability to be subpolynomial.

▶ **Definition 1** (Competitive Ratio w.h.p.). A randomized online algorithm $\mathtt{R}$ is r-competitive *with high probability* (w.h.p. for short) if, for any $\beta \geq 1$, there exists a constant α such that for all initial situations and input sequences (I,x) it holds that

$$\Pr[\mathrm{Cost}(\mathtt{R}(I,x)) \geq r \cdot \mathrm{Cost}(\mathtt{OPT}(I,x)) + \alpha] \leq (2 + \mathrm{Cost}(\mathtt{OPT}(I,x)))^{-\beta}.$$

First, note that the purpose of the constant 2 on the right-hand side of the formula is to properly handle inputs with a small (possibly zero) optimum. The choice of the particular constant is somewhat arbitrary (however, it should be greater than 1) since the α term on the left-hand side hides the effects. However, the two notions of the expected and the high-probability competitiveness are incomparable [12]. Nevertheless, many real-world online problems share additional properties that guarantee a close relationship between the expected and high-probability behavior. We now focus on the cost of a solution.

▶ **Definition 2** (Partition Function). A *partition function* of an online problem is a non-negative function $\mathcal{P}$ such that, for any initial situation I, the sequence of requests $x_1,\ldots,x_n$, and the corresponding solutions $y_1,\ldots,y_n$, we have

$$\mathrm{Cost}(I,(x_1,\ldots,x_n),(y_1,\ldots,y_n)) = \textstyle\sum_{i=1}^{n} \mathcal{P}(I,x_1,\ldots,x_i;y_1,\ldots,y_i).$$

In other words, for a problem with a partition function, the cost of a solution is the sum of the costs of particular answers, and the cost of each answer is independent of the future input and output. The partition function allows us to speak of the cost of a subsequence of the outputs. Note that any online problem for which the input instance may stop after each request has either a unique partition function or none, because the overall cost is fixed after

each answer. In what follows, we further restrict the behavior, and it will be convenient to think in terms of the "cost of a particular answer." We may think of online problems that have a partition function as a separate class of problems. However, all further properties depend on specific partition functions and thus requiring a "partitionability" property would be redundant.

▶ **Definition 3** (Request-Boundedness)**.** An online problem P is called *request-bounded* if, for some constant F, it has a partition function $\mathcal{P}$ such that

$$\forall I, x, y, i\colon \mathcal{P}(I, x_1, \ldots, x_i; y_1, \ldots, y_i) \leq F \text{ or } \mathcal{P}(I, x_1, \ldots, x_i;\, y_1, \ldots, y_i) = \infty.$$

We say that P is *request-bounded* according to $\mathcal{P}$.

Note that for any problem with a partition function there is a natural notion of a state; for instance, it is the content of the memory for the paging problem, the position of the servers for the k-server problem, etc. Now we provide a general definition of this notion. By $a \cdot b$, we denote the concatenation of two sequences a and b; λ denotes the empty sequence. An input $(I, x = (x_1, x_2, \ldots, x_n))$ is *feasible* with a solution $y = (y_1, y_2, \ldots, y_n)$ if starting from I, x is a request sequence that is in accord with the problem definition and for each i, y_i is a feasible answer to the request x_i with respect to I, $(x_1, x_2, \ldots, x_{i-1})$, and $(y_1, y_2, \ldots, y_{i-1})$.

▶ **Definition 4** (State)**.** Consider a partition function $\mathcal{P}$, two initial situations I and I', two sequences of requests $x = (x_1, \ldots, x_n)$ and $x' = (x'_1, \ldots, x'_m)$, and two sequences of outputs $y = (y_1, \ldots, y_n)$ and $y' = (y'_1, \ldots, y'_m)$. The triples (I, x, y) and (I', x', y') are equivalent if, for any sequence of requests $x'' = (x''_1, \ldots, x''_p)$ and a sequence of outputs $y'' = (y''_1, \ldots, y''_p)$, the input $(I, x \cdot x'')$ is feasible with a solution $y \cdot y''$ if and only if the input $(I', x' \cdot x'')$ is feasible with a solution $y' \cdot y''$, and the cost of y'' according to $\mathcal{P}$ is the same for both solutions. A *state* s of the problem is an equivalence class over the triples (I, x, y).

Let (I, x, y) be some triple in a state s. By $\texttt{Opt}_s(x')$ we denote an output sequence y' such that $y{\cdot}y'$ is a feasible solution for the input $(I, x{\cdot}x')$ and $\mathrm{Cost}(I, x{\cdot}x', y{\cdot}y') \leq \mathrm{Cost}(I, x{\cdot}x', y{\cdot}y'')$ for any feasible solution $y \cdot y''$. Note that due to the partition function, the definition of $\texttt{Opt}_s(x')$ is independent of the chosen triple (I, x, y). We sometimes simplify notation and write $\mathrm{Cost}(\texttt{Opt}_s(x'))$ instead of $\mathrm{Cost}(I, x \cdot x', y \cdot \texttt{Opt}_s(x')) - \mathrm{Cost}(I, x, y)$, as it is sufficient to know the state s and x' in order to determine the value of the function and the other parameters are clear from the context.

▶ **Definition 5** (Initial State)**.** A state s is called an *initial* state if and only if it contains some triple (I, λ, λ).

We chose this definition of states as it covers best the properties of online computations as we need them in our main theorem. An alternative definition could use task systems with infinitely many states, but the description would become less intuitive; we will return to task systems in Section 4.1.

Intuitively, a state from Definition 4 encapsulates all information about the ongoing computation of the algorithm that is relevant for evaluating the efficiency of the future processing. Usually, the state is naturally described in the problem-specific domain (content of cache, current position of servers, set of jobs accepted so far, etc.). Similar to our discussion on initial situations, we want to emphasize that a state is independent of the concrete algorithm. The internal state of an algorithm (Turing machine), which is a part of its configuration, is a different notion since it may, e. g., behave differently if the starting request had some particular value. The following properties are crucial for our approach to probability amplification.

▶ **Definition 6** (Opt-Boundedness). An online problem is called *opt-bounded* if there exists a constant B and a partition function such that $\forall s, s', x\colon |\mathrm{Cost}(\mathtt{OPT}_s(x)) - \mathrm{Cost}(\mathtt{OPT}_{s'}(x))| \leq B$.

▶ **Definition 7** (Symmetric Problem). An online problem is called *symmetric* if it is has a partition function for which every state is initial.

Note that for symmetric problems, it follows that every sequence of requests is a feasible input sequence. In particular, the input may end after any time step. Formally, any problem with a partition function may be transformed into a symmetric one simply by redefining the set of initial states. However, this transformation may significantly change the properties of the problem. Now we are going to state the main result of this paper, namely that, under certain conditions, the expected competitive ratio of symmetric problems can be achieved w.h.p.

▶ **Theorem 8.** *Consider an online problem P that is opt-bounded and symmetric according to a common partition function. Suppose there is a randomized online algorithm $\mathtt{A}$ for P with constant expected competitive ratio r. Then, for any constant $\varepsilon > 0$, there is a randomized online algorithm $\mathtt{A}'$ with competitive ratio $(1+\varepsilon)r$ w.h.p. (with respect to the optimal cost).*

3 Proof Sketch of Theorem 8

For the ease of presentation, we first provide a proof for a restricted setting where the online problem at hand is also request-bounded.

The algorithm $\mathtt{A}'$ simulates $\mathtt{A}$ and, on some specific places, performs a *reset* operation: if a part x' of the input has been read so far, and a corresponding output y' has been produced, (I, x', y') belongs to the same state as (I', λ, λ), for some initial situation I', because we are dealing with a symmetric problem; hence, $\mathtt{A}$ can be restarted by $\mathtt{A}'$ from I'.

The general idea to boost the probability of acquiring a low cost is to perform a reset each time the algorithm incurs too much cost and to use Markov's inequality to bound the probability of such an event. However, the exact value of how much is "too much" depends on the optimal cost of the input which is not known in advance. Therefore, the input is first partitioned into *phases* of a fixed optimal cost, and then each phase is cut into *subphases* based on the cost incurred so far. A reset may cause an additional expected cost of $r \cdot B$ for the subsequent phase compared to an optimal strategy starting from another state, where B is the constant of the opt-boundedness (Definition 6), i. e., B bounds the different costs between two optimal solutions for a fixed input for different states. We therefore have to ensure that the phases are long enough so as to amortize this overhead.

From now on let us consider ε, r, B, F, and α to be fixed constants; recall that F originates from the request-boundedness property of the online problem at hand (Definition 3) and α is the constant from the definition of competitiveness. The algorithm $\mathtt{A}'$ is parameterized by two parameters C and D that depend on ε, r, B, F, and α. These parameters control the lengths of the phases and subphases, respectively, such that $C + F$ delimits the optimal cost of one phase and $D + F$ delimits the cost of the solution computed by $\mathtt{A}'$ on one subphase; we require that $D > r(C + F + B + \alpha)$.

Consider an input sequence $x = (x_1, \ldots, x_n)$, an initial situation I, and let the optimal cost of the input (I, x) be between $(k-1)C$ and kC for some integer k. Then x can be partitioned into k phases $\tilde{x}_1 = (x_1, \ldots, x_{n_2-1})$, $\tilde{x}_2 = (x_{n_2}, \ldots, x_{n_3-1}), \ldots, \tilde{x}_k = (x_{n_k}, \ldots, x_n)$ in such a way that n_i is the minimal index for which the optimal cost of the input $(I, (x_1, \ldots, x_{n_i}))$ is at least $(i-1)C$. It follows that the optimal cost for one phase is at least $C - F$ and at most $C + F$, with the exception of the last phase which may be cheaper. Note that this partition

can be generated by the online algorithm itself, i. e., $\mathtt{A}'$ can determine when a next phase starts. There are only two reasons for $\mathtt{A}'$ to perform a reset: at the beginning of each phase and after incurring a cost exceeding D since the last reset. Hence, $\mathtt{A}'$ starts each phase with a reset, and the processing of each phase is partitioned into a number of subphases each of cost at least D (with the exception of the possibly cheaper last subphase) and at most $D + F$.

Now we are going to discuss the cost of $\mathtt{A}'$ on a particular input. Let us fix the input (I, x) which subsequently also fixes the indices $1 = n_1, n_2, \ldots, n_k$. Let S_i be a random variable denoting the state of the problem (according to Definition 4) just before processing request x_i, and let $W(i, j)$, $i \leq j$, be a random variable denoting the cost of $\mathtt{A}'$ incurred on the input $x_i, \ldots, x_j$. The following claim is obvious.

▶ Claim 9. If $\mathtt{A}'$ performs a reset just before processing x_i, then S_i captures all the information from the past $W(i, j)$ depends on. In particular, if we fix $S_i = s$, $W(i, j)$ does not depend on $W(l_1, l_2)$, for any $l_1 \leq l_2 \leq i$ and any state s.

The overall structure of the proof is as follows. We first show in Lemma 11 that the expected cost incurred during a phase (conditioned by the state in which the phase was entered) is at most $\mu := r(C + F + B + \alpha)/(1 - p)$, where $p := r(C + F + B + \alpha)/D < 1$. We can then consider random variables $Z_0, Z_1, \ldots, Z_k$ such that $Z_0 := k\mu$ and $Z_i := (k - i)\mu + \sum_{j=1}^{i} \overline{W}_j$ for $i > 0$, where $\overline{W}_i$ is the cost of the ith phase, clipped from above by some logarithmic bound, i. e., $\overline{W}_i := \min\{W(n_i, n_{i+1} - 1), c \log k\}$, for some suitable constant c. We show in Lemma 12 that $Z_0, Z_1, \ldots, Z_k$ form a bounded supermartingale, and then use the Azuma-Hoeffding inequality to conclude that Z_k is unlikely to be much larger than Z_0. By a suitable choice of the free parameters, this implies that Z_k is unlikely to be much larger than the expected cost of $\mathtt{A}$. Finally, we show that w.h.p. Z_k is the cost of the algorithm $\mathtt{A}'$. In order to argue about the expected cost of a given phase in Lemma 11, let us first show that a phase is unlikely to have many subphases. For the rest of the proof, let X_j be the random variable denoting the number of subphases of phase j.

▶ **Lemma 10.** *For any i, s, and any $\delta \in \mathbb{N}$ we have* $\Pr[X_i \geq \delta \mid S_{n_i} = s] \leq p^{\delta - 1}$.

Now we can argue about the expected cost of a phase.

▶ **Lemma 11.** *For any i and s it holds that* $\mathbb{E}[W(n_i, n_{i+1} - 1) \mid S_i = s] \leq \mu$.

Once the expected cost of a phase is established, we can construct the supermartingale as follows.

▶ **Lemma 12.** *For any constant $c > 0$, the sequence $Z_0, \ldots, Z_k$ is a supermartingale.*

Proof. Consider a fixed c. We have to show that for each i, $\mathbb{E}[Z_{i+1} \mid Z_0, \ldots, Z_i] \leq Z_i$. From the definition of the Z_i's it follows that $Z_{i+1} - Z_i = \overline{W}_{i+1} - \mu$. Consider any elementary event ξ from the probability space, and let $Z_i(\xi) = z_i$, for $i = 0, \ldots, k$, be the values of the corresponding random variables. We have

$$\begin{aligned}
&\mathbb{E}[Z_{i+1} \mid Z_0, \ldots, Z_i](\xi) = \mathbb{E}[Z_{i+1} \mid Z_0 = z_0, \ldots, Z_i = z_i] \\
&\quad = \mathbb{E}[Z_i + \overline{W}_{i+1} - \mu \mid Z_0 = z_0, \ldots, Z_i = z_i] \\
&\quad = z_i - \mu + \mathbb{E}[\overline{W}_{i+1} \mid Z_0 = z_0, \ldots, Z_i = z_i] \\
&\quad = z_i - \mu + \textstyle\sum_s \mathbb{E}[\overline{W}_{i+1} \mid Z_0 = z_0, \ldots, Z_i = z_i, S_{n_{i+1}} = s] \\
&\qquad\quad \cdot \Pr[S_{n_{i+1}} = s \mid Z_0 = z_0, \ldots, Z_i = z_i] \\
&\quad \leq z_i - \mu + \textstyle\sum_s \mathbb{E}[W(n_{i+1}, n_{i+2} - 1) \mid S_{n_{i+1}} = s] \cdot \Pr[S_{n_{i+1}} = s \mid Z_0 = z_0, \ldots, Z_i = z_i] \\
&\quad \leq z_i - \mu + \mu \textstyle\sum_s \Pr[S_{n_{i+1}} = s \mid Z_0 = z_0, \ldots, Z_i = z_i] = z_i = Z_i(\xi),
\end{aligned}$$

where the last inequality is a consequence of Lemma 11. ◀

Now we use the following special case of the Azuma-Hoeffding inequality [1, 10].

▶ **Lemma 13** (Azuma, Hoeffding). *Let $Z_0, Z_1, \dots$ be a supermartingale, such that $|Z_{i+1} - Z_i| < \gamma$. Then for any positive real t, $\Pr[Z_k - Z_0 \geq t] \leq \exp\left(-t^2/(2k\gamma^2)\right)$.*

In order to apply Lemma 13, we need the following bound.

▶ Claim 14. Let k be such that $c \log k > \mu$. For any i it holds that $|Z_{i+1} - Z_i| < c \log k$.

We are now ready to prove the subsequent lemma.

▶ **Lemma 15.** *Let k be such that $c \log k > \mu$. There is a constant C (depending on F, B, ε, r, α) such that $\Pr[Z_k \geq (1+\varepsilon)rkC] \leq \exp\left(-\left(k\left((1+\varepsilon)rC - \mu\right)^2\right)/(2c^2 \log^2 k)\right)$.*

Proof. Applying Lemma 13 for any positive t, we get

$$\Pr[Z_k - Z_0 \geq t] \leq \exp\left(-\frac{t^2}{2kc^2 \log^2 k}\right).$$

Noting that $Z_0 = k\mu$ and choosing $t := k((1+\varepsilon)rC - \mu)$ the statement follows. The only remaining task is to verify that $t > 0$, which can be shown by some simple calculations [12]. ◀

To prove the claim of the main theorem, we show the following bound.

▶ **Lemma 16.** *For any c and $\beta > 1$ there is a k_0 such that for any $k > k_0$*

$$\exp\left(-\frac{k\left((1+\varepsilon)rC - \mu\right)^2}{2c^2 \log^2 k}\right) \leq \frac{1}{2(2+kC)^\beta}.$$

Proof. Note that the left-hand side is of the form $\exp(-\eta k/\log^2 k)$ for some positive constant η. Clearly, for any $\beta > 1$ and large enough k, it holds that $\exp(\eta k/\log^2 k) \geq 2(2+kC)^\beta$. ◀

Combining Lemmata 15 and 16, we get the following result.

▶ **Corollary 17.** *There is a constant C (depending on F, B, ε, r, α) such that for any $\beta > 1$ there is a k_0 such that for any $k > k_0$ we have*

$$\Pr[Z_k \geq (1+\varepsilon)rkC] \leq 1/(2(2+kC)^\beta).$$

To finish the proof of the theorem we show that w.h.p. Z_k is actually the cost of the algorithm $\mathtt{A}'$.

▶ **Lemma 18.** *For any $\beta > 1$ there is a c and a k_1 such that for any $k > k_1$ $\Pr[Z_k \neq \mathrm{Cost}(\mathtt{A}'(I,x))] \leq 1/(2(2+kC)^\beta)$.*

Proof. Since $Z_k = \sum_{j=1}^{k} \min\{W(n_j, n_{j+1}-1), c\log k\}$ the event that $Z_k \neq \mathrm{Cost}(\mathtt{A}'(I,x))$ happens exactly when there is some j such that $W(n_j, n_{j+1}-1) > c\log k$. Consider any fixed j. Since the cost of a subphase is at most $D+F$, it holds that $W(n_j, n_{j+1}-1) \leq X_j(F+D)$. From Lemma 10 it follows that for any c,

$$\Pr[W(n_j, n_{j+1}-1) > c\log k] \leq \Pr\left[X_j \geq \left\lceil \frac{c\log k}{F+D} \right\rceil\right] \leq p^{\frac{c\log k}{F+D}-1}.$$

Consider the function

$$g(k) := \frac{\log\left(\frac{2k}{p}(2+kC)^\beta\right)}{\log k}.$$

It is decreasing, and $\lim_{k\mapsto\infty} g(k) = 1+\beta$. Hence, it is possible to find a constant c and a k_1 such that for any $k > k_1$ it holds that

$$c \geq \frac{F+D}{\log(1/p)} \cdot g(k).$$

From that we obtain

$$(1/p)^{\frac{c\log k}{F+D}-1} \geq 2k(2+kC)^{\beta}.$$

Thus, for this choice of c and k_1, it holds that $\Pr[W(n_j, n_{j+1}-1) > c\log k] \leq p^{\frac{c\log k}{F+D}-1} \leq 1/(2k(2+kC)^{\beta})$. Using the union bound, we conclude that the probability that the cost of any phase exceeds $c\log k$ is at most $1/(2(2+kC)^{\beta})$. ◀

Using the union bound, combining Lemma 18 and Corollary 17, and noting that the cost of the optimum is at most kC, we get the following statement.

▶ **Corollary 19.** *There is a constant C such that for any $\beta > 1$ there is a k_2 such that for any $k > k_2$ we have*

$$\Pr[\mathrm{Cost}(\mathtt{A}'(I,x)) \geq (1+\varepsilon)r\mathrm{Cost}(\mathtt{OPT}(I,x))] \leq (2+kC)^{-\beta}.$$

To conclude the proof by showing that for any $\beta > 1$ there is some α' such that

$$\Pr[\mathrm{Cost}(\mathtt{A}'(I,x)) > (1+\varepsilon)r\mathrm{Cost}(\mathtt{OPT}(I,x)) + \alpha'] \leq (2+kC)^{-\beta}$$

holds for all k, we have to choose α' large enough to cover the cases of $k < k_2$. For these cases, $\mathrm{Cost}(\mathtt{OPT}(I,x)) < k_2 C$, and hence the expected cost of $\mathtt{A}$ is at most rk_2C, and due to Lemma 11, the expected cost of $\mathtt{A}'$ is constant. The right-hand side $(2+kC)^{-\beta}$ is decreasing in k, so it is at least $(2+k_2C)^{-\beta}$, which is again a constant. From Markov's inequality it follows that there exists a constant α' such that $\Pr[\mathrm{Cost}(\mathtt{A}'(I,x)) > \alpha'] < (2+k_2C)^{-\beta}$ finishing the proof of the restricted setting.

3.1 Avoiding Request-Boundedness

All that is left to do is to show how to handle problems that are not request-bounded [12]. The main idea is to apply the restricted Theorem 8 to a modified request-bounded version of the given problem. We show that there is a modified version of the algorithm such that the computed solution has an expected competitive ratio matching the original one for the modified problem. By ensuring that *any* solution to the modified problem translates to a solution of the original problem with at most the same competitive ratio, it is enough to apply our theorem to the modified problem to obtain an analogous result for the original problem.

4 Applications and Lower Bounds

We now discuss the impact of Theorem 8 on task systems, the k-server problem, and paging. Despite being related, these problems have different flavors when analyzing them in the context of high probability results. We show that makespan scheduling does not allow for similar results.

4.1 Task Systems

The properties of online problems needed for Theorem 8 are related to the definition of task systems. There are, however, some important differences.

To analyze the relation, let us recall the definition of task systems as introduced by Borodin et al. [6]. We are given a finite state space S and a function $d\colon S \times S \to \mathbb{R}_+$ that specifies the (finite) cost to move from one state to another. The requests given as input to a task system are a sequence of $|S|$-vectors that specify, for each state, the cost to process the current task if the system resides in that state. An online algorithm for task systems aims to find a schedule such that the overall cost for transitions and processing is minimized. From now on we will call states in S *system states* to distinguish them from the states of Definition 4. The main difference between states of Definition 4 and system states is that states depend on the sequence of requests and answers; this way there may be infinitely many states. States are also more general than system states in that specific state transitions may be impossible.

▶ **Theorem 20.** *Let* $\mathtt{A}$ *be a randomized online algorithm with expected competitive ratio* r *for task systems. Then, for any* $\varepsilon > 0$*, there is a randomized online algorithm* $\mathtt{A}'$ *for task systems with competitive ratio* $(1+\varepsilon)r$ *w.h.p. (with respect to the optimal cost).*

4.2 The *k*-Server Problem

The k-server problem, introduced by Manasse et al. [16], is concerned with the movement of k servers in a metric space. Each request is a location and the algorithm has to move one of the servers to that location. If the metric space is finite, this problem is well known to be a special metrical task system. Recent progress by Bansal et al. [3] suggests that randomization might lead to an expected competitive ratio exponentially better than the deterministic lower bound.

Theorem 20 directly implies that all algorithms with a constant expected competitive ratio for the k-server problem in a finite metric space can be transformed into algorithms that have almost the same competitive ratio w.h.p.

If the metric space is infinite, an analogous result is still valid except that we have to bound the maximum transition cost by a constant. This is the case, because the proof of Theorem 20 uses the finiteness of the state space only to ensure bounded transition costs. Without the restriction to bounded distances, in general we cannot obtain a competitive ratio much better than the deterministic one w.h.p.

▶ **Theorem 21.** *Let* (M,d) *be a metric space with* $|M| = N$ *constant,* $s \in M$ *be the initial position of all servers,* ℓ *a constant and let* r *be the infimum over the competitive ratios of all deterministic online algorithms for the* k*-server problem in* (M,d) *for instances with at most* ℓ *requests. For every* $\varepsilon > 0$*, there is a metric space* (M',d') *where for any randomized online algorithm* $\mathtt{R}$ *for the* k*-server problem there is an oblivious adversary against which the solution of* $\mathtt{R}$ *has a competitive ratio of at least* $r - \varepsilon$ *with constant probability.*

▶ **Corollary 22.** *If we allow the metric to be infinite, then there is no* $(k-\varepsilon)$*-competitive online algorithm w.h.p. for the* k*-server problem for any constant* ε*.*

We simply use that the lower bound of Manesse et al. [16] satisfies the properties of Theorem 21.

4.3 Paging

Analogous to the k-server problem also the paging problem allows for the application of Theorem 8. Thus for any paging algorithm with expected competitive ratio r there is an algorithm with competitive ratio $r(1+\varepsilon)$ w.h.p.

Note that the marking algorithm is analyzed based on phases that correspond to $k+1$ distinct requests, and hence the analysis of the expected competitive ratio immediately gives the $2H_k - 1$ competitive ratio also w.h.p. However, e. g., the optimal algorithm with competitive ratio H_k due to Achlioptas et al. [2] is a distribution-based algorithm where the high probability analysis is not immediate; Theorem 8 gives an algorithm with competitive ratio $H_k(1+\varepsilon)$ w.h.p. also in this case.

4.4 Makespan Scheduling

Let us consider the classical online makespan scheduling problem $\mathrm{P}||\mathrm{C}_{\max}$ where jobs arrive one by one. It is well known that there is a tight deterministic bound $2 - 1/m$ for $m \in \{2, 3\}$ on the competitive ratio, where m is the number of machines [9, 7]. Similar to Theorem 21, we can show the following.

▶ **Theorem 23.** *For any m, let ℓ, k be constants depending on α and let r be the infimum over the competitive ratios of all deterministic online algorithms for the online makespan scheduling problem with m machines for instances with at most ℓ requests such that each request is a job with an integer processing time at most k. Then for any constant $\varepsilon > 0$, the lower bound on the competitive ratio w.h.p. is at least $r - \varepsilon$.*

The restriction to integers does not weaken the result, as we may choose a suitable scaling factor of the processing costs that allows to hide the deviation in the ε. In particular, for $m = 2$ we already obtain the tight bound on the competitive ratio for $\ell = 3$ [7] and thus there is no $(3/2 - \varepsilon)$-competitive algorithm w.h.p. for $m = 2$ and any constant ε whereas there is an online algorithm with an expected competitive ratio of $4/3$ [4].

5 Necessity of Requirements

As mentioned above, our result holds with large generality as many well-studied online problems meet the requirements we imposed. However, the assumptions of Theorem 8 require that for the problem at hand (1) every state is initial, and (2) $\forall s, s', x\colon |\mathrm{Cost}(\mathtt{OPT}_s(x)) - \mathrm{Cost}(\mathtt{OPT}_{s'}(x))| \leq B$. We can show that removing any of the conditions (1) and (2) allows for counterexamples to the theorem [12].

We would like to emphasize that the applicability of Theorem 8 is a property of problems and not of algorithms. There are problems that do not fit the assumptions of the theorem and still can be solved almost optimally by specific randomized online algorithms with high probability; for instance, albeit using a weaker notion of high probability than in the previous sections, online flow shop scheduling with unit-length tasks, $\mathrm{F}|p_{ij} = 1|\mathrm{C}_{\max}$, allows for such algorithms with respect to the number of machines [12].

Acknowledgment. The authors want to express their deepest thanks to Georg Schnitger who gave some very important impulses that contributed to the results of this paper.

References

1 K. Azuma. Weighted sums of certain dependent random variables. *Tôhoku Mathematical Journal*, 19(3):357–367, 1967.
2 D. Achlioptas, M. Chrobak, and J. Noga. Competitive analysis of randomized paging algorithms. *Theoretical Computer Science*, 234(1-2):203–218, 2000.
3 N. Bansal, N. Buchbinder, A. Mądry, and J. Naor. A polylogarithmic-competitive algorithm for the k-server problem (extended abstract). In *Proc. of FOCS 2011*, pages 267–276, 2011.
4 Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New Algorithms for an Ancient Scheduling Problem. *J. Comput. Syst. Sci.*, 51(3):359–366, 1995.
5 A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
6 A. Borodin, N. Linial, and M. E. Saks. An optimal on-line algorithm for metrical task system. *Journal of the ACM*, 39(4):745–763, 1992.
7 U. Faigle, W. Kern, and G. Turán. On the performance of on-line problems for partition problems. *Acta Cybern.*, 9(2):107–119, 1989.
8 A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.
9 R. Graham. Bounds for certain multiprocessing anomalies. *Bell System Tech. J.*, 45:1563–1581, 1966.
10 W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.
11 J. Hromkovič. *Design and analysis of randomized algorithms*. Springer-Verlag, Berlin, 2005.
12 D. Komm, R. Královič, R. Královič, and T. Mömke: Randomized online computation with high probability guarantees. CoRR abs/1302.2805, 2013.
13 E. Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009.
14 S. Leonardi, A. Marchetti-Spaccamela, A. Presciutti, and A. Rosén. On-line randomized call control revisited. *SIAM Journal on Computing*, 31(1):86–112, 2001.
15 B. M. Maggs, F. Meyer auf der Heide, B. Voecking, and M. Westermann. Exploiting locality for networks of limited bandwidth. In *Proc. of FOCS 1997*, pages 284–293, 1997.
16 M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive Algorithms for On-line Problems. *Journal of Algorithms*, 11(2):208–230, 1990.
17 D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.