Constructing Randomized Online Algorithms from Algorithms with Advice

Constructing Randomized Online Algorithms from Algorithms with Advice*

Hans-Joachim Böckenhauer[1], Sascha Geulen[2], Dennis Komm[1], and Walter Unger[2]

[1]Department of Computer Science, ETH Zürich, Universitätstrasse 6, 8092 Zürich, Switzerland
{hjb, dennis.komm}@inf.ethz.ch
[2]Department of Computer Science, RWTH Aachen University, Aachen, Germany
{sgeulen, walter.unger}@informatik.rwth-aachen.de

September 22, 2016

Abstract

In the model of advice complexity for online problems, an online algorithm is amended by an advice tape prepared by an oracle that knows the whole input in advance, and it is measured how many bits from this tape an online algorithm has to read in order to reach a certain solution quality. This model was introduced and successfully applied as a means for an improved analysis of the hardness of online problems. It was shown that it has strong connections to randomized computations, e. g., lower bounds on the advice complexity can be used to prove lower bounds also on randomized algorithms.

In this paper, we complement these known results with a method for constructing randomized online algorithms from online algorithms with advice. More precisely, we consider task systems, which model many well-known online problems such as paging, k-server, ski rental, etc. For these problems, we show how to transform any online algorithm using a small amount of advice into a randomized algorithm with almost the same competitive ratio. We achieve this by using methods from algorithmic learning theory. Using this result, we furthermore show how to translate lower bounds on randomized online computations into lower bounds for the advice complexity.

1 Introduction

Online problems are an important class of computational problems in many application areas like scheduling or planning. Here, the complete input is not available to the algorithm from the beginning of the computation, but it arrives gradually in discrete time steps. In each time step, an online algorithm has to irrevocably produce a part of the output based only on the input parts seen so far. Due to the lack of information about the future parts of the input, the solution computed in an online fashion usually cannot be guaranteed to be optimal, independent of the computing power of the online algorithm. The well-established tool to evaluate the quality of an online algorithm is the so-called competitive analysis, where one compares the output quality of an online algorithm to that of an offline algorithm that knows the whole input in advance and computes an optimal solution for it. In this paper, we only study online minimization problems, i. e., online problems

*Supported in part by SNF grant 200021-146372.

where the objective is to minimize a given cost function. Here, the competitive ratio measures the ratio between the cost of the solution computed online and an optimal solution.

Although competitive analysis is widely used to measure the hardness of online problems, it sometimes only gives a rough estimate. For a more fine-grained analysis, the concept of the information content of online problems has been introduced [14, 25, 27, 36], where one tries to measure how much information any online algorithm needs about the yet unrevealed parts of the input to compute an optimal or near-optimal solution. To this end, the online algorithm is amended by an unbounded tape of advice bits which is prepared by an all-knowing oracle before the start of the computation. The advice complexity of an online algorithm then measures the number of advice bits accessed by the algorithm. The advice complexity was introduced using a slightly different model by Dobrev et al. [25]. Then this model was refined by Emek et al. [27], Hromkovič et al. [36], and Böckenhauer et al. [14]. We will use the model of the latter two publications in this paper. Online algorithms with advice were studied for a large number of online problems such as job shop scheduling [14, 38, 53, 54], the k-server problem [13, 32, 49], metrical task systems [27], disjoint path allocation [3, 14, 30], buffer management [22], online set cover [40], graph exploration [24, 39], online independent set [23], online knapsack [15], online makespan scheduling [26, 50], online bin packing [17, 50], online Steiner tree [2] and spanning tree [4], list update [19], sleep state management [10], and online graph coloring [5, 6, 29, 51]; furthermore, generic online problems have been studied to allow computing lower bounds using a certain type of reduction [12, 18, 27].

Several connections between advice and randomization have been observed [13, 27, 38]. Since online algorithms with advice are an inherently nondeterministic model of computation, especially lower bounds on the advice complexity can be used to prove lower bounds also for randomized computations [13].

In this paper, we will explore the opposite direction. That is, we want to transform online algorithms with advice into randomized online algorithms, thus using algorithms with advice as a design method for the construction of randomized algorithms. To this end, we use the concept of online learning algorithms. In online learning, we are given a fixed set of different (deterministic or randomized) online algorithms, called experts, for some problem at hand, and our goal is to choose one of the experts for each request that computes the respective answer. We say that an online learning algorithm has the no-regret property if, on every instance, it asymptotically reaches the same quality as the best of the experts.

An online algorithm with advice that reads b advice bits can be interpreted as a set of 2^b deterministic algorithms from which the oracle chooses the best one for any concrete input instance. These deterministic algorithms can now be chosen as the experts of an online learning algorithm. We build on results from Geulen et al. [31] showing that, for a specific class of online problems, there exists an online learning algorithm that has the no-regret property. Since the best of the experts in our case corresponds to the best choice of one of the deterministic strategies by the oracle, this means that the learning algorithm is asymptotically almost as good as the advice algorithm. This result in turn also implies that any lower bound for randomized online algorithms for this specific class of problems can be turned into a similar lower bound for the advice complexity. Please note that a very similar result for transferring lower bounds from randomization to advice was obtained by Mikkelsen [47] very recently and independent of our work.

In this paper, we consider mainly a special class of online cost minimization problems, the so-called task systems as introduced by Borodin et al. [20]. In a task system, the considered instance of an online cost-minimization problem is in each time step in some state. The cost of a solution is measured by two cost functions: (1) costs for changing the state and (2) costs for processing the given request in the current state. This allows us to separate clearly between these two types of costs. Many online cost minimization problems can be modeled intuitively by a task system, including the ski-rental problem [41, 42], the paging problem [52], the list accessing problem [52], the file allocation problem [48], and the k-server problem [46]. In fact, each online cost minimization problem can be modeled by a task system using only one state. Besides being a task system, a problem has to satisfy some more properties in order for our transformation to be applicable: We assume that the processing costs are normalized to lie between 0 and 1. Switching from one expert to the other might require to change the state, these switching costs have to be bounded by a constant. Moreover, the overall costs have to grow not too slowly with growing input length.

This paper is organized as follows: In Section 2, we fix our notation and present the necessary definitions to formulate our results. In Section 3, we present the learning algorithm, formulate and prove its no-regret property and apply it to transform online algorithms with constant-size advice into randomized algorithms. Section 4 gives two applications of this result to the paging problem and the list accessing problem. In Section 5, we generalize the transformation to online algorithms using a not too large, but unbounded number of advice bits. We conclude the paper by using our technique to prove a lower bound on the advice complexity of the file allocation problem in Section 6.

2 Preliminaries

In this section, we fix our notation and give the basic definitions needed to formulate our results. We start with the definition of an online optimization problem where we use a similar notation as used in the context of request-answer games [7].

Definition 1 (Online Minimization Problems). *An* online cost minimization $\mathcal{P} = (\mathcal{R}, \mathcal{A}, f^T)$ *consists of a set of possible* requests $\mathcal{R}$*, a set of possible* answers $\mathcal{A}$*, and a sequence of* cost functions $f^T : \mathcal{R}^T \times \mathcal{A}^T \to \mathbb{R}_{\geq 0} \cup \{\perp\}$ *for all* $T \in \mathbb{N}$*. Let* $\mathcal{P} = (\mathcal{R}, \mathcal{A}, f^T)$ *be an online cost minimization problem. An* instance *of* $\mathcal{P}$ *is a* request sequence $\delta = (r^1, r^2, \ldots, r^T)$ *of* length T *with* $r^t \in \mathcal{R}$ *for all* $t \in \{1, \ldots, T\}$*. A* solution *for* $\mathcal{P}$ *is a tuple* $\psi = (A^1, A^2, \ldots, A^T)$ *with* $A^t \in \mathcal{A}$ *for all* $t \in \{1, \ldots, T\}$*. A solution* $\psi = (A^1, A^2, \ldots, A^T)$ *is* feasible *for an instance* $\delta = (r^1, r^2, \ldots, r^T)$ *of* $\mathcal{P}$ *if* $f^T(\delta, \psi) \neq \perp$*. The* total cost *of a feasible solution* $\psi = (A^1, A^2, \ldots, A^T)$ *for an instance* $\delta = (r^1, r^2, \ldots, r^T)$ *of* $\mathcal{P}$ *is* $f^T(\delta, \psi)$*.*

Note that, in the definition of online cost minimization problems, we do not make any restrictions on the sets $\mathcal{R}$ and $\mathcal{A}$. So, they can be both finite or infinite. Furthermore, note that our definition of an online optimization problem does not guarantee that an algorithm which computes a solution for a given instance is indeed an online algorithm. As already mentioned, in competitive analysis, we will compare the solution computed by an online algorithm to an optimal solution computed by an offline algorithm.

A task system consists of a set of requests, a set of states, a set of answers, an initial state, and two cost functions. The input (requests) of the task system is given piecewise to an online algorithm. The online algorithm has to process the given request immediately, based only on the input it has seen so far. For this, the online algorithm can first change the current state. Afterwards, it has to output an answer. This output cannot be changed later. In the following two definitions, we introduce the notation, as used in the context of task systems, of an instance, of a solution, of a feasible solution for an instance, and of the cost of a feasible solution for an instance.

Definition 2 (Task Systems). *A* task system $\mathcal{T} = (\mathcal{R}, \mathcal{A}, \mathcal{S}, s_0, d, p)$ *consists of a set of possible* requests $\mathcal{R}$*, a set of possible* answers $\mathcal{A}$*, a set of possible* states $\mathcal{S}$*, an* initial state s_0*, a* transition cost function $d\colon \mathcal{S} \times \mathcal{S} \to \mathbb{R}_{\geq 0} \cup \{\perp\}$*, and a* processing cost function $p\colon \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}_{\geq 0} \cup \{\perp\}$*. Let $\mathcal{T} = (\mathcal{R}, \mathcal{A}, \mathcal{S}, s_0, d, p)$ be a task system. An* instance *of $\mathcal{T}$ is a* request sequence $\delta = (r^1, r^2, \ldots, r^T)$ *of* length T *with $r^t \in \mathcal{R}$ for all $t \in \{1, \ldots, T\}$. A* solution *for $\mathcal{T}$ is a tuple $\psi = (S^0, S^1, A^1, S^2, A^2, \ldots, S^T, A^T)$ with $S^t \in \mathcal{S}$ for all $t \in \{0, \ldots, T\}$ and $A^t \in \mathcal{A}$ for all $t \in \{1, \ldots, T\}$. A state change from $s_i \in \mathcal{S}$ to $s_j \in \mathcal{S}$ is* feasible *if $d(s_i, s_j) \neq \perp$. An answer $a \in \mathcal{A}$ is* feasible *for a request $r \in \mathcal{R}$ and a state $s \in \mathcal{S}$ if $p(r, s, a) \neq \perp$. A solution $\psi = (S^0, S^1, A^1, S^2, A^2, \ldots, S^T, A^T)$ is* feasible *for an instance $\delta = (r^1, r^2, \ldots, r^T)$ of $\mathcal{T}$ if*

- *S^0 is the initial state, i. e., $S^0 = s_0$,*
- *for $t \in \{1, \ldots, T\}$, the state change from S^{t-1} to S^t is feasible, and*
- *for $t \in \{1, \ldots, T\}$, the answer A^t for request r^t in state S^t is feasible.*

Similarly to Definition 1, we do not make any restrictions on the sets $\mathcal{R}$, $\mathcal{S}$, and $\mathcal{A}$. Thus, they can also be both finite or infinite. Furthermore, the definition of a task system again does not guarantee that an algorithm which computes a solution for a given instance is indeed an online algorithm.

Next, we define the cost of a solution in some time step as the sum of the cost for the state change and the cost for the answer in this time step. Furthermore, we denote by the accumulated cost the sum over the costs until some time step and by the total cost for the whole instance the sum over the costs of all time steps.

Definition 3 (Cost of a Solution). *Let $\psi = (S^0, S^1, A^1, S^2, A^2, \ldots, S^T, A^T)$ be a feasible solution for an instance $\delta = (r^1, r^2, \ldots, r^T)$ of a task system $\mathcal{T} = (\mathcal{R}, \mathcal{A}, \mathcal{S}, s_0, d, p)$. The* transition cost *of ψ for δ in time step t is $c^t_{\psi,\mathrm{trans}}(\delta) = d(S^{t-1}, S^t)$. The* processing cost *of ψ for δ in time step t is $c^t_{\psi,\mathrm{process}}(\delta) = p(r^t, S^t, A^t)$. The* cost *of ψ for δ in time step t is $c^t_\psi(\delta) = c^t_{\psi,\mathrm{trans}}(\delta) + c^t_{\psi,\mathrm{process}}(\delta)$. The* accumulated cost *of ψ for δ until time step t is $C^t_\psi(\delta) = \sum_{i=1}^{t} c^i_\psi(\delta)$. The* total cost *of ψ for δ is $C^T_\psi(\delta)$.*

As in the definition of the accumulated and total cost, we define the accumulated and total transition/processing cost ($C^t_{\psi,\mathrm{trans}}(\delta)$, $C^t_{\psi,\mathrm{process}}(\delta)$, $C^T_{\psi,\mathrm{trans}}(\delta)$, and $C^T_{\psi,\mathrm{process}}(\delta)$) as the sum over the transition/processing cost until time step t or T, respectively.

Next, we introduce the definitions of both deterministic and randomized online algorithms for task systems. Our definition is similar to the definition given by Borodin and El-Yaniv [16] as they used it in the context of request-answer games [7].

In each time step, an online algorithm for a task system first changes the state and then determines an answer for the given request. Formally, let $\mathcal{T}$ be a task system and let $\delta = (r^1, r^2, \ldots, r^T)$ be an instance of $\mathcal{T}$. An online algorithm $\mathtt{A}$ for $\mathcal{T}$ is an algorithm that processes the requests in δ sequentially, i. e., in each time step t, $\mathtt{A}$ only knows the requests that arrived up to time step t. Note that, in general, $\mathtt{A}$ does not even know the length T of the instance δ. In other words, $\mathtt{A}$ has to compute a feasible answer $A^t_{\mathtt{A}}$ for the request r^t without knowledge of the requests in the time steps $t+1$ to T. The chosen answer $A^t_{\mathtt{A}}$ cannot be changed later. For this, for each request r^t, $\mathtt{A}$ first performs a feasible state change from state $S^{t-1}_{\mathtt{A}}$ to state $S^t_{\mathtt{A}}$, and then computes a feasible answer $A^t_{\mathtt{A}}$.

Next, we give the formal definition of a deterministic online algorithm for a task system and the solution computed by this algorithm. Note that we can transform each deterministic online algorithm for a task system to the form given in the definition.

Definition 4 (Deterministic Online Algorithm). *Let $\mathcal{T} = (\mathcal{R}, \mathcal{A}, \mathcal{S}, s_0, d, p)$ be a task system and $\delta = (r^1, r^2, \ldots, r^T)$ be an instance of $\mathcal{T}$. A* deterministic online algorithm $\mathtt{A}$ *for $\mathcal{T}$ is a sequence of* transition functions $D^t_{\mathtt{A}} \colon \mathcal{R}^t \to \mathcal{S}$ *and a sequence of* processing functions $P^t_{\mathtt{A}} \colon \mathcal{R}^t \to \mathcal{A}$ *for all $t \in \mathbb{N}$.*[1] *A deterministic online algorithm $\mathtt{A}$ processes the instance δ as follows: In each time step t, the partial request sequence $\delta\big|_t = (r^1, r^2, \ldots, r^t)$ is given to $\mathtt{A}$. First, $\mathtt{A}$ changes the state to $S^t_{\mathtt{A}} = D^t_{\mathtt{A}}(\delta\big|_t)$. Then, $\mathtt{A}$ answers the request r^t with $A^t_{\mathtt{A}} = P^t_{\mathtt{A}}(\delta\big|_t)$. Using this process, $\mathtt{A}$ constructs a* solution $\psi_{\mathtt{A}}(\delta) = (s_0, S^1_{\mathtt{A}}, A^1_{\mathtt{A}}, S^2_{\mathtt{A}}, A^2_{\mathtt{A}}, \ldots, S^T_{\mathtt{A}}, A^T_{\mathtt{A}})$ *for the instance δ. A deterministic online algorithm $\mathtt{A}$ is* feasible *if, for each instance δ, the constructed solution $\psi_{\mathtt{A}}(\delta)$ is feasible.*

The cost of a deterministic online algorithm (for a given instance) is the cost of the solution that is produced by this algorithm. Therefore, the following corollary follows directly from Definition 3.

Corollary 1 (Cost of the Solution of a Deterministic Online Algorithm). *Let $\psi_{\mathtt{A}}(\delta) = (S^0_{\mathtt{A}}, S^1_{\mathtt{A}}, A^1_{\mathtt{A}}, S^2_{\mathtt{A}}, A^2_{\mathtt{A}}, \ldots, S^T_{\mathtt{A}}, A^T_{\mathtt{A}})$ be the solution produced by a deterministic online algorithm $\mathtt{A}$ for an instance $\delta = (r^1, r^2, \ldots, r^T)$ of a task system $\mathcal{T} = (\mathcal{R}, \mathcal{A}, \mathcal{S}, s_0, d, p)$. The* transition cost *of $\mathtt{A}$ for δ in time step t is $c^t_{\mathtt{A},\mathrm{trans}}(\delta) = d(S^{t-1}_{\mathtt{A}}, S^t_{\mathtt{A}})$. The* processing cost *of $\mathtt{A}$ for δ in time step t is $c^t_{\mathtt{A},\mathrm{process}}(\delta) = p(r^t, S^t_{\mathtt{A}}, A^t_{\mathtt{A}})$. The* cost *of $\mathtt{A}$ for δ in time step t is $c^t_{\mathtt{A}}(\delta) = c^t_{\mathtt{A},\mathrm{trans}}(\delta) + c^t_{\mathtt{A},\mathrm{process}}(\delta)$. The* accumulated cost *of $\mathtt{A}$ for δ until time step t is $C^t_{\mathtt{A}}(\delta) = \sum_{i=1}^t c^i_{\mathtt{A}}(\delta)$. The* total cost *of $\mathtt{A}$ for δ is $C^T_{\mathtt{A}}(\delta)$.*

We define the accumulated and total transition/processing cost ($C^t_{\mathtt{A},\mathrm{trans}}(\delta)$, $C^t_{\mathtt{A},\mathrm{process}}(\delta)$, $C^T_{\mathtt{A},\mathrm{trans}}(\delta)$, and $C^T_{\mathtt{A},\mathrm{process}}(\delta)$) as the sum over the transition/processing costs until time step t or T, respectively. Note that in general it is not necessary that an online algorithm knows the transition cost function or the processing cost function.

To measure the quality of an online algorithm, we frequently use the competitive ratio, which can be formally defined as follows.

Definition 5 (Competitive Ratio). *Let $\mathcal{T} = (\mathcal{R}, \mathcal{A}, \mathcal{S}, s_0, d, p)$ be a task system, let $\mathtt{A}$ be a deterministic online algorithm for $\mathcal{T}$, and let $\mathtt{Opt}$ be an optimal offline algorithm for $\mathcal{T}$, that is, an algorithm that computes an optimal solution for each instance of $\mathcal{T}$. The algorithm $\mathtt{A}$ is called* c-competitive *for $\mathcal{T}$ if there exists a non-negative constant α such that, for any instance δ of $\mathcal{T}$,*

$$C^T_{\mathtt{A}}(\delta) \le c \cdot C^T_{\mathtt{Opt}}(\delta) + \alpha\,.$$

We also call c the competitive ratio *of $\mathtt{A}$ on $\mathcal{T}$. If $\alpha = 0$, we say that $\mathtt{A}$ is* strictly c-competitive*. Moreover, we call $\mathtt{A}$* optimal*, if it is strictly 1-competitive.*

In an offline setting, one can always exclude a constant number of special input instances from the analysis by simply preprocessing the solutions and including them into the description of the algorithm. This is not possible in an online scenario, but the constant α in Definition 5 can be used for the same purpose. We will make heavy use of this possibility in our analyses.

In many online cost minimization problems, the (expected) total cost of an online algorithm can be greatly reduced if we allow the usage of randomization. A randomized online algorithm is an online algorithm that has access to a tape containing random bits. The bits on this tape are chosen uniformly at random. Note that we can transform the uniform distribution of the random bits to an arbitrary probability distribution using inverse transform sampling [21]. In each time step, the

[1] Although we defined an online algorithm as infinite sequences of transition and processing functions, we assume that these sequences have a finite description.

randomized online algorithm can read an arbitrary finite number of bits from this tape. Thus, if we fix the bits on the tape, the considered randomized online algorithm is, in fact, a deterministic online algorithm. Using this idea, we model a randomized online algorithm as a deterministic online algorithm with access to a tape containing bits that were chosen previously uniformly at random. In the following definition, we introduce the concept of randomized online algorithms for task systems formally.

Definition 6 (Randomized Online Algorithm). *Let $\mathcal{T} = (\mathcal{R}, \mathcal{A}, \mathcal{S}, s_0, d, p)$ be a task system, $\delta = (r^1, r^2, \ldots, r^T)$ be an instance of $\mathcal{T}$, and $\mathtt{A}'(x)$ be a feasible deterministic online algorithm for $\mathcal{T}$ using a (possibly unbounded bit sequence) x as a parameter. A* randomized online algorithm $\mathtt{A}$ *for $\mathcal{T}$ chooses in the first time step a bit sequence x uniformly at random and then uses $\mathtt{A}'(x)$ for the whole instance δ.*

For an instance δ of length T given to a randomized online algorithm $\mathtt{A}$, the total cost of the solution $\psi_{\mathtt{A}}(\delta)$ computed by $\mathtt{A}$ is the total cost of the solution of the deterministic online algorithm $\mathtt{A}'(x)$, where x is the parameter chosen by $\mathtt{A}$ in the first time step. Thus, the total cost $C^T_{\mathtt{A}}(\delta)$ of $\mathtt{A}$ is the total cost $C^T_{\mathtt{A}'(x)}(\delta)$ of $\mathtt{A}'(x)$. But this means that both the solution $\psi_{\mathtt{A}}(\delta)$ and the total cost $C^T_{\mathtt{A}}(\delta)$ of $\mathtt{A}$ are random variables. This is the reason why we usually use the expected total cost $\mathbf{E}\left[C^T_{\mathtt{A}}(\delta)\right]$ in the analysis of randomized online algorithms. Note that the equation for the expected accumulated cost of $\mathtt{A}$ given in the following corollary follows by linearity of expectation.

Corollary 2 (Expected Cost of the Solution of a Randomized Online Algorithm). *Let $\mathcal{T} = (\mathcal{R}, \mathcal{A}, \mathcal{S}, s_0, d, p)$ be a task system, $\delta = (r^1, r^2, \ldots, r^T)$ be an instance of $\mathcal{T}$, $\mathtt{A}$ be a randomized online algorithm for $\mathcal{T}$, and $\mathbf{E}[c^t_{\mathtt{A}}(\delta)]$ be the expected cost of $\mathtt{A}$ for δ in time step t. The* expected accumulated cost *of $\mathtt{A}$ for δ until time step t is $\mathbf{E}[C^t_{\mathtt{A}}(\delta)] = \sum_{i=1}^{t} \mathbf{E}[c^i_{\mathtt{A}}(\delta)]$. The* expected total cost *of $\mathtt{A}$ for δ is $\mathbf{E}\left[C^T_{\mathtt{A}}(\delta)\right]$.*

We define the expected accumulated and expected total transition/processing cost

$$\mathbf{E}\left[C^t_{\mathtt{A},\text{trans}}(\delta)\right], \mathbf{E}\left[C^t_{\mathtt{A},\text{process}}(\delta)\right], \mathbf{E}\left[C^T_{\mathtt{A},\text{trans}}(\delta)\right], \text{ and } \mathbf{E}\left[C^T_{\mathtt{A},\text{process}}(\delta)\right]$$

as the sum over the expected transition/processing costs until time step t or T, respectively.

The notion of competitive ratio can be extended to randomized online algorithms in a straightforward way by just using the expected total cost instead of the (deterministic) total cost in Definition 5.

We now give a formal definition of online algorithms with advice.

Definition 7 (Online Algorithm with Advice). *Let $\mathcal{T} = (\mathcal{R}, \mathcal{A}, \mathcal{S}, s_0, d, p)$ be a task system, let $\delta = (r^1, r^2, \ldots, r^T)$ be an instance of $\mathcal{T}$. An* online algorithm $\mathtt{A}$ with advice *computes the output sequence $\psi^{\phi}_{\mathtt{A}}(\delta) = (s_0, S^1_{\mathtt{A}}, A^1_{\mathtt{A}}, S^2_{\mathtt{A}}, A^2_{\mathtt{A}}, \ldots,\ S^T_{\mathtt{A}}, A^T_{\mathtt{A}})$ for the instance δ such that $S^i_{\mathtt{A}}$ and $A^i_{\mathtt{A}}$ are computed from $\phi, r^1, \ldots, r^i$, where ϕ is the content of the advice tape, i. e., an infinite binary sequence. Let $C^T_{\mathtt{A},\phi}(\delta)$ denote the total cost of $\mathtt{A}$ with advice tape ϕ on the instance δ. The algorithm $\mathtt{A}$ is* c-competitive with advice complexity $b(T)$ *if there exists a constant α such that, for every T and for each input sequence δ of length at most T, there exists some ϕ such that*

$$C^T_{\mathtt{A},\phi}(\delta) \leq c \cdot C^T_{\mathtt{Opt}}(\delta) + \alpha$$

and at most the first $b(T)$ bits of ϕ have been accessed during the computation of $\mathtt{A}$ with advice ϕ on δ. If $\alpha = 0$, $\mathtt{A}$ is called strictly *c-competitive.*

Next, we introduce the notation used in the context of online learning algorithms and regret minimization. Regret minimization for online learning algorithms was first studied by Hannan [35]. A general introduction to online learning algorithms and regret minimization can be found in the textbook by Blum [8] or the survey by Blum and Mansour [9].

An online learning algorithm for an online cost minimization problem $\mathcal{P}$ is a deterministic or randomized online algorithm that has access to a set $\mathcal{E}$ of feasible deterministic or randomized online algorithms for $\mathcal{P}$. We call $\mathcal{E}$ the set of experts and each online algorithm in $\mathcal{E}$ an expert. We make two assumptions about $\mathcal{E}$. First, we assume that the cost of each expert in $\mathcal{E}$ in each time step is between zero and one.

Assumption 1 (Normalized Expert Costs). *For each expert in $\mathcal{E}$ and each time step, the cost of this expert in this time step is in the interval* $[0, 1]$*.*

This assumption will ensure that the cost of an expert that was chosen by an online learning algorithm `A` has only a limited effect on the (expected) total cost of `A` for the whole instance. For each online cost minimization problem $\mathcal{P}$ and each set of experts $\mathcal{E}$ for $\mathcal{P}$, we can normalize $\mathcal{P}$ in such a way that Assumption 1 holds: Let C^* be an upper bound on the cost in some time step of the experts in $\mathcal{E}$ that can appear while processing an arbitrary instance of $\mathcal{P}$. We can scale the cost functions of $\mathcal{P}$ by $1/C^*$.

An online learning algorithm `A` uses the decisions of the experts in $\mathcal{E}$ in order to compute a feasible solution for the online cost minimization problem $\mathcal{P}$. For this, `A` selects an expert in each time step and follows the decision made by the chosen expert. For a task system, this means that `A` goes to the same state as the chosen expert and answers the request with the same answer as the chosen expert. The cost of `A` in time step t is composed of both the cost for changing the expert (if `A` changed the expert) and the cost of the expert chosen in this time step. Assumption 2 will ensure that switching between experts has only a limited effect on the (expected) total cost of `A` for the whole instance.

Assumption 2 (Bounded Switching Costs). *Let $\mathcal{T} = (\mathcal{R}, \mathcal{A}, \mathcal{S}, s_0, d, p)$ be a task system, $\mathcal{E} = \{e_1, \dots, e_N\}$ be a set of experts for $\mathcal{T}$, $\delta = (r^1, r^2, \dots, r^T)$ be an instance of $\mathcal{T}$, and, for $i \in \{1, \dots, N\}$, $\psi_{e_i}(\delta) = (s_0, S^1_{e_i}, A^1_{e_i}, S^2_{e_i}, A^2_{e_i}, \dots, S^T_{e_i}, A^T_{e_i})$ be a solution for δ computed by expert $e_i \in \mathcal{E}$. For all time steps $t \in \{1, \dots, T-1\}$ and all experts $e_i, e_j \in \mathcal{E}$, we have $d(S^t_{e_i}, S^{t+1}_{e_j}) \leq B$, where $B \in \mathbb{R}_{\geq 0}$.*

If Assumption 2 holds, we say that $\mathcal{E}$ has *switching cost of at most B*. So, a task system with a set of experts $\mathcal{E}$ has switching cost at most B if the transition cost function allows a change of the chosen expert with cost at most B. For many online cost minimization problems, Assumption 2 holds: For instance, in the paging problem (Definition 13), the switching cost is bounded from above by the size of the cache. Note that, if $\mathcal{E}$ contains randomized online algorithms, Assumption 2 must hold for all possible solutions of these experts, not only in expectation.

The choice of the expert in time step t is based on the evaluation of the experts (cost of the experts) until time step $t-1$. After each time step, the evaluation of each expert in this time step is given to `A`. We explain the evaluation process in detail later in this section.

The goal of the online learning algorithm is to use the decisions of the experts in such a way that it minimizes its regret, i. e., the (expected) total cost of the online learning algorithm has to be close to the (expected) total cost of the best expert in $\mathcal{E}$ chosen in hindsight for each instance δ of the online cost minimization problem $\mathcal{P}$.

Next, we give the formal definition of a deterministic online learning algorithm for a task system and the solution computed by this algorithm.

Definition 8 (Deterministic Online Learning Algorithm). *Let $\mathcal{T} = (\mathcal{R}, \mathcal{A}, \mathcal{S}, s_0, d, p)$ be a task system, $\delta = (r^1, r^2, \ldots, r^T)$ be an instance of $\mathcal{T}$, and $\mathcal{E} = \{e_1, \ldots, e_N\}$ be a set of deterministic or randomized online algorithms for $\mathcal{T}$ that satisfies both Assumptions 1 and 2. Furthermore, let $D^t_{e_i} \colon \mathcal{R}^t \to \mathcal{S}$ be the set of transition functions and $P^t_{e_i} \colon \mathcal{R}^t \to \mathcal{A}$ be the sequence of processing functions of online algorithm $e_i \in \mathcal{E}$, for all $t \in \mathbb{N}$.*[2] *A* deterministic online learning algorithm $\mathtt{A}$ *for $\mathcal{T}$ equipped with the set $\mathcal{E} = \{e_1, \ldots, e_N\}$ of N* experts *is a set of* expert selection functions *$E^t_{\mathtt{A}} \colon ([0,1]^t)^N \to \mathcal{E}$ for all $t \in \mathbb{N}$. A deterministic online learning algorithm $\mathtt{A}$ processes the instance δ as follows: In each time step t, the partial request sequence $\delta\big|_t = (r^1, r^2, \ldots, r^t)$ is given to $\mathtt{A}$. In time step 1, $\mathtt{A}$ selects the expert $e^1 = E^1_{\mathtt{A}}[0, \ldots, 0]$; in time step $t > 1$, $\mathtt{A}$ selects the expert*

$$e^t = E^t_{\mathtt{A}}\Big[\Big(0, b^1_1(\delta\big|_1), \ldots, b^{t-1}_1(\delta\big|_{t-1})\Big), \ldots, \Big(0, b^1_N(\delta\big|_1), \ldots, b^{t-1}_N(\delta\big|_{t-1})\Big)\Big] .$$

Then, $\mathtt{A}$ changes the state to $S^t_{\mathtt{A}} = D^t_{e^t}(\delta\big|_t)$. Afterwards, $\mathtt{A}$ answers the request r^t with $A^t_{\mathtt{A}} = P^t_{e^t}(\delta\big|_t)$. Finally, the evaluation vector $\Big(b^t_1(\delta\big|_t), \ldots, b^t_N(\delta\big|_t)\Big)$ *is revealed to $\mathtt{A}$. Using this process, $\mathtt{A}$ constructs a* solution *$\psi_{\mathtt{A}}(\delta) = (s_0, S^1_{\mathtt{A}}, A^1_{\mathtt{A}}, S^2_{\mathtt{A}}, A^2_{\mathtt{A}}, \ldots, S^T_{\mathtt{A}}, A^T_{\mathtt{A}})$ for the instance δ. A deterministic online learning algorithm $\mathtt{A}$ is* feasible *if, for each instance δ, the constructed solution $\psi_{\mathtt{A}}(\delta)$ is feasible.*

Note that if an expert is a randomized online algorithm, the results of both its transition and processing functions are random variables. Using the definition of deterministic online learning algorithms, we define a randomized online learning algorithm as a deterministic online learning algorithm with a randomly chosen parameter.

Definition 9 (Randomized Online Learning Algorithm). *Let $\mathcal{T} = (\mathcal{R}, \mathcal{A}, \mathcal{S}, s_0, d, p)$ be a task system, $\delta = (r^1, r^2, \ldots, r^T)$ be an instance of $\mathcal{T}$, and $\mathtt{A}'(x)$ be a feasible deterministic online learning algorithm for $\mathcal{T}$ equipped with the set $\mathcal{E}$ of experts using a parameter x. A* randomized online learning algorithm $\mathtt{A}$ *for $\mathcal{T}$ equipped with the set $\mathcal{E}$ of experts chooses, in the first time step, a bit sequence x uniformly at random and then uses $\mathtt{A}'(x)$ for the whole instance δ.*

As in the definition of the cost of a deterministic or randomized online algorithm, the cost of a deterministic or randomized online learning algorithm is the cost of the solution that is produced by the online learning algorithm. Therefore, we use Corollaries 1 and 2 also in the context of online learning algorithms. Furthermore, we define the (expected) accumulated and (expected) total transition/processing cost ($C^t_{\mathtt{A},\text{trans}}(\delta)$, $C^t_{\mathtt{A},\text{process}}(\delta)$, $C^T_{\mathtt{A},\text{trans}}(\delta)$, and $C^T_{\mathtt{A},\text{process}}(\delta)$) as the sum over the considered costs until time step t or T, respectively.

Note that, in the standard setting of online learning algorithms that was studied, for example, in Hannan [35], Blum [8], and Blum and Mansour [9], the cost for changing the expert is zero. Hence, the model presented here extends the standard setting by switching costs. The standard setting corresponds in our model to the setting where all transition costs are set to zero, i. e., where, for all $s_i, s_j \in \mathcal{S}$, $d(s_i, s_j) = 0$.

The choice of the expert in time step t is done using the expert selection function $E^t_{\mathtt{A}}$ of the online learning algorithm $\mathtt{A}$ in time step t. The inputs of this function are composed of all evaluation vectors until time step $t-1$. We discuss the evaluation vector in the next definition. Depending on the considered model, it contains either only the cost of the chosen expert or the cost of all experts. Nevertheless, the choice of an expert in an online learning algorithm does not depend on the cost in time step t. We can interpret this selection process as follows: In each time step, the online learning

[2] Again, we assume that this infinite sequence has a finite description.

algorithm chooses an expert. It uses the expert as a black box for answering the request in this time step. Afterwards, the evaluation (cost) vector is revealed. By this, one might assume that an adversary chooses the cost for each expert in each time step arbitrarily from $[0,1]$ after the online learning algorithm answered the request.

Definition 10 (Information Models). *Let $\mathcal{T} = (\mathcal{R}, \mathcal{A}, \mathcal{S}, s_0, d, p)$ be a task system, let $\delta = (r^1, r^2, \ldots, r^T)$ be an instance of $\mathcal{T}$, $\mathtt{A}$ be an online learning algorithm for $\mathcal{T}$, and e^t be the expert chosen by $\mathtt{A}$ in time step t.*

- *In the* full information model, *after $\mathtt{A}$ has answered the request r^t in time step t, the evaluation vector $\left(b_1^t(\delta|_t), \ldots, b_N^t(\delta|_t)\right) = \left(c_{e_1}^t(\delta), \ldots, c_{e_N}^t(\delta)\right)$ is revealed to $\mathtt{A}$.*
- *In the* partial information model, *after $\mathtt{A}$ has answered the request r^t in time step t, the evaluation vector $\left(b_1^t(\delta|_t), \ldots, b_N^t(\delta|_t)\right) = (\perp, \ldots, \perp, c_{e^t}^t(\delta), \perp, \ldots, \perp)$ is revealed to $\mathtt{A}$, where $\perp$ means undefined.*

We will focus on the full information model in what follows. We study online learning algorithms only in the context of task systems and expert sets with switching cost at most B. By this, each state change done by an online learning algorithm $\mathtt{A}$ is feasible and has cost at most B. Thus, the solution constructed by $\mathtt{A}$ is also always feasible. This means that, in the context of online learning algorithms, we can split the cost experienced by an online learning algorithm $\mathtt{A}$ into two types of costs.

- *Switching costs* $c_{\mathtt{A},\text{switch}}^t(\delta)$: These costs are caused by the online learning algorithm. Whenever the online learning algorithm changes the expert from time step $t-1$ to time step t, i. e., $e^t \neq e^{t-1}$, the online learning algorithm has to change the state from $S_{e^{t-1}}^{t-1}$ to $S_{e^t}^t$. Since we consider here only task systems and expert sets with switching cost at most B, these costs are

$$c_{\mathtt{A},\text{switch}}^t(\delta) = \begin{cases} c_{\mathtt{A},\text{trans}}^t(\delta) = d(S_{\mathtt{A}}^{t-1}, S_{\mathtt{A}}^t) = d(S_{e^{t-1}}^{t-1}, S_{e^t}^t) \leq B & \text{if } e^t \neq e^{t-1} \\ 0 & \text{else} \end{cases} .$$

- *Handling costs* $c_{\mathtt{A},\text{hand}}^t(\delta)$: These costs are caused by the chosen expert e^t in time step t for answering the request r^t given to the online learning algorithm. If the online learning algorithm changes the expert, i. e., $e^t \neq e^{t-1}$, the switching costs also include the transition cost for changing the state. Thus, in this case, the handling costs are given by the processing cost of e^t. On the other hand, if $e^t = e^{t-1}$, the handling costs of the online learning algorithm $\mathtt{A}$ are the costs of the chosen expert e^t. These costs consist of the transition cost for changing the state and the processing cost for answering the request in time step t. Thus,

$$c_{\mathtt{A},\text{hand}}^t(\delta) = \begin{cases} c_{\mathtt{A},\text{process}}^t(\delta) = p(r^t, S_{\mathtt{A}}^t, A_{\mathtt{A}}^t) & \\ \qquad = p(r^t, S_{e^t}^t, A_{e^t}^t) & \\ \qquad \leq d(S_{e^t}^{t-1}, S_{e^t}^t) + p(r^t, S_{e^t}^t, A_{e^t}^t) = c_{e^t}^t(\delta) & \text{if } e^t \neq e^{t-1} \\ c_{\mathtt{A},\text{trans}}^t(\delta) + c_{\mathtt{A},\text{process}}^t(\delta) = d(S_{\mathtt{A}}^{t-1}, S_{\mathtt{A}}^t) + p(r^t, S_{\mathtt{A}}^t, A_{\mathtt{A}}^t) & \\ \qquad = d(S_{e^t}^{t-1}, S_{e^t}^t) + p(r^t, S_{e^t}^t, A_{e^t}^t) = c_{e^t}^t(\delta) & \text{else} \end{cases} .$$

Note that, in both cases, the handling costs of $\mathtt{A}$ are bounded from above by $c_{e^t}^t(\delta)$, and since we assume that $c_{e_i}^t(\delta) \in [0,1]$ for all experts $e_i \in \mathcal{E}$, instances δ, and time steps t (Assumption 1), also $c_{\mathtt{A},\text{hand}}^t(\delta) \in [0,1]$ holds.

So, the cost of an online learning algorithm $\mathtt{A}$ for an instance δ in time step t is the sum of the switching costs and the handling costs, i. e., $c^t_{\mathtt{A}}(\delta) = c^t_{\mathtt{A},\text{switch}}(\delta) + c^t_{\mathtt{A},\text{hand}}(\delta)$. Note that $c^t_{\mathtt{A}}(\delta)$ is the same as $c^t_{e^t}(\delta)$ if $\mathtt{A}$ does not change the expert. Otherwise, it is bounded from above by $B + c^t_{e^t}(\delta)$. As in the definition of the (expected) accumulated and (expected) total transition/processing cost of an online algorithm, we define the (expected) accumulated and (expected) total handling/switching cost ($C^t_{\mathtt{A},\text{hand}}(\delta)$, $C^t_{\mathtt{A},\text{switch}}(\delta)$, $C^T_{\mathtt{A},\text{hand}}(\delta)$, and $C^T_{\mathtt{A},\text{switch}}(\delta)$) as the sum over the (expected) handling/switching costs until time step t or T, respectively. Therefore, the (expected) total cost of an online learning algorithm $\mathtt{A}$ for an instance δ is the sum of the (expected) total switching costs and the (expected) total handling costs, i. e., $C^T_{\mathtt{A}}(\delta) = C^T_{\mathtt{A},\text{switch}}(\delta) + C^T_{\mathtt{A},\text{hand}}(\delta)$. Furthermore, note that, in the standard setting of online learning algorithms, all transition costs are zero. Consequently, $c^t_{\mathtt{A}}(\delta) = c^t_{\mathtt{A},\text{hand}}(\delta) = p(r^t, S^t_{\mathtt{A}}, A^t_{\mathtt{A}}) = p(r^t, S^t_{e^t}, A^t_{e^t})$.

As already mentioned, the goal of the online learning algorithm $\mathtt{A}$ is to use the decisions of the experts in such a way that the (expected) total cost of $\mathtt{A}$ is close to the (expected) total cost of the best expert, i. e., $\mathtt{A}$ aims at minimizing its external regret as defined in the following definition.

Definition 11 (Regret Minimization). *Let $\mathcal{T} = (\mathcal{R}, \mathcal{A}, \mathcal{S}, s_0, d, p)$ be a task system and* $\mathtt{A}$ *be an online learning algorithm for $\mathcal{T}$ equipped with the set $\mathcal{E} = \{e_1, \ldots, e_N\}$ of experts. For an instance $\delta = (r^1, r^2, \ldots, r^T)$, we denote by* $\mathtt{best}$ *the* best expert *in $\mathcal{E}$ for the instance δ, i. e., $C^T_{\mathtt{best}}(\delta) \leq C^T_{e_i}(\delta)$ for all $e_i \in \mathcal{E}$. The* (external) regret *of* $\mathtt{A}$ *on an instance δ of length T is*

$$R^T_{\mathtt{A}}(\delta) \; = \; C^T_{\mathtt{A}}(\delta) - C^T_{\mathtt{best}}(\delta) \;\; .$$

The online learning algorithm $\mathtt{A}$ *is a* no-regret online learning algorithm *if*

$$\lim_{T\to\infty} \frac{\max_{\delta=(r^1,r^2,\ldots,r^T)} \; R^T_{\mathtt{A}}(\delta)}{T} \; = \; 0 \;\; .$$

As a consequence, an online learning algorithm $\mathtt{A}$ is a no-regret online learning algorithm if, for each instance δ of length T, its regret is sublinear in T, i. e., $R^T_{\mathtt{A}}(\delta) \in o(T)$. Note that, in the definition above, we use the expected total cost if the online learning algorithm or the best expert is a randomized online algorithm.

3 Randomized Algorithms from Algorithms with Constant Advice

In this section, we describe how we can employ an online learning algorithm to transform an online algorithm with constant-size advice into a randomized algorithm achieving almost the same competitive ratio.

We consider the *shrinking dartboard algorithm (*$\mathtt{SD}$*)* [31], which is a variant of the well-known randomized weighted majority algorithm [44]. Given a set $\mathcal{E} = \{1, \ldots, N\}$ of N experts, the idea of the randomized weighted majority algorithm $\mathtt{RWM}$ is to maintain, for each expert, a probability of being chosen, which depends on the accumulated cost that the expert has experienced in the past. For this, $\mathtt{RWM}$ uses a weight w^t_i for each expert $i \in \mathcal{E}$ in time step t. Initially, we set all weights to 1 and use the uniform distribution over the experts. Afterwards, the probability q^t_i, that the chosen expert in time step t is expert i, is controlled by the current weight w^t_i of the expert. This weight itself depends on the weights $w^{t'}_i$ and costs $c^{t'}_i$ in the time steps t' up to $t-1$ and a parameter $\eta \in [0, 2/3]$. We call η the *learning rate* of the algorithm. The main improvement of the shrinking dartboard algorithm is to change the chosen expert only if it causes significant handling costs. Algorithm 1 specifies how the experts are chosen and introduces the notation. Our algorithm

Algorithm 1 (`Shrinking Dartboard (SD)`)

```
1:  $w_i^1 = 1, q_i^1 = \frac{1}{N}$, for all $i \in \mathcal{E}$
2:  choose expert $e^1$ at random according to $Q^1 = (q_1^1, \ldots, q_N^1)$
3:  change the state from $s_0$ to $S_{e^1}^1$
4:  use answer $A_{e^1}^1$ for request $r^1$
5:  for $t = 2, \ldots, T$ do
6:      $w_i^t = w_i^{t-1}(1-\eta)^{c_i^{t-1}}$, for all $i \in \mathcal{E}$
7:      $q_i^t = w_i^t / \left(\sum_{j=1}^N w_j^t\right)$, for all $i \in \mathcal{E}$
8:      with probability $w_{e^t}^t / w_{e^t}^{t-1}$ do
9:          set $e^t = e^{t-1}$ (no expert change)
10:     else
11:         choose $e^t$ at random according to $Q^t = (q_1^t, \ldots, q_N^t)$
12:     end
13:     change the state from $S_{e^{t-1}}^{t-1}$ to $S_{e^t}^t$
14:     use answer $A_{e^t}^t$ for request $r^t$
15: end for
```

uses a set $\mathcal{E} = \{1, \ldots, N\}$ of N experts. Again, we call η the learning rate of the algorithm and denote the total weight in time step t by $W^t = \sum_{i=1}^N w_i^t$.

In the first time step, Algorithm 1 chooses an expert e^1 from the set $\mathcal{E}$ uniformly at random (line 2) and processes the first request as it is done by this expert, i. e., it changes the state to $S_{e^1}^1 = D_{e^1}^1(\delta|_1)$ (line 3) and answers the request r^1 in state $S_{e^1}^1$ with $A_{e^1}^1 = P_{e^1}^1(\delta|_1)$ (line 4). In each time step $t > 1$, Algorithm 1 first updates the weights (line 6) and the probabilities of each expert (line 7). With probability $w_{e^t}^t / w_{e^t}^{t-1} = (1-\eta)^{c_i^{t-1}}$ (line 8), the algorithm does not change the expert (line 9). Otherwise, it chooses an expert e^t at random according to the probability distribution $Q^t = (q_1^t, \ldots, q_N^t)$ (line 11). Afterwards, the algorithm changes the state from $S_{e^{t-1}}^{t-1} = D_{e^{t-1}}^{t-1}(\delta|_{t-1})$ to $S_{e^t}^t = D_{e^t}^t(\delta|_t)$ (line 13) and answers the request r^t in state $S_{e^t}^t$ with $A_{e^t}^t = P_{e^t}^t(\delta|_t)$ (line 14).

The algorithm is called *Shrinking Dartboard* (`SD`) as it can be illustrated in terms of a dartboard shrinking over time (Figure 1). Initially, the dartboard is a disc divided into N equally sized sectors, one for each expert (Figure 1a). The total area covered by the disc has size N so that each sector has size 1. In time step 1, `SD` chooses an expert by *throwing a dart* to the dartboard, that is, it picks a point from the disc uniformly at random and chooses that expert into which sector this point falls (Figure 1b). Over time, the expert's sectors shrink as illustrated in Figure 1c. In particular, the size of the area covered by expert i's sector in time step t, denoted as the *allowed area* in time step t, corresponds to the weight w_i^t as specified in Algorithm 1. In time step $t > 1$, `SD` chooses an expert as follows: If the previously picked point is still in the allowed area, then `SD` does not change the expert. This happens with probability $w_{e^t}^t / w_{e^t}^{t-1} = (1-\eta)^{c_i^{t-1}}$ and corresponds to line 9 of Algorithm 1. Otherwise, `SD` *throws a new dart*, that is, it picks a point uniformly at random from the area covered by the sectors of all experts and chooses the expert into which sector this point falls. This happens with probability $1 - w_{e^t}^t / w_{e^t}^{t-1}$ and corresponds to line 11 of Algorithm 1.

The online learning algorithm `SD` causes costs in line 3, 4, 13, and 14 of Algorithm 1. The costs in line 3 and 4 are produced only by the first expert e^1 chosen by `SD`. Thus, these costs are handling costs. Thus, $c_{\mathtt{SD}}^1 = c_{\mathtt{SD},\text{hand}}^1 = d(s_0, S_{e^1}^1) + p(r^1, S_{e^1}^1, A_{e^1}^1) = c_{e^1}^1$. Consider now a time step $t > 1$. The costs of this time step are caused by line 13 and 14. If `SD` does not change the expert, i. e., $e^t = e^{t-1}$,

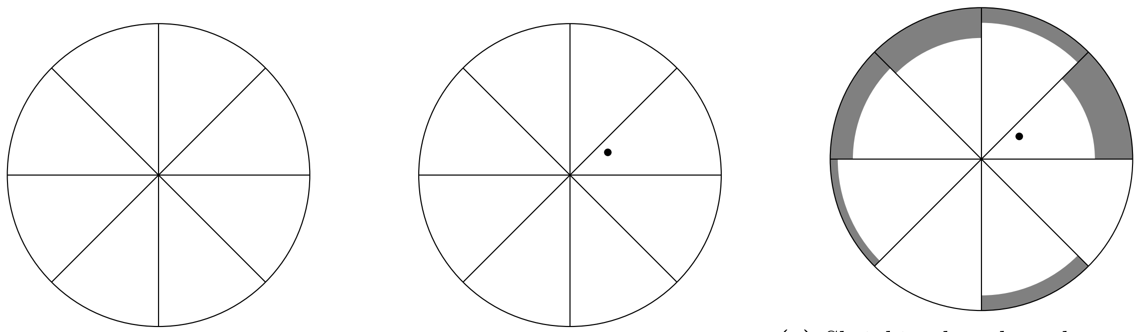
(a) Initial dartboard **(b)** Dart thrown **(c)** Shrinking based on the costs of the experts

Figure 1. Probability distribution as a dartboard

the costs of SD are only handling costs. These costs are caused by the costs of expert e^t for handling the request r^t. They consist of the transition cost for changing the state and the processing cost for answering the request. Thus, the costs of SD are $c^t_{\texttt{SD}} = c^t_{\texttt{SD},\text{hand}} = d(S^{t-1}_{e^t}, S^t_{e^t}) + p(r^t, S^t_{e^t}, A^t_{e^t}) = c^t_{e^t}$. If SD changes the expert, i. e., $e^t \neq e^{t-1}$, SD has both switching costs and handling costs. Since we consider a task system and a set of experts with switching cost at most B, the switching costs are $c^t_{\texttt{SD},\text{switch}} \leq B$. On the other hand, the handling costs are only the processing costs of the chosen expert e^t since the current state is already $S^t_{e^t}$. So, the handling costs of SD are $c^t_{\texttt{SD},\text{hand}} = p(r^t, S^t_{e^t}, A^t_{e^t}) \leq c^t_{e^t}$. Therefore, in this case, both costs together are the costs of SD in time step t, i. e., $c^t_{\texttt{SD}} = c^t_{\texttt{SD},\text{switch}} + c^t_{\texttt{SD},\text{hand}} \leq B + c^t_{e^t}$.

We want to estimate the regret of the SD algorithm. To this end, we first compute the probabilities for choosing the experts.

Lemma 1. *The probability for choosing expert i in time step t is $\mathbf{Pr}[e^t = i] = q^t_i$, for all $i \in \mathcal{E}$ and $t \in \{1, \dots, T\}$.*

Proof. Consider an arbitrary expert $i \in \mathcal{E}$. We show the lemma by an induction on $t \in \{1, \dots, T\}$. For $t = 1$, the statement in the lemma follows immediately from the description of the algorithm (line 1 and 2 of Algorithm 1). Now let $t \geq 2$. Algorithm SD chooses expert i in time step t either because it was selected already in time step $t-1$ and the corresponding dart is still in the allowed area (i.e., the expert is chosen by line 9) or because a new dart is thrown and this dart hits i's sector (i.e., the expert is chosen by line 11). Hence,

$$\begin{aligned}
\mathbf{Pr}\Big[e^t = i\Big] &= \underbrace{\mathbf{Pr}\Big[e^{t-1} = i\Big] \cdot \frac{w^t_i}{w^{t-1}_i}}_{\text{no expert change}} + \underbrace{q^t_i \cdot \sum_{j=1}^{N} \mathbf{Pr}\Big[e^{t-1} = j\Big] \cdot \left(1 - \frac{w^t_j}{w^{t-1}_j}\right)}_{\text{change to expert } i} \\
&= q^{t-1}_i \cdot \frac{w^t_i}{w^{t-1}_i} + q^t_i \cdot \sum_{j=1}^{N} q^{t-1}_j \cdot \left(1 - \frac{w^t_j}{w^{t-1}_j}\right) \\
&= \frac{w^{t-1}_i}{W^{t-1}} \cdot \frac{w^t_i}{w^{t-1}_i} + \frac{w^t_i}{W^t} \cdot \sum_{j=1}^{N} \frac{w^{t-1}_j}{W^{t-1}} \cdot \frac{w^{t-1}_j - w^t_j}{w^{t-1}_j} \\
&= \frac{w^t_i}{W^{t-1}} + \frac{w^t_i}{W^t} \cdot \frac{W^{t-1} - W^t}{W^{t-1}}
\end{aligned}$$

$$= w_i^t \cdot \frac{W^t + W^{t-1} - W^t}{W^t \cdot W^{t-1}} \quad = \quad \frac{w_i^t}{W^t} \quad = \quad q_i^t \ ,$$

where we used the induction hypothesis to go from the first line to the second. □

Note that Lemma 1 in particular shows that the probability for choosing an expert i is the same for `SD` and `RWM`.

Let D denote the number of expert changes during the execution of `SD`. The following lemma bounds the expected value of D in terms of the total cost of the best expert.

Lemma 2. *For $\eta \in [0, 2/3]$, $\mathbf{E}[D] \leq 2\eta C^T_{\texttt{best}} + \ln N$.*

Proof. We bound the total weight at the end of the instance W^{T+1} by both the total cost of the best expert and the expected number of expert changes D. Note that `SD` never computes the value W^{T+1}, but we can compute it by ourselves using line 6 of Algorithm 1 after request T was answered. Since `SD` uses the same weights as `RWM`, we can reuse the lower bound for W^{T+1} given by the learning rate η and the total cost of the best expert as proven in [31], namely

$$W^{T+1} \ \geq \ (1-\eta)^{C^T_{\texttt{best}}} \ . \tag{1}$$

On the other hand, D is bounded from above by the number of times line 11 of Algorithm 1 is applied which corresponds to the number of darts that are thrown because the sector of the chosen expert shrinks. The probability for throwing a new dart in time step $t \geq 2$ is

$$\alpha^t \quad = \quad \sum_{j=1}^{N} \mathbf{Pr}\Big[e^{t-1} = j\Big] \cdot \left(1 - \frac{w_j^t}{w_j^{t-1}}\right) \quad = \quad \frac{W^{t-1} - W^t}{W^{t-1}} \ , \tag{2}$$

where the latter equation follows from the calculation in the proof of Lemma 1. Thus,

$$\alpha^t W^{t-1} = W^{t-1} - W^t \ ,$$

which is equivalent to

$$W^t = (1 - \alpha^t) W^{t-1} \ . \tag{3}$$

Next, we repeatedly use (3) to bound W^{T+1} from above.

$$\begin{aligned} W^{T+1} &= W^T(1-\alpha^{T+1}) \ = \ W^{T-1}(1-\alpha^T)(1-\alpha^{T+1}) \\ &= \ldots \ = \ W^1 \prod_{t=1}^{T}(1-\alpha^{t+1}) \ = \ N \prod_{t=1}^{T}(1-\alpha^{t+1}) \ , \end{aligned} \tag{4}$$

where the last equality follows from $W^1 = N$ (line 1 of Algorithm 1). By combination of (1) with (4), we get

$$(1-\eta)^{C^T_{\texttt{best}}} \ \leq \ N \prod_{t=1}^{T}(1-\alpha^{t+1}) \ .$$

Taking logarithms, we have

$$C^T_{\texttt{best}} \ln(1-\eta) \leq (\ln N) + \sum_{t=1}^{T} \ln(1-\alpha^{t+1}) \leq (\ln N) - \sum_{t=1}^{T} \alpha^{t+1} \ ,$$

where the second inequality is true since $\ln(1-z) \leq -z$ holds for $z \in [0,1)$, and, according to (2),

$$\alpha^{t+1} \;=\; \frac{W^t - W^{t+1}}{W^t} \;=\; 1 - \frac{W^{t+1}}{W^t} \in [0,1) \;\; .$$

Using $\ln(1-z) \geq -2z$ for $z \in [0, 2/3]$, we get

$$-2\eta C^T_{\texttt{best}} \;\leq\; (\ln N) - \sum_{t=1}^{T} \alpha^{t+1} \;\; .$$

Thus, the expected number of expert changes $D = \sum_{t=1}^{T} \alpha^{t+1}$ is bounded from above by

$$D \;\leq\; 2\eta C^T_{\texttt{best}} + \ln N \;\; . \qquad \square$$

Now, we bound the expected total cost of SD depending on the learning rate η (Theorem 1). Afterwards, we will see how to tune η in such a way that it minimizes the regret of SD.

Theorem 1. *For $\eta \in [0, 2/3]$, the expected total cost of* SD *satisfies*

$$\mathbf{E}\Big[C^T_{\texttt{SD}}\Big] \;\leq\; (1 + \eta + 2\eta B) C^T_{\texttt{best}} + \frac{\ln N}{\eta} + B \ln N \;\; .$$

Proof. We claim that the expected total cost of SD is

$$\mathbf{E}\Big[C^T_{\texttt{SD}}\Big] \;=\; \mathbf{E}\Big[C^T_{\texttt{SD},\text{hand}}\Big] + \mathbf{E}\Big[C^T_{\texttt{SD},\text{switch}}\Big] \;\leq\; \sum_{t=1}^{T} c^t_{e^t} + DB \;\; .$$

In words, the expected total cost of SD is bounded from above by the sum over the handling cost of the chosen experts plus the number of expert changes times the upper bound on the switching costs. To see this, consider the total cost in a time period beginning with a time step in which a new expert is chosen and ending with the last time step before the next expert is chosen or the request sequence ends. The total cost of SD in this period can be split into two contributions.

(1) Handling costs $c^t_{\texttt{SD},\text{hand}}$: These costs are caused by answering the requests in this period by both SD and the expert. They consist of the transition cost for changing the state (if SD did not change the expert) and the processing cost for answering the request (line 3, 4, 13 and 14 of Algorithm 1).

(2) Switching costs $c^t_{\texttt{SD},\text{switch}}$: These costs are caused by switching the state at the beginning of the period (line 13 of Algorithm 1).

The handling cost in (1) in each time step t is bounded from above by $c^t_{e^t}$. Thus, it is bounded from above by $\sum_{t=1}^{T} c^t_{e^t}$ for the whole instance since both SD and the expert are in each time step in the same state. Since SD changes the expert only at the beginning of a period and each expert change costs at most B, the switching cost in (2) in a period is at most B. Note that, in the first period, the switching costs of (2) are zero because both SD and the experts start in the same state s_0. This gives the stated upper bound on $\mathbf{E}\Big[C^T_{\texttt{SD}}\Big]$ as the number of periods without counting the first period is D.

Next, we claim that

$$\mathbf{E}\left[\sum_{t=1}^{T} c^t_{e^t}\right] \;\leq\; (1+\eta) C^T_{\texttt{best}} + \frac{\ln N}{\eta} \;\; .$$

This follows from Lemma 1 showing that the probability that SD chooses expert i in time step t is equal to the probability that RWM chooses expert i in time step t. So, $\sum_{t=1}^{T} c_{e^t}^t$ describes the total handling cost of RWM assuming that the handling costs accounted for the online learning algorithm in time step t are $c_{e^t}^t$ (as in the standard setting of online learning without switching costs). Thus, for $\eta \in [0, 2/3]$, we can apply the following well-known bound (see, e. g., [9] for a proof[3]) on the expected total handling cost of RWM:

$$\mathbf{E}\Big[C^T_{\texttt{RWM},\text{hand}}\Big] \;\le\; (1+\eta)C^T_{\texttt{best}} + \frac{\ln N}{\eta} \;. \tag{5}$$

Together with the bound on the expected value of D in Lemma 2 this yields

$$\mathbf{E}\Big[C^T_{\texttt{SD}}\Big] \;\le\; (1+\eta)C^T_{\texttt{best}} + \frac{\ln N}{\eta} + 2\eta B C^T_{\texttt{best}} + B\ln N \;. \qquad \square$$

Thus, we can bound the expected total costs of SD from above using the learning rate, the upper bound on the switching costs, the total costs of the best expert, and the number of experts. Next, we tune the learning rate η. Let C^* be an upper bound for the total cost of the best expert, i. e., $C^T_{\texttt{best}} \le C^*$. Using C^* we get the following upper bound for the expected total cost of SD.

Corollary 3. *For $B \ge 1$ and $C^* \ge \max\{\frac{9\ln N}{4B}, B\ln N\}$, setting $\eta = \sqrt{(\ln N)/BC^*}$ yields*

$$\mathbf{E}\Big[C^T_{\texttt{SD}}\Big] \le C^T_{\texttt{best}} + 5\sqrt{BC^*\ln N} \;.$$

Proof. First, note that, for $C^* \ge \frac{9\ln N}{4B}$, $\eta = \sqrt{(\ln N)/BC^*} \in [0, 2/3]$. So using Theorem 1 and $C^T_{\texttt{best}} \le C^*$ we can bound the expected total cost of SD from above by

$$\begin{aligned}\mathbf{E}\Big[C^T_{\texttt{SD}}\Big] &\le C^T_{\texttt{best}} \;+\; \underbrace{\frac{\sqrt{\ln N}}{\sqrt{BC^*}}C^T_{\texttt{best}}}_{\le\sqrt{BC^*\ln N}} \;+\; \underbrace{\frac{2\sqrt{B\ln N}}{\sqrt{C^*}}C^T_{\texttt{best}}}_{\le 2\sqrt{BC^*\ln N}} \;+\; \underbrace{\frac{\sqrt{BC^*}\ln N}{\sqrt{\ln N}}}_{\le\sqrt{BC^*\ln N}} \;+\; \underbrace{B\ln N}_{\le\sqrt{BC^*\ln N}} \\ &\le C^T_{\texttt{best}} \;+\; 5\sqrt{BC^*\ln N} \;. \end{aligned} \qquad \square$$

Note that, in Corollary 3, the value C^* can also be replaced by the length T of the instance.

The regret analysis of the online learning algorithm SD is based on the assumption that the online learning algorithm knows in advance an upper bound C^* on the total cost of the best expert for the considered instance δ, i. e., $C^T_{\texttt{best}} \le C^*$ or at least the length T of δ. If neither of them is known by the algorithm in advance, one cannot tune the learning rate η in such a way that the online learning algorithm guarantees the no-regret property.

To overcome this drawback, we consider the well-known guess-and-double approach [9] for the online learning algorithm SD for the problem of regret minimization for task systems and sets of experts with switching cost. This approach works as follows: Given an arbitrary online learning algorithm A, we initially set C^* to some suitable value. While the total cost of at least one expert in $\mathcal{E}$ is below C^*, we use A with the learning rate η tuned by C^*. As soon as all experts have at least cost C^*, we double the value of C^* and restart A with the new value for η tuned by C^*. It is well-known [9] that, in the standard setting of online learning algorithms, the guess-and-double approach used with RWM guarantees an expected regret of the same order as the expected regret

[3] There, the result is only claimed and proven for $\eta \in [0, 1/2]$, but it can be easily seen that the only estimation used there which restricts the value of η, namely $-\ln(1-\eta) \le \eta + \eta^2$, is valid also for $\eta \in [0, 2/3]$.

Algorithm 2 (Guess and Double (GAD))

```
1: $M = \max\{\frac{9 \ln N}{4B}, B \ln N\}$
2: for $k = 0, 1, 2, \ldots$ do
3:     $C^*_k = 2^k \cdot M$
4:     use online learning algorithm SD with $\eta = \sqrt{\frac{\ln N}{BC^*_k}}$
5:         until the first time step $t$ where $C^t_i \geq C^*_k$, for all $i \in \mathcal{E}$
6: end for
```

of RWM. We will see that this approach used with SD can also guarantee, for task systems and sets of experts with switching cost, that its expected regret is, up to constant factors, the same as the expected regret of SD.

The online learning algorithm *Guess and Double* (GAD) is given in Algorithm 2. GAD works in rounds. In round k, it uses the online learning algorithm SD with C^*_k (line 4) until the accumulated cost of each expert up to the current time step t is at least C^*_k (line 5). C^*_k is set by the following process. In the first round $k = 0$, we set $C^*_0 = \max\{\frac{9 \ln N}{4B}, B \ln N\}$ (lines 1 and 3). In round $k > 0$, GAD doubles the value of C^*_k (line 3). Note that, by this process, we can bound the expected total cost of SD in each round from above by Corollary 3.

To prove that GAD is a no-regret learning algorithm, we first compute an upper bound K on the number of rounds GAD will run. This upper bound depends on the total cost of the best expert and on M.

Lemma 3. *For $C^T_{\texttt{best}} \geq \max\{\frac{9 \ln N}{4B}, B \ln N\}$, at the end of an instance δ of length T,* GAD *is at most in round*

$$K \;=\; \lfloor \log_2 C^T_{\texttt{best}} - \log_2 M + 1 \rfloor \;\; .$$

Proof. Since GAD goes from round $k-1$ to round k if the total costs of each expert is at least $C^*_{k-1} = 2^{k-1} \cdot M$, we can bound the round K of GAD at the end of an instance δ of length T from above by the largest k for that C^*_{k-1} is still smaller than the total costs of the best expert $C^T_{\texttt{best}}$. So, for $k \in \{1, \ldots, K\}$,

$$C^*_{k-1} \;=\; 2^{k-1} \cdot M \;\leq\; C^T_{\texttt{best}} \;\; ,$$

since $C^T_{\texttt{best}} \geq M$. Taking logarithms, we have

$$(k-1) + \log_2 M \leq \log_2 C^T_{\texttt{best}}$$

and thus

$$k \leq \log_2 C^T_{\texttt{best}} - \log_2 M + 1 \;\; . \qquad \square$$

Next, we bound the expected total cost of GAD for an arbitrary round $k \in \{1, \ldots, K-1\}$ from above. We denote by $\mathbf{E}[C_{\texttt{GAD},k}]$ the expected total cost of GAD in round k and by $C_{\texttt{best},k}$ the total cost of the best expert in round k.

Lemma 4. *For $B \geq 1$, the expected total cost of* GAD *in round $k \in \{1, \ldots, K-1\}$ satisfies*

$$\mathbf{E}[C_{\texttt{GAD},k}] \;\leq\; 2^{k-1} \cdot M + 5\sqrt{2^k M B \ln N} \;\; .$$

Proof. At the beginning of round $k \in \{1, \ldots, K-1\}$, the accumulated cost of each expert is, according to line 5 of Algorithm 2, at least $C^*_{k-1} = 2^{k-1} \cdot M$. On the other hand, at the end of round $k \in \{1, \ldots, K-1\}$, the accumulated cost of each expert is at least $C^*_k = 2^k \cdot M$. So, the cost of the best expert in this round is at most $C_{\texttt{best},k} \leq C^*_k - C^*_{k-1} = 2^{k-1} \cdot M$.

Since $B \geq 1$ and $C^*_k = 2^k \cdot M \geq M = \max\{\frac{9 \ln N}{4B}, B \ln N\}$, we can use Corollary 3 to bound the expected total cost of `GAD` in this round from above using the expected total cost of `SD` in this round by

$$\begin{aligned}\mathbf{E}[C_{\texttt{GAD},k}] &\leq C_{\texttt{best},k} + 5\sqrt{C^*_k B \ln N} \\ &\leq 2^{k-1} \cdot M + 5\sqrt{2^k M B \ln N} \ .\end{aligned}$$ □

Now, we combine the results from Lemma 3 and Lemma 4 to bound the expected total cost of `GAD` for $B \geq 1$.

Theorem 2. *For $B \geq 1$ and $C^T_{\texttt{best}} \geq \max\{\frac{9 \ln N}{4B}, B \ln N, 256B^2\}$, the expected total cost of* `GAD` *satisfies*

$$\mathbf{E}\left[C^T_{\texttt{GAD}}\right] \ \leq \ C^T_{\texttt{best}} + 29\sqrt{BC^T_{\texttt{best}} \ln N} \ .$$

Proof. Since $C^T_{\texttt{best}} \geq \max\{\frac{9 \ln N}{4B}, B \ln N\}$, we can use the upper bound on the number of rounds from Lemma 3. So, $K = \lfloor \log_2 C^T_{\texttt{best}} - \log_2 M + 1 \rfloor$. Note that $K \geq 1$.

Furthermore, since $B \geq 1$, for each round $k \in \{1, \ldots, K-1\}$, we can use the upper bound on the expected total cost of `GAD` in round k given in Lemma 4. So,

$$\mathbf{E}[C_{\texttt{GAD},k}] \ \leq \ 2^{k-1} \cdot M + 5\sqrt{2^k M B \ln N} \ ,$$

for all $k \in \{1, \ldots, K-1\}$. Note that we can prove similarly to Lemma 4 the bounds

$$\mathbf{E}[C_{\texttt{GAD},0}] \ \leq \quad M \quad + 5\sqrt{M B \ln N}$$

for round $k = 0$ and

$$\mathbf{E}[C_{\texttt{GAD},K}] \ \leq \ C_{\texttt{best},K} \ + 5\sqrt{2^K M B \ln N}$$

for round $k = K$. Let $C^T_{\texttt{best},K}$ be the cost of the best expert for the whole instance in round $k = K$. Since the best expert in round K has cost $C_{\texttt{best},K}$, we have $C_{\texttt{best},K} \leq C^T_{\texttt{best},K}$. Furthermore, note that $C^T_{\texttt{best},K} + 2^{K-1} \cdot M \leq C^T_{\texttt{best}}$ since, in round $k = K$, the cost of each expert is at least $2^{K-1} \cdot M$. So,

$$\mathbf{E}[C_{\texttt{GAD},K}] \ \leq \ \left(C^T_{\texttt{best}} - 2^{K-1} \cdot M\right) + 5\sqrt{2^K M B \ln N} \ .$$

At the beginning of each round, we restart the online learning algorithm `SD`. So, for each round except the first one, `GAD` has switching costs at most B that are not considered in the regret analysis of `SD`. Since `GAD` uses $K+1$ rounds, the switching costs are at most BK.

So, the expected total cost of `GAD` is

$$\mathbf{E}\left[C^T_{\texttt{GAD}}\right] \leq \underbrace{\mathbf{E}[C_{\texttt{GAD},0}]}_{\text{round } 0} + \underbrace{\sum_{k=1}^{K-1} \mathbf{E}[C_{\texttt{GAD},k}]}_{\text{round } 1 \text{ to } K-1} + \underbrace{\mathbf{E}[C_{\texttt{GAD},K}]}_{\text{round } K} + \underbrace{BK}_{\text{switching costs}}$$

$$
\begin{aligned}
&\leq \left[M + 5\sqrt{MB\ln N}\right] + \left[\sum_{k=1}^{K-1}\left(2^{k-1}\cdot M + 5\sqrt{2^k MB\ln N}\right)\right] \\
&\quad + \left[\left(C^T_{\texttt{best}} - 2^{K-1}\cdot M\right) + 5\sqrt{2^K MB\ln N}\right] + \left[BK\right] \\
&= C^T_{\texttt{best}} + M + BK + \sum_{k=1}^{K-1} 2^{k-1}\cdot M - 2^{K-1}\cdot M + \sum_{k=0}^{K} 5\sqrt{2^k MB\ln N} \\
&= C^T_{\texttt{best}} + M + BK + \sum_{k=0}^{K-2} 2^{k}\cdot M - 2^{K-1}\cdot M + 5\sqrt{MB\ln N}\cdot\sum_{k=0}^{K} 2^{k/2} \\
&\leq C^T_{\texttt{best}} + M + BK + 2^{K-1}\cdot M - 2^{K-1}\cdot M + 5\sqrt{MB\ln N}\cdot\frac{2^{(K+1)/2}-1}{\sqrt{2}-1} \\
&\leq C^T_{\texttt{best}} + M + BK + \frac{5}{\sqrt{2}-1}\sqrt{MB\ln N}\cdot 2^{(K+1)/2} \\
&\leq C^T_{\texttt{best}} + M + B\cdot(\log_2 C^T_{\texttt{best}} + 1) + \frac{5}{\sqrt{2}-1}\sqrt{MB\ln N}\cdot 2^{(\log_2 C^T_{\texttt{best}} - \log_2 M + 2)/2} \\
&\leq C^T_{\texttt{best}} + M + B + B\log_2 C^T_{\texttt{best}} + \frac{10}{\sqrt{2}-1}\sqrt{BC^T_{\texttt{best}}\ln N}\ ,
\end{aligned}
$$

where we used in the third inequality the sum of the first terms of a geometric series and in the fifth inequality that $K = \lfloor \log_2 C^T_{\texttt{best}} - \log_2 M + 1\rfloor$.

For the rest of the proof, we assume that $N \geq 2$. Note that, for $N = 1$, $\mathbf{E}\left[C^T_{\texttt{GAD}}\right] = C^T_{\texttt{best}}$. Since $B \geq 1$ and $M = \max\{\frac{9\ln N}{4B}, B\ln N\}$, we have $M \leq \frac{9}{4}B\ln N$. Furthermore, since $C^T_{\texttt{best}} \geq B\ln N$ and $1 \leq \frac{3}{2}\ln N$, we have $B \leq \frac{3}{2}\sqrt{BC^T_{\texttt{best}}\ln N}$ and $M \leq \frac{9}{4}\sqrt{BC^T_{\texttt{best}}\ln N}$.

Finally, we show that $B\log_2 C^T_{\texttt{best}} \leq \sqrt{BC^T_{\texttt{best}}\ln N}$ for $C^T_{\texttt{best}} \geq 256B^2$. Since we assume that $N \geq 2$, we have $\frac{4}{5} \leq \sqrt{\ln N}$. So, we show $\sqrt{B}\log_2 C^T_{\texttt{best}} \leq \frac{4}{5}\sqrt{C^T_{\texttt{best}}}$ for $C^T_{\texttt{best}} \geq 256B^2$. First consider the case $C^T_{\texttt{best}} = 256B^2$. Here, we have

$$
\sqrt{B}\log_2 256B^2 \ = \ 8\sqrt{B} + 2\sqrt{B}\log_2 B \ \leq \ 10B \ \leq \ \frac{4}{5}\cdot 16B \ = \ \frac{4}{5}\sqrt{256B^2}
$$

since $\log_2 B \leq \sqrt{B}$. Since B is a constant, the inequality holds for $C^T_{\texttt{best}} = 256B^2$, and $\log_2 C^T_{\texttt{best}} \leq \sqrt{C^T_{\texttt{best}}}$, the inequality holds for all $C^T_{\texttt{best}} \geq 256B^2$. So,

$$
\begin{aligned}
\mathbf{E}\left[C^T_{\texttt{GAD}}\right] &\leq C^T_{\texttt{best}} + \left(\frac{9}{4} + \frac{3}{2} + 1 + \frac{10}{\sqrt{2}-1}\right)\sqrt{BC^T_{\texttt{best}}\ln N} \\
&\leq C^T_{\texttt{best}} + 29\sqrt{BC^T_{\texttt{best}}\ln N}\ . \qquad \square
\end{aligned}
$$

It directly follows that the expected regret of GAD on an instance δ of length T is $R^T_{\texttt{GAD}}(\delta) = 29\sqrt{BC^T_{\texttt{best}}\ln N}$ if $B \geq 1$. Since $C^T_{\texttt{best}} \leq T$, we can summarize our results in the following theorem.

Theorem 3. *For $B \geq 1$ and $C^T_{\texttt{best}} \geq \max\{\frac{9\ln N}{4B}, B\ln N, 256B^2\}$, the expected total cost of* GAD *satisfies*

$$
\mathbf{E}\left[C^T_{\texttt{GAD}}\right] \ \leq \ C^T_{\texttt{best}} + 29\sqrt{BC^T_{\texttt{best}}\ln N}\ . \qquad \square
$$

We are now ready to show a method that converts an online algorithm with advice for a task system into a randomized online algorithm. We interpret an online algorithm with advice as a set of deterministic online algorithms where, for each instance, the oracle chooses one of these algorithms. We construct the randomized online algorithm based on the online learning algorithm *Guess and Double* (`GAD`) given in Algorithm 2. The set of experts used by `GAD` corresponds to the set of online algorithms that the online algorithm with advice uses. Since `GAD` has the no-regret property, if the set of deterministic strategies used by the online algorithm with advice satisfies both Assumption 1 (normalized expert costs) and Assumption 2 (bounded switching costs), the expected total cost of `GAD` is almost the same as the (expected) total cost of the best online algorithm used by the online algorithm with advice for all instances. So, if the online algorithm with advice reaches some competitive ratio for all instances, then the constructed randomized online algorithm has almost the same competitive ratio.

More formally, let $\mathcal{T} = (\mathcal{R}, \mathcal{A}, \mathcal{S}, s_0, d, p)$ be a task system. An online algorithm with advice `A` for $\mathcal{T}$ with advice complexity $\rho(T)$ reads, for each instance δ of length T, at most $\rho(T)$ bits from an advice tape. The bits on the advice tape are computed by an oracle `O` for $\mathcal{T}$ that knows the whole instance δ in advance. Using δ, it computes an arbitrary function of δ. Note that we can represent $2^{\rho(T)}$ different numbers with a bit string of length $\rho(T)$. Using this idea, we can interpret the usage of the $\rho(T)$ advice bits by the following process: Assume that `A` knows the length T of the instance δ. Then, in the first time step, `A` asks the advice oracle `O` for the first $\rho(T)$ advice bits. `A` uses these advice bits to select one of at most $2^{\rho(T)}$ deterministic online algorithms. Afterwards, it uses the selected online algorithm for the whole instance without usage of the advice oracle `O`.

We first consider the special case of online algorithms with advice that have bounded advice complexity. We say that an online algorithm with advice has bounded advice complexity $\rho^* \in \mathbb{N}$ if the online algorithm uses, for each instance, at most ρ^* advice bits. For many online optimization problems, online algorithms with advice are known that use only a bounded number of advice bits, e. g., the ski rental problem [25], paging [14], or the list accessing problem [19].

If an online algorithm with advice `A` has bounded advice complexity ρ^*, we can interpret it as a fixed set of at most 2^{ρ^*} deterministic or randomized online algorithms used by `A`. We call this set the *associated set of online algorithms of* `A` and denote it by $\mathfrak{A}$.

Definition 12 (Associated Set of Online Algorithms). *Let $\mathcal{T} = (\mathcal{R}, \mathcal{A}, \mathcal{S}, s_0, d, p)$ be a task system and* `A` *be an online algorithm with advice for $\mathcal{T}$ that has bounded advice complexity ρ^* and uses an advice oracle* `O`*. The* associated set of online algorithms *used by* `A` *is*

$$\mathfrak{A} = \left\{ \mathtt{A}[x_i] \;\middle|\; i \in \{0, \dots, 2^{\rho^*} - 1\},\ x_i = bin_{\rho^*}(i),\ \exists \textit{ instance } \delta \textit{ s.t. the output of } \mathtt{O} \textit{ is } x_i \right\} ,$$

where $bin_{\rho^*}(i)$ is the bit string of length ρ^* corresponding to the binary representation of the integer i and $\mathtt{A}[x_i]$ is the online algorithm used by `A` *if the advice is x_i.*

In the next theorem, we show that we can convert an online algorithm with bounded advice complexity into a randomized online algorithm using the online learning algorithm *Guess and Double* (`GAD`) given in Algorithm 2. The idea of the proof is that if, for an online algorithm with advice, the associated set of online algorithms $\mathfrak{A}$ satisfies both Assumption 1 (normalized expert costs) and Assumption 2 (bounded switching costs), we can use $\mathfrak{A}$ as the set of experts of the online learning algorithm `GAD`. Since `GAD` has the no-regret property, its expected total cost is almost the same as the (expected) total cost of the best online algorithm in the set of experts. Thus, if the online algorithm with advice is γ-competitive, then, for each instance, there is a γ-competitive online algorithm in the set of experts and `GAD` has almost the same competitive ratio.

Theorem 4. *Let $\mathcal{T} = (\mathcal{R}, \mathcal{A}, \mathcal{S}, s_0, d, p)$ be a task system,* $\mathtt{A}$ *be a γ-competitive online algorithm with bounded advice complexity ρ^* for $\mathcal{T}$, and $\mathfrak{A}$ be the associated set of online algorithms used by* $\mathtt{A}$.

If $\mathfrak{A}$ satisfies both Assumption 1 (normalized expert costs) and Assumption 2 (bounded switching costs), then there is a randomized online algorithm $\mathtt{A}'$ with expected total cost

$$\mathbf{E}\Big[C^T_{\mathtt{A}'}(\delta)\Big] \;\leq\; \left(1 + \frac{25\sqrt{B\rho^*}}{\sqrt{\gamma C^T_{\mathtt{opt}}(\delta)}}\right)\gamma \cdot C^T_{\mathtt{opt}}(\delta) \;+\; \beta \;, \tag{6}$$

for some constant β.

Proof. For the proof, we assume without loss of generality that $B \geq 1$. If $B \in [0, 1]$, we round B up to 1. Since $\mathfrak{A}$ is a set of online algorithms for $\mathcal{T}$ that satisfies both Assumption 1 (normalized expert costs) Assumption 2 (bounded switching costs), we can use the online learning algorithm $\mathtt{GAD}$ with set of experts $\mathcal{E} = \mathfrak{A}$. Let N be the size of $\mathfrak{A}$. We have $N \leq 2^{\rho^*}$. Furthermore, assume that

$$C^T_{\mathtt{A}}(\delta) \;=\; C^T_{\mathtt{best}}(\delta) \;\geq\; \max\left\{\frac{9\ln N}{4B},\; B\ln N,\; 256B^2\right\} \;.$$

Since $B \geq 1$, we can use Theorem 3 to bound the expected total cost of $\mathtt{GAD}$ from above by

$$\begin{aligned}
\mathbf{E}\Big[C^T_{\mathtt{GAD}}(\delta)\Big] &\leq C^T_{\mathtt{best}}(\delta) + 29\sqrt{BC^T_{\mathtt{best}}(\delta)\ln N}\\
&= C^T_{\mathtt{A}}(\delta) + 29\sqrt{BC^T_{\mathtt{A}}(\delta)\ln N}\\
&\leq \gamma \cdot C^T_{\mathtt{opt}}(\delta) + \alpha + 29\sqrt{B(\gamma C^T_{\mathtt{opt}}(\delta) + \alpha)\ln 2^{\rho^*}}\\
&\leq \gamma \cdot C^T_{\mathtt{opt}}(\delta) + \alpha + 25\sqrt{B\gamma C^T_{\mathtt{opt}}(\delta)\rho^* + B\alpha\rho^*}\\
&\leq \gamma \cdot C^T_{\mathtt{opt}}(\delta) + \alpha + 25\sqrt{B\gamma C^T_{\mathtt{opt}}(\delta)\rho^*} + 25\sqrt{B\alpha\rho^*}\\
&= \left(1 + \frac{25\sqrt{B\rho^*}}{\sqrt{\gamma C^T_{\mathtt{opt}}(\delta)}}\right)\gamma \cdot C^T_{\mathtt{opt}}(\delta) + \alpha' \;,
\end{aligned}$$

for some constant α'. In order to deal with instances where the (expected) total cost $C^T_{\mathtt{A}}(\delta) = C^T_{\mathtt{best}}(\delta) < \max\{\frac{9\ln N}{4B}, B\ln N, 256B^2\}$, we increase the constant α' to an appropriate constant β to bound the expected total cost also in this case. Note that β is indeed a constant since $\mathtt{GAD}$ has only costs if the chosen expert has costs and both B and N are constants. $\mathtt{GAD}$ is a randomized online learning algorithm. Consequently, the constructed online algorithm $\mathtt{A}'$ is randomized, too. ☐

Note that the term in parentheses in (6) contains the constants B, ρ^*, and γ. Thus, given a γ-competitive online algorithm $\mathtt{A}$ with bounded advice complexity, for each $\varepsilon > 0$, we can find a constant β such that the randomized online algorithm $\mathtt{A}'$ has competitive ratio $(1+\varepsilon)\gamma$. Furthermore, for $C^T_{\mathtt{opt}}(\delta) \to \infty$, the randomized online algorithm $\mathtt{A}'$ has almost the same competitive ratio as $\mathtt{A}$.

4 Two Applications

In this section, we consider two applications of Theorem 4 for paging and list accessing.

4.1 Paging

Modern computer architectures use hierarchically structured memory systems. One of the challenges in these systems is to determine which data units to store in which memory without knowledge of the future access patterns. In the paging problem, we consider a simplified system with two types of memory: a slow memory that stores a fixed set $P = \{p_1, \dots, p_n\}$ of n fixed-size data units (pages) and a fast memory that stores, in each time step, at most $k < n$ pages. Given a request for a page p_i, the system checks whether p_i is in the fast memory. If p_i is not in the fast memory, the system has to replace one of the pages in the fast memory by p_i. Each copy operation (page fault) has cost one. The goal of a page replacement algorithm is to replace the pages in such a way that it minimizes the total number of page faults. Next, we show how we can model the paging problem as a task system.

Definition 13 (Paging Problem). *We can model the paging problem as a task system* $\mathcal{T}_{paging} = (\mathcal{R}, \mathcal{A}, \mathcal{S}, \{\}, d, p)$ *with* $\mathcal{R} = \{p_1, \dots, p_n\}$, $\mathcal{A} = \{a\}$, *and* $\mathcal{S} = \{\ \{p_{x_1}, \dots, p_{x_l}\}\ \mid\ 0 \leq l \leq k,\ 1 \leq x_i < x_j \leq n \text{ with } i \neq j\}$. *The transition cost function is*

$$d(s, s') = |\{\ p_i \in s'\ \mid\ p_i \notin s\ \}|$$

and the processing cost function is

$$p(p_i, s, a) = \begin{cases} 0 & \text{if } p_i \in s \\ \infty & \text{else} \end{cases} .$$

As already mentioned, the transition cost of each online algorithm for the paging problem in each time step is at most k. Hence, the switching cost B of each set of online algorithms is also at most k. A general introduction to page replacement algorithms and known results on the competitiveness of these algorithms is given by Borodin and El-Yaniv [16].

The following theorem of Böckenhauer et al. [14] introduces a $(3 \log_2 k)$-competitive online algorithm with bounded advice complexity $\log_2 k$ for the paging problem (see also Komm and Královič [38], who observed that the analysis of this algorithm actually allows for a barely random algorithm).

Theorem 5 (Böckenhauer et al. [14]). *Let k be the size of the fast memory and $c < k$ be a power of 2. There is an online algorithm with advice complexity $\log_2 c$ for the paging problem with competitive ratio at most*

$$3 \log_2 c + \frac{2(k+1)}{c} + 1\ .$$

Therefore, if we set c to the largest number $c < k$ that is a power of 2, the online algorithm with advice of Theorem 5 has bounded advice complexity $\log_2 k$ and its competitive ratio is at most $3 \log_2 k + 9$ since $c \geq k/2$.

The associated set of online algorithms of the online algorithm with advice of Theorem 5 consists of c deterministic *marking algorithms*. A marking algorithm maintains a mark for each page. Initially, all pages are unmarked. If a page is requested, the page is marked and, if it is currently not in the fast memory, it is copied to it. If the fast memory is full and there is at least one unmarked page in it, then one of the unmarked pages in the fast memory is replace by the requested page. If the fast memory is full, and all pages in it are marked, then all pages are unmarked and one of the pages in the fast memory is replace by the requested page.

Marking algorithms are *demand paging algorithms*, i. e., they replace a page in the fast memory only if a page is requested that is currently not in it and there are already k pages in the fast memory. Thus, these online algorithms have, in each time step, either cost zero (if the requested page is in the fast memory) or cost one (if the requested page is not in the fast memory).

Next, we use the method studied in Section 3 to show that the online algorithm with advice of Theorem 5 can be converted to a randomized online algorithm with almost the same competitive ratio.

Theorem 6. *Let k be the size of the fast memory. For each $\varepsilon > 0$, there is a randomized online algorithm for the paging problem with competitive ratio at most $(1+\varepsilon)\cdot(3\log_2 k + 9)$.*

Proof. The associated set of online algorithms $\mathfrak{A}$ of the online algorithm with advice of Theorem 5 satisfies Assumption 1 (normalized expert costs) since all online algorithms in $\mathfrak{A}$ are marking algorithms and, thus, have either cost zero or one in each time step. Furthermore, $\mathfrak{A}$ satisfies Assumption 2 (bounded switching costs) with $B = k$ since the switching cost of each set of online algorithms is at most k for the paging problem. So, we can apply Theorem 4 to the $(3\log_2 k + 9)$-competitive online algorithm of Theorem 5 that has bounded advice complexity $\log_2 k$ to construct a randomized online algorithm with competitive ratio at most

$$\left(1 + \frac{25\sqrt{k\log_2 k}}{\sqrt{(3\log_2 k + 9)C^T_{\mathtt{opt}}(\delta)}}\right)\cdot(3\log_2 k + 9)\ .$$

Note that, for each $\varepsilon > 0$, there is an additive constant β such that the constructed randomized online algorithm has indeed a competitive ratio of at most $(1+\varepsilon)\cdot(3\log_2 k + 9)$. □

4.2 List Accessing

The storage of data objects can be done in many different ways in modern computer architectures. One way is the usage of a linked list. A linked list contains a sequence of data objects. Given an initial list, an instance of the list accessing problem consists of a sequence of objects accessed over time. The cost for accessing an object is proportional to its distance from the head of the list. We assume here that accessing the object at position i costs i. An algorithm that manages the list can reorder it at any time. There are two types of reorderings: free transpositions and paid transpositions. When the object at position i is accessed, it can be moved anywhere ahead of the current position for cost 0 (free transposition). Furthermore, the algorithm can swap any two items in the list for cost 1 (paid transposition). Next, we show how we can model the list accessing problem as a task system.

Definition 14 (List Accessing Problem). *We can model the list accessing problem with initial list $(o_1,\ldots,o_k)$ as a task system $\mathcal{T}_{list} = (\mathcal{R},\mathcal{A},\mathcal{S},((o_1,\ldots,o_k),0),d,p)$ with set of requests $\mathcal{R} = \{o_1,\ldots,o_k\}$, set of answers $\mathcal{A} = \{a\}$, and set of states $\mathcal{S} = \{\ ((o_{x_1},\ldots,o_{x_k}),l)\ \mid\ 1 \le x_i, l \le k, x_i \neq x_j \text{ for } i \neq j\}$. The transition cost function is*

$$d((s,l),(s',l')) = \begin{matrix}\textit{minimal number of transpositions to transform } s \textit{ to } s' \textit{ where}\\ \textit{all transpositions moving object } l \textit{ closer to the front are free}\end{matrix}$$

and the processing cost function is

$$p(o_i,((o_{x_1},\ldots,o_{x_k}),l),a) = \begin{cases} j \quad \textit{with } i = x_j & \textit{if } l = i\\ \infty & \textit{else}\end{cases}\ .$$

Note that free transpositions are done after answering the current request in the list accessing problem. Thus, by adding the current request to the state space, we use the transition cost function to do free transpositions in the next time step after the current request was answered.

Albers et al. [1] presented the 1.6-competitive online algorithm `COMB` for the list accessing problem. This algorithm chooses, according to a fixed probability distribution, initially one of two online algorithms for the list accessing problem (`BIT` or `TIMESTAMP`). Afterwards, it uses the chosen online algorithm for the whole instance. We can transform `COMB` to an online algorithm with advice by replacing the initial choice of the online algorithm using the probability distribution by one advice bit indicating which online algorithm has lower costs (see also Boyar et al. [19]).

Both online algorithms `BIT` and `TIMESTAMP` use only free transpositions. Furthermore, the costs of state changes are bounded from above by k^2. So, if we scale all costs by $1/k$, we can use the method studied in Section 3 to show that this online algorithm with advice can be converted to a randomized online algorithm with almost the same competitive ratio.

Theorem 7. *For each $\varepsilon > 0$, there is a randomized online algorithm for the list accessing problem with competitive ratio at most $1.6 \cdot (1 + \varepsilon)$.*

Proof. The associated set of online algorithms $\mathfrak{A}$ of the online algorithm with advice satisfies Assumption 1 (normalized expert costs) since all online algorithms in $\mathfrak{A}$ use only free transpositions and, thus, their scaled costs are between zero and one in each time step. Furthermore, $\mathfrak{A}$ satisfies Assumption 2 (bounded switching costs) with $B = k$ since by scaling all state changes can be done by costs of at most k. So, we can apply Theorem 4 to the 1.6-competitive online algorithm with advice that uses one advice bit to construct a randomized online algorithm with competitive ratio at most

$$1.6 \cdot \left(1 + \frac{25\sqrt{k}}{\sqrt{1.6 \cdot C^T_{\mathtt{opt}}(\delta)}}\right) .$$

Note that, for each $\varepsilon > 0$, there is an additive constant β such that the constructed randomized online algorithm has indeed a competitive ratio of at most $1.6 \cdot (1 + \varepsilon)$. □

5 Generalization to Unbounded Advice

In this section, we generalize the results of Section 3 to online algorithms with advice that have an unbounded, but not too large advice complexity.

The idea of our proof is roughly based on the guess-and-double approach that was already used to construct the online learning algorithm *Guess and Double* (`GAD`) in Section 3. Let `A` be an online algorithm with advice that has advice complexity $\rho(T)$. We reuse the idea of the construction of the randomized online algorithm `A`$'$ in Section 3 for the special case of online algorithms with bounded advice complexity. For this, we use the online learning algorithm `GAD` initially with the associated set of online algorithms corresponding to $\rho^* = \rho(1)$. While $\rho(t) = \rho(t-1)$, we continue using this instance of `GAD`. Whenever $\rho(t) > \rho(t-1)$ for some time step $t \in \{1, \ldots, T\}$, we restart `GAD` with the associated set of online algorithms corresponding to $\rho^* = \rho(t)$. By this, `GAD` is restarted at most $\rho(T)$ times. Furthermore, we have to assume that the cost of an optimal solution for an instance δ is not too low, i. e., $C^T_{\mathtt{opt}}(\delta) \in \omega(\rho(T)^3)$. In order to prove that the restarts of `GAD` do not affect the competitive ratio much, we first extend Definition 12 of the associated set $\mathfrak{A}$ of online algorithms used by an online algorithm with advice `A` as follows.

Definition 15 (Associated Set of Online Algorithms). *Let $\mathcal{T} = (\mathcal{R}, \mathcal{A}, \mathcal{S}, s_0, d, p)$ be a task system and* `A` *be an online algorithm with advice for $\mathcal{T}$ that has advice complexity $\rho(T)$ and uses an advice oracle* `O`*.*

The associated set of online algorithms of instances of length T *used by* `A` *is*

$$\mathfrak{A}(T) \;=\; \Big\{ \mathtt{A}[x_i] \;\mid\; i \in \{0, \ldots, 2^{\rho(T)} - 1\},\ x_i = bin_{\rho(T)}(i),\ \exists\ \delta \text{ s.t. the output of } \mathtt{O} \text{ is } x_i \Big\},$$

where $bin_{\rho(T)}(i)$ is the bit string of length $\rho(T)$ corresponding to the binary representation of the integer i and $\mathtt{A}[x_i]$ is the online algorithm used by `A` *if the advice is x_i.*

The next theorem shows that we can construct a randomized online algorithm using an online algorithm with advice complexity $\rho(T)$ whose associated set $\mathfrak{A}(T)$ of online algorithms of instances of length T satisfies both Assumption 1 (normalized expert costs) and Assumption 2 (bounded switching costs).

Theorem 8. *Let $\mathcal{T} = (\mathcal{R}, \mathcal{A}, \mathcal{S}, s_0, d, p)$ be a task system,* `A` *be a γ-competitive online algorithm with advice complexity $\rho(T)$ for $\mathcal{T}$, and $\mathfrak{A}(T)$ be the associated set of online algorithms of instances of length T used by* `A`*. If $\mathfrak{A}(T)$ satisfies both Assumption 1 (normalized expert costs) and Assumption 2 (bounded switching costs) for all $T \in \mathbb{N}$, then there is a randomized online algorithm $\mathtt{A}'$ with expected total cost*

$$\mathbf{E}\Big[C^T_{\mathtt{A}'}(\delta)\Big] \;\leq\; \left(1 + \frac{25\sqrt{B\rho(T)^3}}{\sqrt{\gamma C^T_{\mathtt{opt}}(\delta)}} + \frac{25\sqrt{B\alpha\rho(T)^3}}{\gamma \cdot C^T_{\mathtt{opt}}(\delta)}\right)\gamma \cdot C^T_{\mathtt{opt}}(\delta) \;+\; \beta \;,$$

for some constant β.

Proof. We divide the instance δ of length T given to the online algorithm into rounds. The first round starts with the first request in δ. A new round starts whenever $\rho(t) > \rho(t-1)$. We denote by K the number of rounds and by t_k the time step that started round k. Similarly to the proof of Theorem 4, we assume without loss of generality that $B \geq 1$. If $B \in [0, 1]$, we round B up to 1. Since all sets $\mathfrak{A}(t_k)$ are sets of online algorithms for $\mathcal{T}$ that satisfy both Assumption 1 (normalized expert costs) and Assumption 2 (bounded switching costs) for all $k \in \{1, \ldots, K\}$, we can use in each round k the online learning algorithm `GAD` with set of experts $\mathcal{E} = \mathfrak{A}(t_k)$. Let $C_{\mathtt{A},k}$ be the cost of `A` in round k, $C_{\mathtt{best},k}$ be the cost of the best expert in round k, and N_k be the size of $\mathfrak{A}(t_k)$. We have $N_k \leq 2^{\rho(t_k)}$. Furthermore, assume that, for each $k \in \{1, \ldots, K\}$,

$$C_{\mathtt{best},k} \;\geq\; \max\left\{\frac{9\ln N}{4B},\ B\ln N,\ 256B^2\right\} \;.$$

So, since $C_{\mathtt{A},k} \geq C_{\mathtt{best},k}$ and we assume that $B \geq 1$, we can use Theorem 3 to bound the expected total cost of the constructed randomized online algorithm $\mathtt{A}'$ from above by

$$\begin{aligned}
\mathbf{E}\Big[C^T_{\mathtt{A}'}(\delta)\Big] &\leq \sum_{k=1}^{K}\Big(C_{\mathtt{best},k} \;+\; 29\sqrt{BC_{\mathtt{best},k}\ln N_k}\Big) \\
&\leq \sum_{k=1}^{K}\Big(C_{\mathtt{A},k} \;+\; 29\sqrt{BC_{\mathtt{A},k}\ln N_k}\Big) \\
&\leq \sum_{k=1}^{K} C_{\mathtt{A},k} \;+\; 29\sqrt{B}\cdot\sum_{k=1}^{K}\sqrt{C_{\mathtt{A},k}\ln 2^{\rho(t_k)}}
\end{aligned}$$

$$
\begin{aligned}
&\leq \sum_{k=1}^{K} C_{\mathtt{A},k} \;+\; 29\sqrt{B}\cdot \sum_{k=1}^{K} \sqrt{C_{\mathtt{A},k} \ln 2^{\rho(T)}} \\
&\leq C_{\mathtt{A}}^{T}(\delta) \;+\; 25K\sqrt{BC_{\mathtt{A}}^{T}(\delta)\rho(T)} \\
&\leq \gamma \cdot C_{\mathtt{opt}}^{T}(\delta) \;+\; \alpha \;+\; 25\rho(T)\sqrt{B(\gamma C_{\mathtt{opt}}^{T}(\delta)+\alpha)\rho(T)} \\
&\leq \gamma \cdot C_{\mathtt{opt}}^{T}(\delta) \;+\; \alpha \;+\; 25\sqrt{B\gamma C_{\mathtt{opt}}^{T}(\delta)\rho(T)^3} + 25\sqrt{B\alpha\rho(T)^3} \\
&= \left(1 + \frac{25\sqrt{B\rho(T)^3}}{\sqrt{\gamma C_{\mathtt{opt}}^{T}(\delta)}} + \frac{25\sqrt{B\alpha\rho(T)^3}}{\gamma\cdot C_{\mathtt{opt}}^{T}(\delta)}\right)\gamma\cdot C_{\mathtt{opt}}^{T}(\delta) \;+\; \alpha \;,
\end{aligned}
$$

where the fifth inequality follows from $\sum_{k=1}^{K}\sqrt{C_{\mathtt{A},k}} \leq \sum_{k=1}^{K}\sqrt{C_{\mathtt{A}}^{T}(\delta)} = K\cdot\sqrt{C_{\mathtt{A}}^{T}(\delta)}$. □

6 Lower Bounds on the Advice Complexity

In this section, we show how we can use the methods studied in Section 3 and Section 5 to derive lower bounds on the advice complexity of online problems. For this, we consider the file allocation problem [48].

In the file allocation problem, we are given a set of data objects and a network of processors that have to access these objects. Our goal is to minimize the communication in the network by storing the objects on processors that use them frequently or that are close to them. Formally, we are given a weighted undirected graph $G = (V, E)$, where each node represents a processor, each edge represents a communication link between processors, and the weight of an edge is its bandwidth. We denote by $b(e)$ the bandwidth of edge $e \in E$ and by X the set of data objects. An instance of the file allocation problem consists of a sequence of read and write requests to the data objects in X. For $x \in X$, we denote by $R(x) \subseteq V$ the set of nodes that have currently a copy of data object x.

The cost of a request directly corresponds to the bandwidth required to serve the request: If a node $u \in V$ performs a read request on some object $x \in X$ and it has already a copy of x, i. e., $u \in R(x)$, then the required bandwidth is 0. Otherwise, it has to allocate a path through the network to some node $v \in R(x)$. This increases the load of each edge e on the path by $1/b(e)$. If a node $u \in V$ performs a write request on some object $x \in X$, it has to update all copies of x in the network. For this, it has to compute a Steiner tree and send the update to all nodes in $R(x)$. By this, it increases the load of each edge e on the Steiner tree by $1/b(e)$.

An online algorithm that manages the network can reallocate the data objects in the network after each request. In order to reallocate some object $x \in X$, it chooses a tree whose root is a node in $R(x)$ and copies x to an arbitrary subset of the nodes of the chosen tree. By this, it increases the load of each edge e on the tree by $1/b(e)$, where $D(x) \geq 1$.

In [45], a lower bound on the competitive ratio of $\Omega(\log_2 n/d)$ for randomized online algorithms was presented for the file allocation problem on graphs that are d-dimensional meshes with n nodes. Note that we can model the file allocation problem as a task system: The set of requests consists of all read and write requests of all nodes to all data objects; the set of answers consists of all subsets of edges of the graph for all data objects; the set of states consists of all assignments of data objects to nodes. In order to ensure that reallocations are done after answering a request, we can use the same trick as in Section 4 for the list update problem. If we scale all costs by $1/|E|$, we can use the method studied in Section 5 to give a lower bound on the advice complexity of each online algorithm with advice that has a competitive ratio smaller than $\Omega(\log_2 n/d)$.

Theorem 9. *Each online algorithm with advice for the file allocation problem on graphs that are d-dimensional meshes with n nodes and with a competitive ratio smaller than $\Omega(\log_2 n/d)$ has an advice complexity larger than $\rho(T) = \log_2 T$.*

Proof. Assume that there is an online algorithm with advice $\mathtt{A}$ for this file allocation problem with a competitive ratio smaller than $c \cdot (\log_2 n/d)$ that has an advice complexity of $\rho(T) = \log_2 T$. For all $T \in \mathbb{N}$, the associated set of online algorithms $\mathfrak{A}(T)$ of each feasible online algorithm with advice satisfies Assumption 1 (normalized expert costs) since by scaling all feasible online algorithms have costs between zero and one in each time step. Furthermore, for all $T \in \mathbb{N}$, $\mathfrak{A}(T)$ satisfies Assumption 2 (bounded switching costs) with $B = |X|$ since by scaling all state changes can be done with costs of at most $|X|$. Consider an instance δ of length T with $C^T_{\mathtt{opt}}(\delta) \in \omega\Big(\rho(T) \cdot \log_2^2 \rho(T)\Big) = \omega\Big(\log_2 T \cdot \log_2^2 \log_2 T\Big)$. Note that

$$\begin{aligned}
&\frac{70 \log_2 \rho(T) \sqrt{B\rho(T)}}{\sqrt{\gamma C^T_{\mathtt{opt}}(\delta)}} + \frac{70 \log_2 \rho(T) \sqrt{B\alpha\rho(T)}}{\gamma C^T_{\mathtt{opt}}(\delta)} \\
&= \frac{70 \log_2 \log_2 T \sqrt{B \log_2 T}}{\sqrt{\gamma C^T_{\mathtt{opt}}(\delta)}} + \frac{70 \log_2 \log_2 T \sqrt{B\alpha \log_2 T}}{\gamma C^T_{\mathtt{opt}}(\delta)} \\
&\in o(1) \ .
\end{aligned}$$

Therefore, we can apply Theorem 8 to the online algorithm with advice $\mathtt{A}$ to construct a randomized online algorithm $\mathtt{A}'$ with competitive ratio smaller than

$$c \cdot (\log_2 n/d) \cdot \left(1 + \frac{70 \log_2 \log_2 T \sqrt{B \log_2 T}}{\sqrt{\gamma C^T_{\mathtt{opt}}(\delta)}} + \frac{70 \log_2 \log_2 T \sqrt{B\alpha \log_2 T}}{\gamma C^T_{\mathtt{opt}}(\delta)}\right)$$

and, for each $\varepsilon > 0$, we can find a constant β such that the randomized online algorithm $\mathtt{A}'$ has a competitive ratio smaller than $c \cdot (\log_2 n/d) \cdot (1 + \varepsilon)$. However, this is a contradiction to the lower bound on the competitive ratio given in [45]. $\square$

Acknowledgment

The authors would like to thank Juraj Hromkovič for many helpful discussions.

References

[1] S. Albers, B. von Stengel, R. Werchner. A combined BIT and TIMESTAMP algorithm for the list update problem. *Information Processing Letters* 56(3):135–139. Elsevier 1995.

[2] K. Barhum. Tight bounds for the advice complexity of the online minimum Steiner tree problem. In *Proc. of SOFSEM 2014*, LNCS 8327, pp. 77–88, Springer 2014.

[3] K. Barhum, H.-J. Böckenhauer, M. Forišek, H. Gebauer, J. Hromkovič, S. Krug, J. Smula, B. Steffen. On the power of advice and randomization for the disjoint path allocation problem. In *Proc. of SOFSEM 2014*, LNCS 8327, pp. 89–101, Springer 2014.

[4] M. P. Bianchi, H.-J. Böckernhauer, T. Brülisauer, D. Komm, B. Palano. Online minimum spanning tree with advice. In *Proc. of SOFSEM 2016*, to appear.

[5] M. P. Bianchi, H.-J. Böckenhauer, J. Hromkovič, L. Keller. Online coloring of bipartite graphs with and without advice. In *Proc. of COCOON 2012*, LNCS 7434, pp. 519–530, Springer 2012.

[6] M. P. Bianchi, H.-J. Böckenhauer, J. Hromkovič, S. Krug, B. Steffen. On the advice complexity of the online L(2,1)-coloring problem on paths and cycles. In *Proc. of COCOON 2013*, LNCS 7936, pp. 53–64. Springer 2013.

[7] S. Ben-David, A. Borodin, R. M. Karp, G. Tardos, A. Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, Springer 1994.

[8] A. Blum. On-line algorithms in machine learning. *Online Algorithms: The State of the Art*, LNCS 1442, pp. 306–325, Springer 1998.

[9] A. Blum and Y. Mansour. Learning, regret minimization, and equilibria. *Algorithmic Game Theory*, pp. 79–101, Cambridge University Press 2007.

[10] H.-J. Böckenhauer, R. Dobson, S. Krug, K. Steinhöfel. On energy-efficient computations with advice. in *Proc. of COCOON 2015*, LNCS 9198, pp. 747–758, Springer 2015.

[11] H.-J. Böckenhauer, J. Hromkovič, D. Komm, R. Královič, P. Rossmanith. On the power of randomness versus advice in online computation. In *Languages Alive: Essays Dedicated to Jürgen Dassow on the Occasion of His 65th Birthday*, LNCS 7300, pp. 30–43, Springer 2012.

[12] H.-J. Böckenhauer, J. Hromkovič, D. Komm, S. Krug, J. Smula, A. Sprock. The string guessing problem as a method to prove lower bounds on the advice complexity. In *Proc. of COCOON 2013*, LNCS 7936, pp. 493–505, Springer 2013.

[13] H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič. On the advice complexity of the k-server problem. In *Proc. of ICALP 2011*, LNCS 6755, pp. 207–218. Springer 2011.

[14] H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič, T. Mömke. On the advice complexity of online problems. In *Proc. of ISAAC 2009*, LNCS 5878, pp. 331–340. Springer 2009.

[15] H.-J. Böckenhauer, D. Komm, R. Královič, P. Rossmanith. The online knapsack problem: advice and randomization. *Theoretical Computer Science* 527, 2014, pp. 61–72.

[16] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[17] J. Boyar, S. Kamali, K. S. Larsen, A. López-Ortiz. Online bin packing with advice. In *Proc. of STACS 2014*, LIPIcs 25, pp. 174–186, Schloss Dagstuhl 2014.

[18] J. Boyar, L. M. Favrholdt, Ch. Kudahl, J. W. Mikkelsen. Advice Complexity for a Class of Online Problems. In *Proc. of STACS 2015*: LIPIcs 30, pp. 116–129, Schloss Dagstuhl 2015.

[19] J. Boyar, S. Kamali, K. S. Larsen, A. López-Ortiz. On the list update problem with advice. In *Proc. of LATA 2014*, LNCS 8370, Springer 2014.

[20] A. Borodin, N. Linial, M. E. Saks. An optimal online algorithm for metrical task systems. *Journal of the ACM* 39(4):745–763, 1992.

[21] L. Devroye. *Non-Uniform Random Variate Generation.* Springer 1986.

[22] R. Dorrigiv, M. He, N. Zeh. On the advice complexity of buffer management. In *Proc. of ISAAC 2012*, LNCS 7676, pp. 136–145, Springer 2012.

[23] S. Dobrev, R. Královič, R. Královič. Independent set with advice: the impact of graph knowledge. In *Proc. of WAOA 2012*, LNCS 7846, pp. 2–15, Springer 2012.

[24] S. Dobrev, R. Královič, E. Markou. Online graph exploration with advice. In *Proc. of SIROCCO 2012*, LNCS 7355, pp. 267–278, Springer 2012.

[25] S. Dobrev, R. Královič, D. Pardubská. How much information about the future is needed? In *Proc. of SOFSEM 2008*, LNCS 4910, pp. 247–258. Springer 2008.

[26] J. Dohrau. Online makespan scheduling with sublinear advice. Technical Report, ETH Zurich, 2013.

[27] Y. Emek, P. Fraigniaud, A. Korman, A. Rosén. Online computation with advice. In *Proc. of ICALP 2009*, LNCS 5555, pp. 427–438, Springer 2009.

[28] P. Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory* 21(2):194–203, 1975.

[29] M. Forišek, L. Keller, M. Steinová. Advice complexity of online coloring for paths. In *Proc. of LATA 2012*, LNCS 7183, pp. 228–239, Springer 2012.

[30] H. Gebauer, D. Komm, R. Královič, R. Královič, J. Smula. Disjoint path allocation with sublinear advice. In *Proc. of COCOON 2015*, LNCS 9198, pp. 417–429, Springer 2015.

[31] S. Geulen, B. Vöcking, M. Winkler. Regret minimization for online buffering problems using the weighted majority algorithm. In *Proc. of COLT 2010*, pp. 132–143, Omnipress 2010.

[32] S. Gupta, S. Kamali, A. López-Ortiz. On advice complexity of the k-server problem under sparse metrics. In *Proc. of SIROCCO 2013*, LNCS 8179, Springer 2013.

[33] R. L. Graham, D. E. Knuth, O. Patashnik: *Concrete Mathematics*, 2nd ed. Addison-Wesley 1994.

[34] J. R. Griggs and R. K. Yeh. Labelling graphs with a condition at distance 2. *SIAM Journal on Discrete Mathematics*, 5(4):586–595, 1992.

[35] J. Hannan. Approximation to Bayes risk in repeated plays. *Contributions to the Theory of Games* 3, pp. 97–139, Princeton University Press 1957.

[36] J. Hromkovič, R. Královič, R. Královič. Information complexity of online problems. In *Proc. of MFCS 2010*, LNCS 6281, pp. 24–36. Springer 2010.

[37] J. Hromkovič, T. Mömke, K. Steinhöfel, P. Widmayer. Job shop scheduling with unit length tasks: Bounds and algorithms. *Algorithmic Operations Research* 2(1):1–14, 2007.

[38] D. Komm and R. Královič. Advice complexity and barely random algorithms. *Theoretical Informatics and Applications (RAIRO)* 45(2):249–267. IEEE Computer Society 2011.

[39] D. Komm, R. Královič, R. Královič, J. Smula. Treasure hunt with advice. In *Proc. of SIROCCO 2015*, LNCS 9439, pp. 328–341. Springer 2015.

[40] D. Komm, R. Královič, T. Mömke. On the advice complexity of the set cover problem. In *Proc. of CSR 2012*, LNCS 7353, pp. 241–252. Springer 2012.

[41] A. R. Karlin, M. S. Manasse, L. A. McGeoch, S. Owicki. Competitive randomized algorithms for non-uniform problems. In *Proc. of SODA 1990*, pp. 301–309. Society for Industrial and Applied Mathematics, 1990.

[42] A. R. Karlin, M. S. Manasse, L. Rudolph, D. D. Sleator. Competitive snoopy caching. *Algorithmica* 3(1):79–119, Springer 1988.

[43] E. Koutsoupias and Ch. H. Papadimitriou. On the k-server conjecture. *Journal of the ACM* 42(5):971–983, 1995.

[44] N. Littlestone, M. K. Warmuth. The weighted majority algorithm. In *Proc. of FOCS 1989*, pp. 256–261. IEEE 1989.

[45] B. M. Maggs, F. Meyer auf der Heide, B. Vöcking, M. Westermann. Exploiting locality for networks of limited bandwidth. In *Proc. of FOCS 1997*, pp. 284–293, 1997.

[46] M. S. Manasse, L. A. McGeoch und D. D. Sleator. Competitive algorithms for on-line problems. In *Proc. of STOC 1988*, pp. 322–333. 1988.

[47] J. W. Mikkelsen. Randomization can be as helpful as a glimpse of the future in online computation. CoRR abs/1511.05886, 2015.

[48] F. Meyer auf der Heide, B. Vöcking, M. Westermann. Provably good and practical strategies for non-uniform data management in networks. In *Proc. of ESA 1999*, LNCS 1643, pp. 89–100, Springer 1999.

[49] M. P. Renault and A. Rosén. On Online algorithms with advice for the k-server problem. In *Proc. of WAOA 2012*, LNCS 7164, pp. 198–210, Springer 2012.

[50] M. P. Renault, A. Rosén, R. van Stee. Online algorithms with advice for bin packing and scheduling problems. *CoRR abs/1311.7589*, 2013.

[51] S. Seibert, A. Sprock, W. Unger. Advice complexity of the online vertex coloring problem. In *Proc. of CIAC 2013*, LNCS 7878, pp. 345–357, Springer 2013.

[52] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, Association for Computing Machinery, 1985.

[53] D. Wehner. A new concept in advice complexity of job shop scheduling. In *Proc. of MEMICS 2014*, LNCS 8934, pp. 147–158. Springer 2014.

[54] D. Wehner. Advice complexity of fine grained job shop scheduling. In *Proc. of CIAC 2015*, LNCS 9079, pp. 416–428. Springer 2015.

[55] A. C.-C. Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *Proc. of FOCS 1977*, pp. 222–227. IEEE Computer Society 1977.