Online Computation with Advice*

Yuval Emek † Pierre Fraigniaud‡ Amos Korman§ Adi Rosén¶

Abstract

We consider a model for online computation in which the online algorithm receives, together with each request, some information regarding the future, referred to as *advice*. The advice is a function, defined by the online algorithm, of the whole request sequence. The advice provided to the online algorithm may allow an improvement in its performance, compared to the classical model of complete lack of information regarding the future. We are interested in the impact of such advice on the competitive ratio, and in particular, in the relation between the size b of the advice, measured in terms of bits of information per request, and the (improved) competitive ratio. Since $b = 0$ corresponds to the classical online model, and $b = \lceil \log |\mathcal{A}| \rceil$, where $\mathcal{A}$ is the algorithm's action space, corresponds to the optimal (offline) one, our model spans a spectrum of settings ranging from classical online algorithms to offline ones.

In this paper we propose the above model and illustrate its applicability by considering two of the most extensively studied online problems, namely, *metrical task systems (MTS)* and the *k-server* problem. For MTS we establish tight (up to constant factors) upper and lower bounds on the competitive ratio of deterministic and randomized online algorithms with advice for any choice of $1 \leq b \leq \Theta(\log n)$, where n is the number of states in the system: we prove that any randomized online algorithm for MTS has competitive ratio $\Omega(\log(n)/b)$ and we present a deterministic online algorithm for MTS with competitive ratio $O(\log(n)/b)$. For the k-server problem we construct a deterministic online algorithm for general metric spaces with competitive ratio $k^{O(1/b)}$ for any choice of $\Theta(1) \leq b \leq \log k$.

*A preliminary version of this paper appeared in the Proceedings of ICALP 2009, pp. 427–438.

†Tel Aviv University, Tel Aviv, 69978 Israel. E-mail: yuvale@eng.tau.ac.il. This work was partially done during this author's visit at LIAFA, CNRS and University Paris Diderot, supported by Action COST 295 DYNAMO.

‡CNRS and University Paris Diderot, France. Email: pierre.fraigniaud@liafa.jussieu.fr. Additional support from the ANR project ALADDIN, by the INRIA project GANG, and by COST Action 295 DYNAMO.

§CNRS and University Paris Diderot, France. Email: amos.korman@liafa.jussieu.fr. Additional support from the ANR project ALADDIN, by the INRIA project GANG, and by COST Action 295 DYNAMO.

¶CNRS and University of Paris 11, France. Email: adiro@lri.fr. Research partially supported by ANR projects AlgoQP, QRAC and ALADDIN.

1 Introduction

Online algorithms are algorithms that receive their input piece by piece and have to act upon the receipt of each piece of input (a.k.a. request). Yet, their goal is usually to guarantee a performance which is as close as possible to the optimal performance achievable if the entire input is known in advance. How close do they get to this optimal performance is usually analyzed by means of competitive analysis (cf. [4]).

From a theoretical standpoint, the *complete* lack of knowledge about the future makes it many times impossible to achieve 'reasonable' competitive ratios. From a practical standpoint, complete lack of knowledge about the future does not always accurately model realistic situations. Consequently, several attempts have been made in the literature to somewhat relax the 'absolutely no knowledge' setting, and achieve better competitive ratios in such relaxed settings. Most notable are the setting where a limited number of steps into the future is known at any time (lookahead) (e.g., [1, 7, 19]), and the 'access graph' setting for paging (e.g., [5, 13]). These settings and their analyses are usually specific to the problem they address.

In this paper we study a new, general framework whose purpose is to model online algorithms which have access to *some* information about the future. This framework is intended to analyze the impact of such information on the achievable competitive ratio. One important feature of our framework is that it takes a *quantitative* approach for measuring the amount of information about the future available to an online algorithm. Roughly speaking, we define a finite *advice space* $\mathcal{U}$, and augment the power of the online algorithm `Alg` (and thus reduce the power of the adversary) by means of a series of *queries* u_t, $t = 1, 2, \ldots$, where u_t maps the whole request sequence σ (including the future requests) to an *advice* $u_t(\sigma) \in \mathcal{U}$ provided to `Alg` in conjunction with the t^{th} request of σ. This advice can then be used by the online algorithm to improve its performance. At the risk of a small loss of generality, we assume that the advice space is of size 2^b for some integer $b \geq 0$ and consider the advice to be a string of b bits.

Example 1. For the paging problem, it is relatively easy to verify that the following is a 1-competitive algorithm which uses 1 bit of advice per request (i.e., $|\mathcal{U}| = 2$) [10]. The bit of advice indicates whether the optimal offline algorithm keeps in memory the requested page until the next request to that same page. The online algorithm tries to imitate the behavior of the offline algorithm: if the offline algorithm indeed keeps in memory the requested page until the next request to that same page, then so does the online algorithm. Whenever a page must be swapped out from memory, the online algorithm picks an arbitrary page among all pages that are not supposed to remain in memory until they are requested again.

Clearly, since for a 'usual' online problem the set of all possible request sequences is infinite, our framework just imposes some 'commitment' of the adversary regarding the future. This reduces the power of the adversary, and gives to the online algorithm some information about the future. Since (typically) an online algorithm has at any time a finite set of possible actions, our setting

additionally provides a smooth spectrum of computation models whose extremes are (classical) online computation with no advice (advice space of size 1) and optimal, offline computation, where the advice is simply the optimal action (the advice space corresponds to the set of all possible actions).

The main motivation for studying online algorithms that receive a small advice with each request is purely theoretical. Nevertheless, this framework may be motivated by settings such as the following, which may be dubbed 'spy behind enemy lines': an entity which is aware of the plans of the adversary collaborates with the online algorithm, however the communication between this entity and the online algorithm is limited in terms of its capacity.

In this work we concentrate on two of the most extensively studied online problems, *metrical task systems (MTS)* and the *k-server* problem. We establish several (upper and lower) bounds on the achievable competitive ratios for these problems by online algorithms with advice, thus demonstrating the applicability of our approach for online problems, and giving a more refined analysis for online algorithms having some information about the future. Specifically, for MTS we establish asymptotically tight upper and lower bounds by proving Theorems 1 and 2.

Theorem 1. *Any randomized online algorithm for uniform n-node MTS with $1 \leq b \leq \Theta(\log n)$ bits of advice per request cannot be ρ-competitive against an oblivious adversary unless $\rho = \Omega(\log(n)/b)$.*

Theorem 2. *For any choice of $1 \leq b \leq \lceil \log n \rceil$, there exists a deterministic online algorithm for general n-node metrical task systems that receives b bits of advice per request and whose competitive ratio is $O(\log(n)/b)$.*

For the k-server problem we first prove Theorem 3 and then generalize it to establish Theorem 4.

Theorem 3. *There exists an $O(\sqrt{k})$-competitive deterministic algorithm for the k-server problem that receives $O(1)$ bits of advice per request.*

Theorem 4. *For any choice of $\Theta(1) \leq b \leq \lceil \log k \rceil$, there exists a deterministic online algorithm for the k-server problem that receives b bits of advice per request and whose competitive ratio is $k^{O(1/b)}$.*

1.1 Related work

Online algorithms operating against restricted adversaries have been considered in the literature on many occasions, and under different settings. For example, online algorithms that operate against an adversary that has to provide some lookahead into the future have been considered, e.g., for the list accessing problem [1], the bin-packing problem [19], and the paging problem [7]. Another example is the model of 'access graph' for the paging problem [5, 13].

The notion of advice is central in computer science (actually, checking membership in NP-languages can be seen as computing with advice). In particular, the concept of advice and the analysis of its size and its impact on various computations has recently found various applications

in distributed computing. It is for instance present in frameworks such as informative labeling for graphs [27], distance oracles [28], and proof labeling [22, 23]. A formalism of computing with advice based on a pair of collaborative entities, usually referred to as an oracle and an algorithm, has been defined in [17] for the purpose of differentiating the broadcast problem from the wake-up problem. This framework has been recently used in [16] for the design of distributed algorithms for computing minimum spanning trees (MST), in [15] for tackling the distributed coloring problem, and in [26] for analyzing the graph searching problem (a.k.a. the cops-and-robbers problem). Other applications can be found in [8, 18, 20]. In the framework of computing with advice, the work probably most closely related to the present one is the work of Dobrev, Královič, and Pardubská [10] who essentially prove that there is a 1-competitive online algorithm for the paging problem, with 1 bit of advice[1] (see Example 1).

Online algorithms (without advice) for metrical task systems have been extensively studied. For deterministic algorithms it is known that the competitive ratio is exactly $2n-1$, where n is the number of states in the system [6]. For randomized algorithms, the known upper bound for general metrical task systems is $O(\log^2 n \log\log n)$ [12, 14] and the known lower bound is $\Omega(\log n/\log\log n)$ [2, 3]. For uniform metric spaces the randomized competitive ratio is known to be $\Theta(\log n)$ [6, 21].

For the k-server problem the best competitive ratio for deterministic algorithms on general metric spaces is $2k-1$ [24], and the lower bound is k [25]. Randomized algorithms for the k-server problem (against oblivious adversaries) are not well understood: it is known that in general metric spaces no algorithm has competitive ratio better than $\Omega(\log k/\log\log k)$ [2, 3], but no upper bound better than the one of [24] (that holds for deterministic algorithms) is known.

1.2 Organization

The rest of the paper is organized as follows. In Section 2 we give the necessary preliminaries. Our lower bound for metrical task systems is established in Section 3 and the matching upper bound is established in Section 4. In Section 5 we present our results for the k-server problem. We conclude in Section 6 with some further discussion and open problems.

2 Preliminaries

An online algorithm is an algorithm that receives its input piece by piece. Each such piece is an element in some set $\mathcal{S}$ and we refer to it as a *request*. Let σ be a finite request sequence. The t^{th}

[1] The model and interests of [10] actually differ from ours in two aspects. First, they are interested in the amount of information required in order to obtain online algorithms with optimal performance, rather than improved competitive ratios. Second, they allow the advice to be of variable size, including size zero, and concentrate their work on the question of how much below 1 can the average size of the advice be. This is done by means of encoding methods such as encoding the 3-letter alphabet $\{\emptyset, 0, 1\}$ using one bit only.

request is denoted $\sigma[t] \in \mathcal{S}$. The online algorithm has to perform an action upon the receipt of each request, that is, at *round* t, $1 \leq t \leq |\sigma|$, this action has to be performed when the online algorithm only knows the requests $\sigma[1], \ldots, \sigma[t]$.

To formally define a deterministic online algorithm we use the formulation of [4] (cf. Chapter 7). A deterministic online algorithm is a sequence of functions $g_t : \mathcal{S}^t \to \mathcal{A}_t$, $t \geq 1$, where $\mathcal{A}_t$ is the set of possible actions for request t (in many cases all $\mathcal{A}_t$ are identical and we denote them by $\mathcal{A}$.) In this work we strengthen the online algorithm (and thus weaken the adversary) in the following manner. For some finite set $\mathcal{U}$, referred to as the *advice space*, the online algorithm is augmented by means of a sequence of *queries* $u_t : \mathcal{S}^* \to \mathcal{U}$, $t \geq 1$. The value of $u_t(\sigma)$, referred to as *advice*, is provided to the online algorithm in each round $1 \leq t \leq |\sigma|$. The complexity of the advice is defined to be $\log |\mathcal{U}|$. For simplicity of presentation, and at the risk of an inaccuracy in our results by a factor of at most 2, we only consider advice spaces of size 2^b for some integer $b \geq 0$, and view the advice as a string of b bits.

Formally, a deterministic online algorithm with advice is a sequence of pairs (g_t, u_t), $t \geq 1$, where $g_t : \mathcal{S}^t \times \mathcal{U}^t \to \mathcal{A}_t$, and $u_t : \mathcal{S}^* \to \mathcal{U}$. Given a finite sequence of requests $\sigma = (\sigma[1], \ldots, \sigma[\ell])$, the action taken by the online algorithm in round t is

$$g_t(\sigma[1], \ldots, \sigma[t], u_1(\sigma), \ldots, u_t(\sigma)) \ .$$

A *randomized* online algorithm with advice is allowed to make random choices (i.e., 'toss coins') to determine its actions (the functions g_t) and the advice scheme (the queries u_t). Formally, then, a randomized online algorithm with advice is a probability distribution over deterministic online algorithms with advice.

A deterministic online algorithm $\texttt{Alg}$ (with or without advice) is said to be *c-competitive* if for all finite request sequences σ, we have $\texttt{Alg}(\sigma) \leq c \cdot \texttt{Opt}(\sigma) + \beta$, where $\texttt{Alg}(\sigma)$ is the cost incurred by $\texttt{Alg}$ on σ, $\texttt{Opt}(\sigma)$ is the cost incurred by an optimal (offline) algorithm on σ, and β is a constant which does not depend on σ. If the above holds with $\beta = 0$, then $\texttt{Alg}$ is said to be *strictly c-competitive*. For a randomized online algorithm (with or without advice) we consider the expectation (over the random choices of the algorithm) of the cost incurred by $\texttt{Alg}$ on σ. Therefore a randomized online algorithm $\texttt{Alg}$ (with or without advice) is said to be *c-competitive* (against an oblivious adversary) if for all finite request sequences σ, we have $\mathbb{E}[\texttt{Alg}(\sigma)] \leq c \cdot \texttt{Opt}(\sigma) + \beta$.

As commonly done for the analysis of online algorithms, one may view the setting as a game between the online algorithm and an adversary that issues the request sequence round by round. In this framework, the values of the queries u_t can be thought of as commitments made by the adversary to issue a request sequence which is consistent with the advice seen so far. For an online algorithm $\texttt{Alg}$, augmented with advices in $\mathcal{U}$, we are interested in both the competitive ratio of $\texttt{Alg}$ and the advice complexity $\log |\mathcal{U}|$, and in the interplay between those.

Metrical Task Systems. A *metrical task system (MTS)* is a pair $(\mathcal{M}, \mathcal{R})$, where $\mathcal{M} = (V, \delta)$ is an n-point metric space[2], and $\mathcal{R} \subseteq (\mathbb{R}_{\geq 0} \cup \{\infty\})^n$ is a set of allowable *tasks*. The points in V are usually referred to as *states*. We assume without loss of generality that $\mathcal{M}$ is scaled so that the minimum distance between two distinct states is 1. When the metric space is uniform, we sometimes call the metrical task system a uniform n-node Metrical Task Systems (or MTS).

An instance I of $(\mathcal{M}, \mathcal{R})$ consists of an initial state s_0 and a finite task sequence $r_1, \ldots, r_m$, where $r_t \in \mathcal{R}$ for all $1 \leq t \leq m$. Consider some algorithm `Alg` for $(\mathcal{M}, \mathcal{R})$ and suppose that `Alg` is in state s at the beginning of round t (the algorithm is in state s_0 at the beginning of round 1). In round t `Alg` first moves to some state s' (possibly equal to s), incurring a *transition cost* of $\delta(s, s')$, and then processes the task r_t in state s', incurring a *processing cost* of $r_t(s')$. The *cost* incurred by `Alg` on I is the sum of the transition costs in all rounds and the processing costs in all rounds.

The k-server problem. Let $\mathcal{M} = (V, \delta)$ be a metric space. We consider instances of the k-server problem on $\mathcal{M}$, and when clear from the context, omit the mention of the metric space. At any given time, the k servers reside in some *configuration*, i.e., a subset $X \subseteq V$, $|X| = k$. The *distance* between two configurations X and Y, denoted by $D(X, Y)$, is defined as the weight of a minimum weight matching between X and Y.

An instance I of the k-server problem on $\mathcal{M}$ consists of an initial configuration X_0 and a finite node sequence $r_1, \ldots, r_m$, where $r_t \in V$ for all $1 \leq t \leq m$. Consider some algorithm `Alg` for the k-server problem on $\mathcal{M}$ and suppose that `Alg` is in configuration X at the beginning of round t (the algorithm is in configuration X_0 at the beginning of round 1). The request r_t must be *processed* by one of the k servers in round t, which means that `Alg` moves to some configuration Y such that $r_t \in Y$ (Y may be equal to X if $r \in X$), incurring a cost of $D(X, Y)$. The *cost* incurred by `Alg` on I is the sum of the costs incurred in all rounds.

Remark. Throughout the paper we denote the logarithm on base 2, as log (without a subscript). Logarithms on other bases are denoted with the base as a subscript.

3 A lower bound for MTS

In this section we prove Theorem 1, that is, we show that if a randomized online algorithm for uniform n-node MTS with $1 \leq b \leq \Theta(\log n)$ bits of advice per request is ρ-competitive, then $\rho = 1 + \Omega(\log(n)/b)$. For the sake of the analysis, we consider in this section a stronger model for online algorithms with advice, where the whole advice is provided at the beginning of the execution rather than round by round. This only makes our results stronger.

Our proof is based on a lower bound on the competitive ratio of randomized online algorithms

[2] Throughout the paper, we use the standard definition of a metric space consisting of a set V of points and a distance function δ.

with advice for an online problem we call *generalized matching pennies (GMP)*, and on a reduction from this problem to the MTS problem.

3.1 The GMP problem

An instance I of the (ϕ, τ)-GMP problem, for positive integers ϕ and τ, consists of a finite sequence σ of $m \geq 1$ requests, where $\sigma[t]$ is one of the *digits* $0, \ldots, \phi - 1$ for $1 \leq t \leq m$. (We refer to the integers $0, \ldots, \phi - 1$ as digits for reasons that will be clarified soon.) An algorithm for the GMP problem outputs, for each request $\sigma[t]$, $1 \leq t \leq m$, an action $a[t] \in \{0, \ldots, \phi - 1\}$; it incurs a cost of 0 if $\sigma[t] = a[t]$ and a cost of 1 otherwise. In addition, the algorithm incurs a *dummy cost* of $\lceil m/\tau \rceil$. Formally, the cost of an algorithm `Alg` on σ is

$$\lceil \frac{m}{\tau} \rceil + \sum_{t=1}^{m} Z_t \ ,$$

where

$$Z_t = \begin{cases} 0 & \text{if } a[t] = \sigma[t]; \\ 1 & \text{otherwise.} \end{cases}$$

An online algorithm for the GMP problem must output $a[1]$ before it sees request $\sigma[1]$, and must output $a[t]$, $t > 1$, as a function of $\sigma[1], \ldots, \sigma[t-1]$ only.[3]

We now give a lower bound on the competitive ratio of any randomized online algorithm with advice for the (ϕ, τ)-GMP problem. We first consider a single step of the GMP problem and regard it as a game between a max-player that chooses some digit $\sigma \in \{0, \ldots, \phi - 1\}$ and a min-player that chooses some digit $a \in \{0, \ldots, \phi - 1\}$. The min-player incurs a cost of 0 if $a = \sigma$ and a cost of 1 otherwise. Clearly, if the max-player chooses its action uniformly at random, then the expected cost incurred by the min-player is $1 - 1/\phi$. The following lemma provides a lower bound on the expected cost incurred by the min-player when the entropy in the mixed strategy of the max-player is not maximal.

Lemma 3.1. *Let S be the mixed strategy of the max-player (i.e., a probability distribution over the digits $0, \ldots, \phi - 1$). If $\mathrm{H}(S) \geq \delta \log \phi$, where $0 < \delta < 1$, then the expected cost incurred by the min-player is greater than $\delta - 1/\log \phi$.*

Proof. Let $p_i = \mathbb{P}(S = i)$ for every $0 \leq i \leq \phi - 1$. Assume without loss of generality that $p_0 \geq p_2 \geq \cdots \geq p_{\phi-1}$ and fix $p = p_0$. Clearly, the expected cost incurred by the min-player is minimized by choosing digit $a = 0$, in which case the expected cost is $1 - p$. We bound the entropy

[3] For clarity and simplicity, we give this definition, which does not fully conform to the definition of online algorithms of Section 2. Alternatively, one may add to the definition of the problem a dummy request $\sigma[0]$, define the output of the online algorithm $a[t]$, $0 \leq t \leq m$, to be a function of $\sigma[0], \ldots, \sigma[t]$, and define Z_t, $1 \leq t \leq m$ to be 0 if and only if $a[t-1] = \sigma[t]$ and 1 otherwise.

in S as follows:

$$\begin{aligned}
\mathrm{H}(S) &= \sum_{i=0}^{\phi-1} p_i \log\left(\frac{1}{p_i}\right) \\
&= p\log\left(\frac{1}{p}\right) + \sum_{i=1}^{\phi-1} p_i \log\left(\frac{1}{p_i}\right) \\
&\leq p\log\left(\frac{1}{p}\right) + (1-p)\log\left(\frac{\phi-1}{1-p}\right) \qquad (1) \\
&= p\log\left(\frac{1}{p}\right) + (1-p)\left(\log(\phi-1) + \log\left(\frac{1}{1-p}\right)\right) \\
&\leq 1 + (1-p)\log(\phi-1) \qquad (2) \\
&< (1-p)\log\phi + 1 \ ,
\end{aligned}$$

where (1) and (2) are due to Jensen's inequality. Since $\mathrm{H}(S) \geq \delta \log\phi$, it follows that $\delta\log\phi < (1-p)\log\phi + 1$, hence $1 - p > \delta - 1/\log\phi$ and the assertion holds. □

We can now prove the following theorem.

Theorem 3.2. *Let* `Alg` *be a ρ-competitive randomized online algorithm for the (ϕ,τ)-GMP problem that receives b bits of advice per request. If $\phi \geq 4$ and $1 \leq b \leq (1/3)\log\phi$ then $\rho = 1 + \Omega(\tau)$.*

Proof. We use Yao's principle and show that for an arbitrarily large integer L, there is a probability distribution over request sequences for the (ϕ,τ)-GMP problem such that (1) the optimal cost on any sequence in the support of this distribution is L; and (2) the expected cost, over this distribution, of any deterministic online algorithm with b bits of advice per request is at least $L \cdot (1 + \Omega(\tau))$.

We define this distribution to be the uniform distribution over all sequences of length $L\tau$. Clearly, the optimal cost on all of these sequences is L. It remains to establish the lower bound on the expected cost incurred by an arbitrary deterministic online algorithm `Alg` on this distribution. Let $m = L\tau$ and let σ be a request sequence drawn from the above distribution. For the purpose of the proof we fix arbitrary constants $1/2 < \gamma < \beta < \alpha < 1$ such that $b < (1-\alpha)\log\phi$. Note that this is possible since we assume that $b \leq (1/3)\log\phi$.

For $1 \leq t \leq m$, let X_t be the random variable that takes on the digit $\sigma[t] \in \{0,\ldots,\phi-1\}$. Let Y be the random variable that takes on the advice. So, `Alg` may base its choice of $a[t]$ on the knowledge of Y and $X_1,\ldots,X_{t-1}$.

Fix $X = (X_1,\ldots,X_m)$. Since $X_1,\ldots,X_m$ are chosen uniformly at random and independently, the entropy in X is $\mathrm{H}(X) = m\log\phi$. The advice Y is encoded by bm bits, thus $\mathrm{H}(Y) \leq bm$. A crucial observation is that Y is fully determined by X (this is how we defined the advice). Therefore $\mathrm{H}(Y \mid X) = 0$ and

$$\mathrm{H}(X \mid Y) = \mathrm{H}(X,Y) - \mathrm{H}(Y) = \mathrm{H}(X) - \mathrm{H}(Y) \geq m\log\phi - mb \ . \qquad (3)$$

On the other hand, by a straightforward variant of the chain rule of conditional entropy, we have

$$\mathrm{H}(X|Y) = \mathrm{H}(X_1, \ldots, X_m \mid Y) = \mathrm{H}(X_1 \mid Y) + \mathrm{H}(X_2 \mid X_1, Y) + \cdots + \mathrm{H}(X_m \mid X_1, \ldots, X_{m-1}, Y)\ . \quad (4)$$

By combining (3) and (4), we conclude that on average, a typical term in (4) is at least $\log\phi - b > \alpha\log\phi$. As each request admits ϕ possible digits, we have $\mathrm{H}(X_t \mid X_1, \ldots, X_{t-1}, Y) \leq \log\phi$, and hence the fraction of terms in (4) which are smaller than $\beta\log\phi$ is smaller than $\frac{1-\alpha}{1-\beta}$. Therefore the fraction of terms in (4) which are at least $\beta\log\phi$ is greater than $1 - \frac{1-\alpha}{1-\beta} \geq \frac{\alpha-\beta}{1-\beta}$.

Fix $T = \{1 \leq t \leq m \mid \mathrm{H}(X_t \mid X_1, \ldots, X_{t-1}, Y) \geq \beta\log\phi\}$. We know that $|T| > \frac{m(\alpha-\beta)}{1-\beta}$. (In fact, given a deterministic online algorithm with advice, the function $Y = Y(X)$ is known to us and the set T can be computed.) Consider some index $1 \leq t \leq m$. The amount of entropy that remains in request t after the online algorithm saw the advice (Y) and the previous requests $(X_1, \ldots, X_{t-1})$ is a random variable[4], denote it by H_t. Assuming that $t \in T$, we have $\mathbb{E}[H_t] = \mathrm{H}(X_t \mid X_1, \ldots, X_{t-1}, Y) \geq \beta\log\phi$. Since H_t is bounded from above by $\log\phi$, it follows that $\mathbb{P}(H_t < \gamma\log\phi) < \frac{1-\beta}{1-\gamma}$ and $\mathbb{P}(H_t \geq \gamma\log\phi) > 1 - \frac{1-\beta}{1-\gamma} = \frac{\beta-\gamma}{1-\gamma}$.

Recall that Z_t takes on the cost incurred by `Alg` on request t for every $1 \leq t \leq m$ — this is now a random variable. Let $Z = \sum_{t=1}^{m} Z_t$. We have $\mathbb{E}[Z_t] \geq \mathbb{E}[Z_t \mid H_t \geq \gamma\log\phi] \cdot \mathbb{P}(H_t \geq \gamma\log\phi)$. Lemma 3.1 guarantees that $\mathbb{E}[Z_t \mid H_t \geq \gamma\log\phi] > \gamma - 1/\log\phi$ which is at least $\gamma - 1/2$ since $\phi \geq 4$. Therefore for every $t \in T$, we have $\mathbb{E}[Z_t] > (\gamma - 1/2) \cdot \frac{\beta-\gamma}{1-\gamma}$ and by summing over all indices $t \in T$, we conclude that

$$\mathbb{E}[Z] > m \cdot \frac{\alpha-\beta}{1-\beta} \cdot \frac{\beta-\gamma}{1-\gamma}(\gamma - 1/2) = L\tau \cdot \frac{\alpha-\beta}{1-\beta} \cdot \frac{\beta-\gamma}{1-\gamma}(\gamma - 1/2)\ .$$

Since the dummy cost incurred by `Alg` on σ is L, it follows that the expected cost incurred by `Alg` is greater than $L(1 + \Omega(\tau))$. This completes the proof. □

3.2 Metrical Task Systems

To establish our lower bound for metrical task systems, we present a reduction from the GMP problem to MTS. That is, we build a randomized online algorithm with advice for the GMP problem from a randomized online algorithm with advice for uniform MTS.

Theorem 3.3. *Suppose that there exists a ρ-competitive randomized online algorithm with b bits of advice per request for uniform n-node MTS. Then there exists a ρ-competitive randomized online algorithm with $2b$ bits of advice per request for the (ϕ, τ)-GMP problem as long as $\phi^\tau \leq n/2$.*

[4] The random variable H_t maps the event $(X_1 = x_1) \wedge \cdots \wedge (X_{t-1} = x_{t-1}) \wedge (Y = y)$ to the entropy in X_t conditioned on that event. It should not be confused with the conditional entropy in X_t given $X_1, \ldots, X_{t-1}, Y$, denoted by $\mathrm{H}(X_t \mid X_1, \ldots, X_{t-1}, Y)$, which is the expected value of H_t, nor with the conditional self information in X_t given $X_1, \ldots, X_{t-1}, Y$ which is a random variable that maps the event $X_t = x_t$ conditioned on $(X_1 = x_1) \wedge \cdots \wedge (X_{t-1} = x_{t-1}) \wedge (Y = y)$ to minus the logarithm of the probability for that event.

Proof. Let $\texttt{Alg}_{\text{MTS}}$ be a ρ-competitive randomized online algorithm with b bits of advice per request for uniform n-node MTS. We design a randomized online algorithm $\texttt{Alg}_{\text{GMP}}$ with $2b$ bits of advice per request for the (ϕ, τ)-GMP problem. $\texttt{Alg}_{\text{GMP}}$ works by issuing requests to $\texttt{Alg}_{\text{MTS}}$ in an online manner, and deciding on its actions as a function of the actions taken by $\texttt{Alg}_{\text{MTS}}$. We first define the requests issued to $\texttt{Alg}_{\text{MTS}}$ and the actions taken by $\texttt{Alg}_{\text{GMP}}$ and then prove that $\texttt{Alg}_{\text{GMP}}$ is ρ-competitive.

Let σ_{GMP} be the request sequence for the GMP problem and let $m = |\sigma_{\text{GMP}}|$. We denote by σ_{MTS} the request sequence produced by $\texttt{Alg}_{\text{GMP}}$ for $\texttt{Alg}_{\text{MTS}}$. The request sequence σ_{GMP} is divided into *cycles*, where each cycle consists of τ consecutive requests (the last cycle may be shorter); σ_{MTS} is also divided into *cycles*, each consisting of $\tau + 1$ tasks.

Fix $n' = \phi^\tau$. We index the n states of the uniform MTS by the integers $0, \ldots, n-1$. Each task r of the MTS is of the form $r = \langle r(0), \ldots, r(n-1) \rangle \in \{0, \infty\}^n$, where $r(i) = \infty$ for every $2n' \leq i \leq n-1$. Furthermore, in odd (respectively, even) cycles the task r also satisfies $r(i) = \infty$ for every $n' \leq i \leq 2n'-1$ (resp., for every $0 \leq i \leq n'-1$). Note that this means that a competitive MTS algorithm can be in state i in an odd (resp., even) cycle only if $0 \leq i \leq n'-1$ (resp., $n' \leq i \leq 2n'-1$).

The initial state of the MTS is set to be n'. We now define the request sequence precisely. In what follows we consider some odd cycle c; the construction for even cycles is analogous. Let $\psi_1, \ldots, \psi_\tau \in \{0, \ldots, \phi-1\}$ be the τ GMP requests input to $\texttt{Alg}_{\text{GMP}}$ in cycle c and let $r_0, r_1, \ldots, r_\tau \in \{0, \infty\}^n$ be the $\tau+1$ tasks produced by $\texttt{Alg}_{\text{GMP}}$ (in an online fashion) for $\texttt{Alg}_{\text{MTS}}$. Task r_0 is defined as follows: set $r_0(i) = 0$ for every $0 \leq i \leq n'-1$ (recall that since r_0 is an odd cycle task, we have $r_0(i) = \infty$ for every other state i). For every integer $0 \leq i \leq n'-1$, let $u(i)$ be the τ-letter word over the alphabet $0, \ldots, \phi-1$ that represents i in base ϕ. Task r_t, $1 \leq t \leq \tau$, is designed so that $r_t(i) = 0$ if and only if the t most significant digits in $u(i)$ are exactly $\psi_1, \ldots, \psi_t$ (in that order); otherwise, $r_t(i) = \infty$. That is, there are $\phi^{\tau-t}$ states with zero processing cost in r_t (all other states have infinite processing cost); note that these states also had zero processing cost in r_{t-1}. Eventually, in the last task of cycle c (task r_τ) there remains a single state with zero processing cost — denote it s_c — while all other states have infinite processing cost.

Now, consider some $1 \leq t \leq \tau$ and suppose that $\texttt{Alg}_{\text{MTS}}$ serves task r_{t-1} in state $0 \leq i \leq n'-1$. Then the action a_t of $\texttt{Alg}_{\text{GMP}}$ in step t is the t^{th} most significant digit in $u(i)$, i.e., $a_t = \lfloor i/\phi^{\tau-t} \rfloor \mod \phi$.

Note that the request sequence σ_{MTS} consists of $m + \lceil \frac{m}{\tau} \rceil$ requests. Thus the advice that $\texttt{Alg}_{\text{MTS}}$ should receive is of size $b \cdot (m + \lceil \frac{m}{\tau} \rceil)$ bits. This is feasible since $\texttt{Alg}_{\text{GMP}}$ is assumed to receive $2b$ bits of advice per request which sums up to $2bm \geq b \cdot (m + \lceil \frac{m}{\tau} \rceil)$ bits of advice in total.

We now prove that $\texttt{Alg}_{\text{GMP}}$ is ρ-competitive. This is done by showing that $\texttt{Opt}(\sigma_{\text{GMP}}) \geq \texttt{Opt}(\sigma_{\text{MTS}})$ and that $\texttt{Alg}_{\text{GMP}}(\sigma_{\text{GMP}}) \leq \texttt{Alg}_{\text{MTS}}(\sigma_{\text{MTS}})$. The assertion follows since $\texttt{Alg}_{\text{MTS}}$ is assumed to be ρ-competitive.

First observe that by definition, $\texttt{Opt}(\sigma_{\text{GMP}}) = \lceil m/\tau \rceil$. On the other hand, σ_{MTS} can be served at cost $\lceil m/\tau \rceil$ by moving to state s_c at the beginning of cycle c and remaining there until the end of the cycle. (Recall that s_c admits a zero processing cost throughout the cycle.) Thus during each cycle the transition cost is 1 and the processing cost is 0. Since there are $\lceil m/\tau \rceil$ cycles, we conclude that $\texttt{Opt}(\sigma_{\text{MTS}}) \leq \lceil m/\tau \rceil = \texttt{Opt}(\sigma_{\text{GMP}})$.

To prove that $\texttt{Alg}_{\text{GMP}}(\sigma_{\text{GMP}}) \leq \texttt{Alg}_{\text{MTS}}(\sigma_{\text{MTS}})$ we consider σ_{GMP} and σ_{MTS} cycle by cycle. Consider some odd cycle c of σ_{GMP} (the analysis for even cycles is analogous) and some step $1 \leq t \leq \tau$ in that cycle. If $\texttt{Alg}_{\text{GMP}}$ incurs a cost of 1 (and not 0) at this step, then the action a_t of $\texttt{Alg}_{\text{GMP}}$ differs from the request ψ_t. We argue that in this case $\texttt{Alg}_{\text{MTS}}$ serves task r_{t-1} in some state i such that $r_t(i) = \infty$, and hence $\texttt{Alg}_{\text{MTS}}$ is forced to change its state in step t, incurring a unit transition cost. Indeed, the definition of the actions of $\texttt{Alg}_{\text{GMP}}$ implies that the t^{th} most significant digit of $u(i)$ is a_t while the construction of σ_{MTS} implies that for every state j admitting zero processing cost in task r_t, the t^{th} most significant digit of $u(j)$ is ψ_t.

Moreover, note that $\texttt{Alg}_{\text{MTS}}$ always incurs a unit transition cost when the cycle begins as at the end of the previous (even) cycle, or at the beginning of the execution if $c = 1$, $\texttt{Alg}_{\text{MTS}}$ must be in state $s_{c-1} \in \{n', \ldots, 2n' - 1\}$ while the first task of cycle c must be served in some state in $\{0, \ldots, n' - 1\}$. Summing over all cycles, we get a cost of $\lceil m/\tau \rceil$ which accounts for the dummy cost incurred by $\texttt{Alg}_{\text{GMP}}$. It follows that the total cost incurred by $\texttt{Alg}_{\text{GMP}}$ on σ_{GMP} is bounded from above by the total cost incurred by $\texttt{Alg}_{\text{MTS}}$ on σ_{MTS}. □

We are now ready to establish Theorem 1.

Proof of Theorem 1. Let $\texttt{Alg}_{\text{MTS}}$ be a ρ-competitive randomized online algorithm for uniform n-node MTS with $b \leq (1/6)\log(n/2)$ bits of advice per request. Fix $\phi = 2^{6b}$ and $\tau = \lfloor \log_\phi \frac{n}{2} \rfloor$. Note that ϕ and τ are positive integers satisfying $\phi^\tau \leq n/2$. Therefore by Theorem 3.3, there exists a ρ-competitive randomized online algorithm $\texttt{Alg}_{\text{GMP}}$ for the (ϕ, τ)-GMP problem with $2b$ bits of advice per request. Since $\phi \geq 4$ and $2b \leq (1/3)\log\phi$, Theorem 3.2 can be applied to conclude that $\rho = 1 + \Omega(\tau) = 1 + \Omega(\log(n)/b)$. □

4 An upper bound for MTS

In this section we establish Theorem 2 by presenting a deterministic online algorithm for general MTS that gets b, $1 \leq b \leq \log n$, bits of advice per request, and achieves a competitive ratio of $\lceil \frac{\lceil \log n \rceil}{b} \rceil = O(\log(n)/b)$. This algorithm is called `Follow`.

Let $(\mathcal{M}, \mathcal{R})$ be a metrical task system. The request sequence is divided into *cycles*, each consisting of $\alpha = \lceil \frac{\lceil \log n \rceil}{b} \rceil$ requests, with the last cycle possibly shorter. The first cycle, cycle 0, consists of the first α requests, the second one of the next α requests, and so on. During cycle

$i \geq 0$, `Follow` receives advice of $\lceil \log n \rceil$ bits, which indicate the state in which the optimal algorithm serves (will serve) the first request of cycle $i+1$.

For cycle i, let s_i be the state in which the optimal algorithm serves the first request of the cycle. Let OPT be the cost of the optimal algorithm on the whole request sequence and let OPT_i be the cost of the optimal algorithm during cycle i.

The operation of `Follow`. Before starting to serve cycle i, $i \geq 0$, `Follow` places itself at state s_i. This is possible (in the empty sense - at no cost) for the first request of cycle 0, because both the optimal algorithm and `Follow` start at the same initial state s_0. This is possible for any cycle $i > 0$, by moving, at the end of phase $i-1$, to state s_i, known to `Follow` by the advice given in cycle $i-1$.

To describe how `Follow` serves the requests in a cycle we give the following definition. Let $B_i(j)$, $j \geq 0$, be the set of states in the metrical task system that are at distance less than 2^j from s_i. I.e.,

$$B_i(j) = \{s : d(s, s_i) < 2^j\} .$$

We now partition the (at most) α requests of cycle i, into *phases*. When the cycle starts, phase 0 starts too. During phase j, `Follow` serves the requests by moving to the state, among the states in $B_i(j)$, which has the least processing cost for the given task, and serves the request there. A request no longer belongs to phase j, and phase $j+1$ starts, if serving the request according to the above rule, will bring the total *processing cost* since the cycle started to be at least 2^j. Note that if a given request belongs to some phase j, the next request may belong to phase $j' > j+1$. That is, there may be phases with no request.

To analyze the algorithm we first give a lower bound on the cost of the optimal algorithm in each cycle in terms of the number of phases that occurred in that cycle.

Lemma 4.1. *If the last request of cycle i belongs to phase k, $k \geq 1$, then $OPT_i \geq 2^{k-2}$.*

Proof. Let σ be the sequence of requests that belong to phase $k-1$, concatenated with the first request of phase k. Note that the sequence of requests that belong to phase $k-1$ may be empty, but the first request of phase k must exist.

We consider how the optimal algorithm serves σ, and assume by way of contradiction that $OPT_i < 2^{k-2}$. It follows that the optimal algorithm serves all the requests of σ in states within $B_i(k-2) \subseteq B_i(k-1)$, incurring for those a processing cost less than 2^{k-2}. Since for each of the requests of σ the optimal algorithm must at least incur the minimum processing cost among the states in $B_i(k-1)$, it follows that $\sum_{\ell=1}^{|\sigma|} \min_{s \in B_i(k-1)} \sigma_\ell(s) < 2^{k-2}$. Now observe that the processing cost incurred by `Follow` for those requests in cycle i which are before σ is less than 2^{k-2}. And by the above, the processing cost of `Follow` for σ is less than 2^{k-2}. But then the processing cost of

`Follow` in cycle i up to and including the last request of σ would be less than 2^{k-1}, and phase k would not have started at the last request of σ — a contradiction. □

We conclude this section with the following theorem.

Theorem 4.2. *`Follow` is $O(\alpha)$-competitive.*

Proof. We consider the cost incurred by `Follow` cycle by cycle. We denote by C_i the cost of `Follow` during cycle i. Let $C_i = C_i^s + C_i^t + C_i^*$, where, C_i^s is the processing cost during cycle i, C_i^t is the transition cost during cycle i, and C_i^* is the cost incurred by `Follow`, at the end of cycle i, to move to state s_{i+1} (we do not count this cost in C_i^t).

For cycle i we consider two cases.

Case 1: The last request of cycle i is in phase $k = 0$. It follows that $C_i^s < 1$ and this processing cost was incurred in state s_i. The optimal algorithm has either moved away from s_i, incurring a cost of at least 1, or serves the whole cycle in s_i, incurring a cost of exactly C_i^s. In any case $C_i^s \leq OPT_i$, and $C_i^t = 0$. We thus have for this case $C_i^s + C_i^t \leq OPT_i$.

Case 2: The last request of cycle i is in phase $k > 0$. By Lemma 4.1 we have that $OPT_i \geq 2^{k-2}$. We have $C_i^s < 2^k$ by definition of the phases. We further have $C_i^t \leq (\alpha - 1)2^{k+1}$, because `Follow` moves between states at most $\alpha - 1$ times, and stays during the whole cycle within $B_i(k)$. We thus have for this case $C_i^s + C_i^t \leq 8\alpha \cdot OPT_i$.

Now consider the cost C_i^* incurred by `Follow` to move to state s_{i+1} at the end of phase i. By the triangle inequality, this cost is at most $C_i^t + d(s_i, s_{i+1})$.

If the whole sequence has ℓ cycles, summing over all cycles we have

$$\begin{aligned}
\sum_{i=1}^{\ell} C_i &= \sum_{i=1}^{\ell} (C_i^s + C_i^t + C_i^*) \\
&\leq \sum_{i=1}^{\ell} (C_i^s + C_i^t) + \sum_{i=1}^{\ell-1} (C_i^t + d(s_i, s_{i+1})) \\
&\leq \sum_{i=1}^{\ell} 8\alpha \cdot OPT_i + \sum_{i=1}^{\ell} 8\alpha \cdot OPT_i + \sum_{i=1}^{\ell-1} d(s_i, s_{i+1}) \\
&\leq (16\alpha + 1)OPT \ .
\end{aligned}$$

The theorem follows. □

5 An upper bound for the k-server problem

In this section we establish Theorem 4. This is done in two stages. First, in Section 5.1 we present an online k-server algorithm, referred to as `Partition`, that receives $O(1)$ bits of advice per

request and whose competitive ratio is $O(\sqrt{k})$, thus establishing Theorem 3. Then, in Section 5.2, we use `Partition` in a recursive manner to obtain the desired upper bound: for every choice of $1 \leq j \leq \log k$, we present an online k-server algorithm that receives $6j$ bits of advice per request and whose competitive ratio is $O\left(k^{1/(j+1)}\right)$.

5.1 An $O(\sqrt{k})$-competitive k-server algorithm with advice of size $O(1)$

In this section we present a k-server algorithm that receives $O(1)$ bits of advice per request, and achieves a competitive ratio of $O(\sqrt{k})$, thus establishing Theorem 3. The algorithm, denoted hereafter `Partition`, works in *iterations*, where each iteration except, maybe, the last one, consists of k requests. For simplicity of presentation, we assume that the request sequence ρ satisfies $|\rho| \bmod k = 0$ so that the last iteration consists of k requests as well. The request sequence in iteration i is merely the subsequence of ρ that starts at request $\rho[(i-1)k+1]$ and ends at request $\rho[ik]$. We may sometimes use the notation (i,j), where $1 \leq i \leq |\rho|/k$ and $1 \leq j \leq k$, to denote round j in iteration i, namely, round $(i-1)k+j$. Therefore $\rho[i,j] = \rho[(i-1)k+j]$ stands for the request presented in round j of iteration i.

Let the initial configuration be A and the request sequence ρ. We define an order on the points of the metric space; we let the points of A be the first k points in that order (this last condition is necessary for technical reasons only for the recursive algorithms of Section 5.2). Let `Opt` denote an optimal (offline) algorithm for ρ, when starting at A. The configuration of `Opt` just before request $\rho[i,j]$ is presented is denoted $C_{\texttt{Opt}}[i,j]$. Note that for every iteration i except the last one, $C_{\texttt{Opt}}[i+1,1]$ is also the configuration of `Opt` at the end of the iteration, i.e., after serving the last request in iteration i. For convenience, if i is the last iteration then we may slightly abuse notation and let $C_{\texttt{Opt}}[i+1,1]$ denote the configuration of `Opt` at the end of iteration i.

For a configuration C and a request sequence σ we denote by $\texttt{Opt}(C,\sigma)$ the optimal cost to start at C and serve σ. We remark that in this notation the number of servers used to serve the request sequence is implicitly understood from the size of the configuration C.

5.1.1 The `Partition` algorithm

We subsequently assume that `Partition` has some information in addition to the request sequence so far. This information comes in pieces that are provided to the algorithm by means of the bits of advice, as described later on. In particular, it is assumed that for every iteration i, after serving the last request in that iteration, `Partition` knows the configuration $C_{\texttt{Opt}}[i+1,1]$. Using this knowledge, `Partition` moves its servers so that its configuration coincides with that of `Opt` at the beginning of the next iteration. This is done by implementing the following step.

Configuration matching. After serving the last request in iteration i, `Partition` moves the servers to configuration $B = C_{\texttt{Opt}}[i+1, 1]$ along a minimum weight matching, paying a cost of $D(X, B)$, where X is the configuration of `Partition` upon serving the last request in iteration i. Therefore, in what follows we assume that when each iteration starts, `Partition` and `Opt` are in the same configuration. (This is true by definition for the first iteration, when the algorithm starts.)

To serve the requests within each iteration we proceed as follows. Let $x_1^i, \ldots, x_k^i$ be the nodes occupied by the servers of `Partition` (and `Opt`) at the beginning of iteration i, ordered in the defined order. In this context, we ignore any identity that the servers may have and rename them from scratch at the beginning of each iteration: the server residing in node x_j^i at the beginning of iteration i is (re)named s_j^i for every $1 \leq j \leq k$. As both `Partition` and `Opt` agree on the configuration at the beginning of the iteration, and since the order $x_1^i, \ldots, x_k^i$ is predetermined (and known to `Partition`), it follows that the server residing in node x_j^i is named s_j^i in both executions.

We now partition the servers into two sets, namely, *heavy* servers and *light* servers, as follows. The decision whether a server is heavy or light is made based on the number of requests served by that server, according to `Opt`, during the iteration. A server s_j^i is said to be *heavier* than server $s_{j'}^i$ if, according to `Opt`, s_j^i serves more requests during iteration i than $s_{j'}^i$ does. We define the set of heavy servers to consist of the $\lfloor \sqrt{k} \rfloor$ heaviest servers; the set of light servers contains the rest of the servers.

As we show later on, `Partition` knows which servers are heavy and which are light at the beginning of iteration i for every $1 < i \leq |\rho|/k$ (this is made possible by employing the advice provided to `Partition` in previous rounds, as discussed later). In the first iteration we make an arbitrary heavy/light server classification by classifying an arbitrary subset of $\lfloor \sqrt{k} \rfloor$ servers as heavy, while the rest of the servers are classified as light. Clearly, this arbitrary classification may differ from the "right" one. The implications of such a "wrong" classification are discussed later on. (In the next section we will invoke `Partition` as a subroutine, and sometimes provide to it the heavy/light server classification also for the first iteration. Once again, this will be discussed later on.)

Based on the classification of the servers, the requests of the iteration are also partitioned into two sets. A request that `Opt` serves with a heavy (respectively, light) server is called a *heavy* (resp., *light*) request. We may also say at times that a round is *heavy* (resp., *light*) if the request presented in that round is heavy (resp., light). We define the *heavy subsequence* (resp., *light subsequence*) of iteration i to be the subsequence of the heavy (resp., light) requests and denote it by ρ_h^i (resp., ρ_l^i). Recall that when the iteration starts both `Partition` and `Opt` occupy the same configuration. Let $A^i = C_{\texttt{Opt}}[i, 1]$ denote the configuration at the beginning of iteration i and let $A_h^i \subset A^i$ and $A_l^i \subset A^i$ denote the configurations of the heavy servers and light serves, respectively, at the beginning of iteration i.

Serving the light subsequence. Each light request $r \in V$ is served by the closest light server, which then returns to its initial position. That is, let $x \in A_l^i$ be the node minimizing $\delta(x, r)$. Then the server residing in x serves the request at r and subsequently returns to x.

Serving the heavy subsequence. To serve the heavy subsequence we invoke the Work Function Algorithm (`WFA`) [9, 24] on the set of heavy servers, starting in configuration A_h^i, serving only the heavy subsequence (and ignoring the light requests). Note that this algorithm is invoked with $\lfloor \sqrt{k} \rfloor$ servers.

5.1.2 The advice

The above description of `Partition` assumes that certain information is available to the algorithm. Specifically, we assumed the following.

(a) At the end of each iteration $1 \leq i \leq |\rho|/k$, the algorithm knows $C_{\texttt{Opt}}[i+1, 1]$, which is the configuration of `Opt` at the end of iteration i.
(b) For each iteration $1 < i \leq |\rho|/k$, the algorithm knows the heavy/light server classification.
(c) For each iteration $1 \leq i \leq |\rho|/k$, the algorithm knows the classification of each request as light or heavy.

We now turn to show how to provide this information to the algorithm, using 4 bits of advice per request.

(a) Matching the configuration of `Opt`. In order to know $C_{\texttt{Opt}}[i+1, 1]$ at the end of iteration i, the following two bits of advice are provided to `Partition` in round (i, j) for every $1 \leq j \leq k$.
[1st bit of advice] `occupied_request`(i, j): this bit is set if the node which corresponds to the request presented in round (i, j) belongs to the configuration of `Opt` at the beginning of iteration $i+1$, that is, if $\rho[i, j] \in C_{\texttt{Opt}}[i+1, 1]$.
[2nd bit of advice] `occupied_node`(i, j): recall that $x_1^i, \ldots, x_k^i$ are the nodes of the configuration of `Opt` at the beginning of iteration i in the defined order; this bit is set if x_j^i belongs to the configuration of `Opt` at the beginning of iteration $i+1$, that is, if $x_j^i \in C_{\texttt{Opt}}[i+1, 1]$.

Let $C = C_{\texttt{Partition}}[i, 1]$ be the configuration of `Partition` at the beginning of iteration i and assume by induction that $C = C_{\texttt{Opt}}[i, 1]$ (the base case holds since `Partition` and `Opt` start at the same configuration). We assume without loss of generality that `Opt` is *lazy* (cf. Chapter 10 in [4]), namely, `Opt` moves a server to node x in round t only if $\rho[t] = x$ and x is not presently covered by a server. Therefore $C_{\texttt{Opt}}[i+1, 1] \subseteq C \cup \{\rho[i, j] \mid 1 \leq j \leq k\}$. It follows that the configuration $C_{\texttt{Opt}}[i+1, 1]$ can be deduced from the advice collection $\{\texttt{occupied\_request}(i, j), \texttt{occupied\_node}(i, j) \mid 1 \leq j \leq k\}$ provided to the online algorithm along iteration i, together with the knowledge of C.

(b) Heavy vs. light servers. Recall that $x_1^i, \ldots, x_k^i$ are the nodes of the configuration of `Opt` at the beginning of iteration i in a predetermined order. When iteration i starts, $1 < i \leq |\rho|/k$, our algorithm needs to know which of the servers are light and which are heavy. To implement this, an additional bit of advice is provided in round j of the previous iteration, namely, in round $(i-1, j)$. [3^{rd} bit of advice] `heavy_server`$(i-1, j)$: this bit is set if the server s_j^i residing at x_j^i when iteration i starts is heavy (in iteration i).
As each iteration lasts k rounds, the required information is available to `Partition` when iteration i starts.

(c) Heavy vs. light requests. Our algorithm is further designed so that it knows for each request, upon the receipt of this request, whether it is a light request or a heavy one. This is implemented simply by providing in round (i, j) a bit that distinguishes between the two possibilities. [4^{th} bit of advice] `heavy_request`(i, j): this bit is set if the request presented at round (i, j) is heavy.

5.1.3 Analysis

Denote the total cost incurred by `Opt` in the heavy (respectively, light) rounds of iteration i by $\text{cost}_h^*(i)$ (resp., $\text{cost}_l^*(i)$). The total cost incurred by `Opt` in iteration i is thus $\text{cost}^*(i) = \text{cost}_h^*(i) + \text{cost}_l^*(i)$. Denote the total cost incurred by `Partition` in the heavy (respectively, light) rounds of iteration i by $\text{cost}_h(i)$ (resp., $\text{cost}_l(i)$). An additional configuration matching cost is incurred by `Partition` upon completion of iteration i — denote this cost by $\text{cost}_m(i)$. The total cost incurred by `Partition` in iteration i is thus $\text{cost}(i) = \text{cost}_h(i) + \text{cost}_l(i) + \text{cost}_m(i)$. Recall that A^i, A_h^i, and A_l^i, denote the configuration at the beginning of iteration i, the configuration of the heavy servers at the beginning of iteration i, and the the configuration of the light servers at the beginning of iteration i, respectively.

We now prove an upper bound on the cost incurred by `Partition` during an iteration, in terms of the cost incurred by `Opt` during the same iteration. Consider iteration i for some $1 < i \leq |\rho|/k$. Observe first that $\text{cost}_h^*(i) \geq \texttt{Opt}(A_h^i, \rho_h^i)$, as $\texttt{Opt}(A_h^i, \rho_h^i)$ is the minimum possible cost of serving ρ_h^i with $|A_h^i| = \lfloor\sqrt{k}\rfloor$ servers initially residing in configuration A_h^i. The cost incurred by our algorithm in the heavy rounds of iteration i is $\text{cost}_h(i) = \texttt{WFA}(A_h^i, \rho_h^i)$. We now use the fact that `WFA` is not only $(2k-1)$-competitive [24], but is also strictly $(4k-2)$-competitive [11]. Therefore $\text{cost}_h(i) \leq 4\lfloor\sqrt{k}\rfloor\texttt{Opt}(A_h^i, \rho_h^i)$ and

$$\text{cost}_h(i) \leq 4\lfloor\sqrt{k}\rfloor\text{cost}_h^*(i) \ . \tag{5}$$

Now, observe that a light server cannot serve more than $\lfloor\sqrt{k}\rfloor$ requests in iteration i according to `Opt` (otherwise, this would imply that every heavy server serves at least as many requests which would account to $\lfloor\sqrt{k}\rfloor(\lfloor\sqrt{k}\rfloor + 1) + \lfloor\sqrt{k}\rfloor + 1 = (\lfloor\sqrt{k}\rfloor + 1)^2 > k$ requests in iteration i). We partition the light requests of iteration i according to which (light) server of `Opt` serves them. Let

s_j^i be a light server residing in node $x_j^i \in A_l^i$ at the beginning of iteration i. Let $\langle r_1, \ldots, r_m \rangle$, where $m \leq \lfloor \sqrt{k} \rfloor$, be the sequence of (light) requests served according to `Opt` by s_j^i in (light rounds of) iteration i. We will show that the cost incurred by `Partition` for serving each of the requests $r_1, \ldots, r_m$ is at most twice the total cost incurred by `Opt` for serving all these requests. This establishes a ratio of at most $2m \leq 2\lfloor \sqrt{k} \rfloor$ between the cost incurred by `Partition` for serving the light requests, and the cost incurred by `Opt` for serving them.

The total cost incurred by `Opt` in iteration i for serving the requests $r_1, \ldots, r_m$ is $\chi_j = \delta(x_j^i, r_1) + \sum_{t=1}^{m-1} \delta(r_t, r_{t+1})$. The cost incurred by `Partition` for serving each request r_i, $1 \leq i \leq m$, is $2 \min_{x \in A_l^i} \delta(x, r_i) \leq 2\delta(x_j^i, r_i)$. By the triangle inequality, this is at most $2\chi_j$. As each light server serves at most $\lfloor \sqrt{k} \rfloor$ (light) requests (in iteration i), we have

$$\text{cost}_l(i) \leq 2\lfloor \sqrt{k} \rfloor \text{cost}_l^*(i) \ . \tag{6}$$

It remains to bound from above the configuration matching cost incurred by `Partition` upon completion of the iteration, i.e., $\text{cost}_m(i)$. Let X be the configuration of `Partition` upon serving the last request of iteration i, and let B the configuration of `Opt` upon serving the last request of that iteration. By definition, $\text{cost}_m(i) = D(X, B)$. We argue that $D(X, B)$ can be bounded from above by $\text{cost}^*(i) + \text{cost}_h(i)$. To see that, observe that `Partition` always returns the light servers to their initial position after serving each light request. Thus, one can bring all k servers from configuration X back to configuration A^i (where the servers resided at the beginning of iteration i), incurring a cost of at most $\text{cost}_h(i)$. On the other hand, along iteration i, `Opt` moved its servers from configuration A^i to configuration B, incurring a cost of $\text{cost}^*(i)$, hence we must have $D(A^i, B) \leq \text{cost}^*(i)$. By the triangle inequality, we conclude that $D(X, B) \leq \text{cost}^*(i) + \text{cost}_h(i)$, therefore

$$\text{cost}_m(i) \leq \text{cost}_h^*(i) + \text{cost}_l^*(i) + \text{cost}_h(i) \ . \tag{7}$$

By combining inequalities (5), (6), and (7), it follows that $\text{cost}(i) = \text{cost}_h(i) + \text{cost}_l(i) + \text{cost}_m(i)$ incurred by `Partition` in iteration i, $1 < i \leq |\rho|/k$, satisfies

$$\text{cost}(i) \leq (8\lfloor \sqrt{k} \rfloor + 1)\text{cost}_h^*(i) + (2\lfloor \sqrt{k} \rfloor + 1)\text{cost}_l^*(i) \ .$$

In fact, the above analysis holds for any iteration as long as the "right" heavy/light server classification is provided to `Partition` at (or actually before) the beginning of the iteration. This property is cast in the following lemma.

Lemma 5.1. *Consider some iteration i of `Partition` and suppose that the heavy/light server classification is provided to `Partition` for that iteration. Then,* $\text{cost}(i) \leq 9\lfloor \sqrt{k} \rfloor \cdot \text{cost}^*(i)$.

The heavy/light server classification is provided to `Partition` by the bits of advice for every iteration i, $1 < i \leq |\rho|/k$, hence Lemma 5.1 can be applied to all such iterations. We are left with the analysis of iteration i for $i = 1$. For this iteration we give the following Lemma.

Lemma 5.2. $\text{cost}(1) \leq 9\lfloor\sqrt{k}\rfloor \cdot \text{cost}^*(1) + \beta$, *where* β *is a constant that does not depend on the request sequence (but may depend on other parameters of the instance such as the number of servers or the initial configuration) .*

Proof. Since the iteration consists of k requests, the claim is obvious for finite metric spaces where the diameter is bounded. In the following we prove this claim for infinite metric spaces.

Consider the first iteration of `Partition`, where its classification of servers into heavy or light may be wrong. Recall that A^1 denotes the configuration at the beginning of the iteration. Fix $\Delta = \max\{\delta(u,v) \mid u,v \in A^1\}$. Let H^* be the set of nodes in which the "genuine" heavy servers reside at the beginning of the iteration. Let H be the set of nodes resided by the servers which are classified as heavy servers by `Partition` at the beginning of the iteration. The line of arguments that established inequality (5) can be repeated to deduce that $\text{cost}_h(1) \leq 4\lfloor\sqrt{k}\rfloor\texttt{Opt}(H, \rho_h^1)$. We can bound $\texttt{Opt}(H, \rho_h^1)$ by noticing that it cannot exceed $D(H, H^*) + \text{cost}_h^*(1)$ as $\text{cost}_h^*(1)$ is the cost incurred by some execution that starts at H^* and serves ρ_h^1 with $\lfloor\sqrt{k}\rfloor$ servers. Since $D(H, H^*) \leq \lfloor\sqrt{k}\rfloor\Delta$, it follows that

$$\text{cost}_h(1) \leq 4\lfloor\sqrt{k}\rfloor\text{cost}_h^*(1) + O(k\Delta) \ . \tag{8}$$

Consider some "genuine" light server s_j^1 residing in node x_j^1 at the beginning of the first iteration and let $\langle r_1, \ldots, r_m\rangle$, where $m \leq \lfloor\sqrt{k}\rfloor$, be the sequence of requests served by s_j^1 in (light rounds of) the iteration according to `Opt`. Let γ be the cost incurred by `Opt` for serving the requests $r_1, \ldots, r_m$. If s_j^1 was classified as light by our algorithm, then by following the lines of arguments that established inequality (6), we conclude that the cost incurred by `Partition` for serving the requests $r_1, \ldots, r_m$ is at most $2m\gamma$.

Assume that s_j^1 was mistakenly classified as heavy by our algorithm. There must exist some server $s_{j'}^1$ residing in node $x_{j'}^1$ at the beginning of the first iteration, which was classified as light by our algorithm, where $\delta(x_{j'}^1, x_j^1) \leq \Delta$. Once again, by following the lines of arguments that established inequality (6), we conclude that the cost incurred by `Partition` for serving the requests $r_1, \ldots, r_m$ is at most $2m(\gamma + \Delta)$. As there are $\lfloor\sqrt{k}\rfloor$ nodes which were classified as heavy by our algorithm, and by the bound on m, it follows that

$$\text{cost}_l(1) \leq 2\lfloor\sqrt{k}\rfloor\text{cost}_l^*(1) + O(k\Delta) \ . \tag{9}$$

It is easy to verify that equation (7) holds for the first iteration as well. By combining equations (8), (9), and (7), we conclude that $\text{cost}(1) = \text{cost}_h(1) + \text{cost}_l(1) + \text{cost}_m(1)$ incurred by `Partition` in the first iteration satisfies

$$\text{cost}(1) \leq (8\lfloor\sqrt{k}\rfloor + 1)\text{cost}_h^*(1) + (2\lfloor\sqrt{k}\rfloor + 1)\text{cost}_l^*(1) + O(k\Delta) \leq 9\lfloor\sqrt{k}\rfloor\text{cost}^*(1) + O(k\Delta) \ . \tag{10}$$

□

Using Lemmas 5.1 and 5.2 we conclude with the following theorem.

Theorem 5.3. *`Partition` is an $O(\sqrt{k})$-competitive k-server algorithm that receives 4 bits of advice per request.*

5.2 A general k-server algorithm

Our goal in this section is to establish Theorem 4, i.e., to show that for any $1 \leq j \leq \log k$, there exists an online k-server algorithm $\texttt{Alg}_j$ that receives $6j$ bits of advice per request and has competitive ratio $O(k^{1/(j+1)})$. We define the algorithm recursively, where the base of the recursion, for $j = 1$, is $\texttt{Alg}_1 = \texttt{Partition}$ presented in Section 5.1 (subsequently, we assume familiarity with that section).

Overview. The main idea of $\texttt{Alg}_j$ is very similar to that of `Partition`: $\texttt{Alg}_j$ works in iterations of length k (that is, the number of servers available to $\texttt{Alg}_j$), where in each iteration the servers are partitioned into heavy servers and light servers according to the number of requests served by each server according to `Opt`; the number of heavy servers here is $\lfloor k^{1-1/(j+1)} \rfloor$. The light servers then serve the light requests according to the greedy rule presented in Section 5.1.1. The heavy requests are served by invoking $\texttt{Alg}_{j-1}$ with the heavy servers.

It is essential that in each iteration $\texttt{Alg}_j$ knows: (a) the configuration of `Opt`, at the beginning of the iteration, for the servers serving the requests of that iteration; (b) the heavy/light server classification of these servers; and (c) the heavy/light request classification of the requests of that iteration (see Section 5.1.2). For all iterations other than the first one, this information is available to $\texttt{Alg}_j$ through the bits of advice described in Section 5.1.2. Two additional bits of advice are used to ensure that this information is also available in the first iteration. The above holds for all iterations other than a number of iterations at the beginning of the requests sequence, all included in the first iteration of the highest-level algorithm (i.e., the algorithm called from the "outside"); we describe later the small technical addition to handle this first iteration.

5.2.1 The algorithm $\texttt{Alg}_j$, $j > 1$

We now define $\texttt{Alg}_j$ that controls k servers. The algorithm works in iterations. It is assumed that when each iteration starts, the servers of $\texttt{Alg}_j$ occupy the same points as those occupied by the servers of `Opt` that serve the requests of the iteration (this holds for all iterations other than a number of iterations at the beginning of the request sequence). At the beginning of each iteration, we classify the k servers into $\lfloor k^{1-1/(j+1)} \rfloor$ heavy servers and $k - \lfloor k^{1-1/(j+1)} \rfloor$ light servers, where the heavy servers are the $\lfloor k^{1-1/(j+1)} \rfloor$ servers that serve the biggest number of requests according to `Opt`, during that iteration. For iteration i, the requests served according to `Opt` by a light server are called light requests, and constitute the light subsequence of the iteration, denoted ρ_l^i. The

rest are called heavy requests, and constitute the heavy subsequence of the iteration, denoted ρ_h^i. Algorithm $\texttt{Alg}_j$ serves each of the above subsequences independently.

Let A^i be the initial configuration at the beginning of iteration i, and let $A_h^i \subset A^i$ and $A_l^i \subset A^i$ be the configurations of the heavy servers, and light serves, respectively, at the beginning of iteration i.

Serving the light subsequence. Each light request $r \in V$ is served by the closest light server, which then returns to its initial position. That is, let $x \in A_l^i$ be the node minimizing $\delta(x, r)$. Then the server residing in x serves the request at r and subsequently returns to x. (This is the same rule as described in Section 5.1.1.)

Serving the heavy subsequence. To serve the heavy subsequence we use the $k' = \lfloor k^{1-1/(j+1)} \rfloor$ heavy servers and invoke algorithm $\texttt{Alg}_{j-1}$. That is, algorithm $\texttt{Alg}_{j-1}$ is invoked with k' servers, starting at the initial configuration A_h^i, and serving the subsequence ρ_h^i.

Recall that $\texttt{Alg}_j$ runs in iterations and that when each iteration begins, the algorithm should hold an initial configuration and a heavy/light classification of its servers. We will soon describe how this information is provided for the first iteration. For all other iterations, the algorithm is provided with this information through the advice received along the previous iteration as described in Section 5.1.2. For the special case of the first iteration of the first recursive invocation, the algorithm chooses an arbitrary classification into $\lfloor k^{1-1/(j+1)} \rfloor$ heavy servers and $k - \lfloor k^{1-1/(j+1)} \rfloor$ light servers, and otherwise proceeds as described above.

Configuration change. Upon completing to serve the last request of the iteration, $\texttt{Alg}_j$ ensures that all its servers occupy the same points occupied by the server of $\texttt{Opt}$ that served the requests of the iteration. To do that $\texttt{Alg}_j$ moves its light servers to their required configuration, i.e., to the configuration of these servers at the end of the iteration, according to $\texttt{Opt}$. Note that during the current iteration of $\texttt{Alg}_j$ there is a sequence of iterations of $\texttt{Alg}_{j-1}$, all of which use (precisely) the heavy servers of $\texttt{Alg}_j$. The heavy servers of $\texttt{Alg}_j$ are therefore brought to their required configuration at the last stage of the last iteration of $\texttt{Alg}_{j-1}$. The only exception to that rule is for the first iteration of $\texttt{Alg}_j$ when $\texttt{Alg}_j$ is called from the "outside" and not by $\texttt{Alg}_{j+1}$ (i.e., we use a total of $6j$ bits of advice per request). In this case $\texttt{Alg}_j$ also moves its heavy servers to the configuration as specified by its own bits of advice. We count this cost separately in the analysis and do not include it in the cost of $\texttt{Alg}_j$ for this first iteration.

5.2.2 Advice

We now describe the bits of advice used by $\texttt{Alg}_j$, $j \geq 1$. The algorithm uses the same 4 bits of advice described in Section 5.1.2, plus two additional bits.

The first additional bit is used only during the first iteration of the algorithm. Let q be the index of the *last* iteration.
[5^{th} bit of advice] $\texttt{occupied\_node\_last}(1, j)$: let $x_1^q, \ldots, x_k^q$ be the nodes of the configuration of those servers of $\texttt{Opt}$ which serve the request sequence of iteration q, at the beginning of iteration q (according to the predetermined order); this bit is set if, according to $\texttt{Opt}$, x_j^q holds one of these severs at the end of iteration q.

This bit is a duplication of the second bit of advice in the last iteration. It is provided in the first iteration since the last iteration may be shorter than k requests, in which case the amount of information that can be encoded by the second bits of advice along that iteration may be insufficient.

The other additional bit is used to provide $\texttt{Alg}_j$ (when invoked as a subroutine of $\texttt{Alg}_{j+1}$) with the right heavy/light server classification for its first iteration. This bit can be thought of as a replacement for the 3^{rd} bit of advice that cannot be provided for the first iteration, since there is no preceding iteration.

Let $x_1^1, \ldots, x_k^1$ be the nodes occupied, at the beginning of iteration 1 of a given invocation of $\texttt{Alg}_j$, by the servers of $\texttt{Opt}$ which serve the request sequence of that iteration. The following is done at the first iteration of *the previous invocation* of $\texttt{Alg}_j$, for $1 \leq s \leq k$:
[6^{th} bit of advice] $\texttt{heavy\_server}(1, s)$: this bit is set if the server at x_s^1 is heavy for the first iteration of the next invocation of $\texttt{Alg}_j$.

In this way, when an algorithm $\texttt{Alg}_j$ is invoked, it is provided with the heavy/light server classification for the first iteration. Obviously, this cannot be done if we are facing the first invocation of $\texttt{Alg}_j$. We deal with that in the analysis.

In all, $\texttt{Alg}_j$ uses $6j$ bits of advice per request.

5.2.3 Analysis

We now turn to analyze the competitive ratio of our algorithm. We consider a request sequence ρ, an initial configuration A, and denote by $\texttt{Opt}$ the optimal algorithm starting at A and serving ρ. Let $\texttt{Alg}_J$ be the algorithm that handles the whole of ρ, that is, this is the algorithm that is called from the "outside" (in particular, we use a total of $6J$ bits of advice per request). We now analyze the cost incurred by $\texttt{Alg}_j$, $j \leq J$. Similarly to Section 5.1.3, we denote the total cost incurred by $\texttt{Alg}_j$ in the heavy (resp., light) rounds of iteration i by $\text{cost}_h(i)$ (resp., $\text{cost}_l(i)$). The additional configuration matching cost incurred by $\texttt{Alg}_j$ upon completion of iteration i is denoted by $\text{cost}_m(i)$. We further denote the total cost incurred by $\texttt{Opt}$ in the heavy (respectively, light)

rounds of iteration i of $\texttt{Alg}_j$ by $\text{cost}_h^*(i)$ (resp., $\text{cost}_l^*(i)$). Let $\text{cost}^*(i) = \text{cost}_h^*(i) + \text{cost}_l^*(i)$. Note that interleaved between the heavy and light requests of iteration i of $\texttt{Alg}_j$ there could be other requests, but the set of servers serving those according to $\texttt{Opt}$ is distinct of the set of servers serving, according to $\texttt{Opt}$, the requests of iteration i of $\texttt{Alg}_j$.

We distinguish between a *regular* iteration and a *special* iteration. Any iteration of $\texttt{Alg}_j$, $j \leq J$ which is included in the first iteration of $\texttt{Alg}_J$ is a *special* iteration. All other iterations of $\texttt{Alg}_j$, $j \leq J$ are *regular* iterations. We first deal with regular iterations. The important properties of a regular iteration of an algorithm $\texttt{Alg}_j$ are that (1) the initial configuration of the servers controlled by $\texttt{Alg}_j$ is identical to the configuration of the servers of $\texttt{Opt}$ that serve the requests of this iteration (2) at the beginning of that iteration the correct classification into heavy/light servers is known to $\texttt{Alg}_j$; and (3) all iterations of all $\texttt{Alg}_{j'}$, $j' < j$, invoked during a regular iteration of $\texttt{Alg}_j$ are themselves regular iterations. The configuration and the heavy/light servers classification, at the beginning of a special iteration, may differ from the "right" ones since we are unable to provide all the required bits of advice until the end of the first iteration of $\texttt{Alg}_J$. Once the first iteration of $\texttt{Alg}_J$ is over, all the information is available and correct. We now start with a regular iteration.

Lemma 5.4. *Consider iteration i of $\texttt{Alg}_j$ invoked on k servers and assume it is a regular iteration. Then* $\text{cost}(i) \leq 9k^{1/(j+1)}\text{cost}^*(i)$.

Proof. The assertion is proved by induction on j. Lemma 5.1 establishes the base case $j = 1$. Consider some $j > 1$ and suppose that the lemma holds for every $j' < j$.

We first consider the cost incurred by $\texttt{Alg}_j$ on serving the light requests. As each light server serves at most $\lfloor k^{1/(j+1)} \rfloor$ requests, we can repeat the line of arguments which establishes inequality (6), to conclude that

$$\text{cost}_l(i) \leq 2 \left\lfloor k^{1/(j+1)} \right\rfloor \text{cost}_l^*(i) \ .$$

As to the heavy requests, by the inductive hypothesis on $\texttt{Alg}_{j-1}$, and since $\texttt{Alg}_{j-1}$ is invoked with $\lfloor k^{1-1/(j+1)} \rfloor$ servers, it follows that

$$\text{cost}_h(i) \leq 9 \cdot \left(\left\lfloor k^{1-1/(j+1)} \right\rfloor \right)^{1/j} \text{cost}_h^*(i) \leq 9 \cdot k^{1/(j+1)} \, \text{cost}_h^*(i) \ .$$

It remains to bound from above the configuration matching cost $\text{cost}_m(i)$. The crucial observation is that the heavy servers of $\texttt{Alg}_j$ are exactly the servers used by $\texttt{Alg}_{j-1}$, hence the cost of placing them in the desired configuration is already accounted for in the cost incurred by $\texttt{Alg}_{j-1}$. The light servers of $\texttt{Alg}_j$ reside in their initial configuration at the end of iteration i, thus by the line of arguments that establishes inequality (7), we conclude that

$$\text{cost}_m(i) \leq \text{cost}_l^*(i) \ .$$

It follows that the total cost incurred by $\texttt{Alg}_j$ in iteration i is

$$\begin{aligned}\text{cost}_h(i) + \text{cost}_l(i) + \text{cost}_m(i) &\leq 9 \cdot k^{1/(j+1)} \text{cost}_h^*(i) + 2 \cdot k^{1/(j+1)} \text{cost}_l^*(i) + \text{cost}_l^*(i) \\ &= 9 \cdot k^{1/(j+1)} \text{cost}_h^*(i) + \left(2 \cdot k^{1/(j+1)} + 1\right) \text{cost}_l^*(i) \\ &\leq 9 \cdot k^{1/(j+1)} \text{cost}^*(i) \ .\end{aligned}$$

The assertion follows. □

We now give an upper bound on the cost incurred by $\texttt{Alg}_j$ in a special iteration.

Lemma 5.5. *Consider iteration i of $\texttt{Alg}_j$ invoked on k servers and assume it is a special iteration. Then $\text{cost}(i) \leq 9k^{1/(j+1)}\text{cost}^*(i) + \beta$, where β is a constant that does not depend on the request sequence (but may depend on other parameters of the instance such as the number of servers or the initial configuration).*

Proof. Since the iteration consists of k requests, the claim is obvious for finite metric spaces where the diameter is bounded. In the following we prove this claim for infinite metric spaces.

Let A be the initial configuration of the instance at hand, and let $\Delta = \max\{\delta(u, v) \mid u, v \in A\}$. Consider a special iteration of $\texttt{Alg}_j$, and let σ be the sequence of requests served by $\texttt{Alg}_j$ in that iteration.

For a special iteration it is no longer the case that the initial configuration of the servers of $\texttt{Alg}_j$ always coincides with that of the servers of $\texttt{Opt}$ that serve σ. This is because we take an arbitrary heavy/light server classification for the first iteration of the first invocation of every algorithm. Thus, if, say, $\texttt{Alg}_{j+1}$ takes an arbitrary heavy/light server classification, it also means that the servers provided to $\texttt{Alg}_j$ may not be the "right" ones. The bits of advice may then also not give the "right" information. We now take this fact into account in the analysis for a special iteration.

Let C denote the initial configuration of the servers of $\texttt{Alg}_j$ at the beginning of the iteration, and let $C_h \subset C$, and $C_l \subset C$ be the configuration of the heavy and light servers, respectively, as classified by $\texttt{Alg}_j$. Let C^* be the configuration of the servers of $\texttt{Opt}$ that serve σ, when the iteration starts, and let $C_h^* \subset C$ and $C_l^* \subset C$ be the configuration of the heavy and lights servers of $\texttt{Opt}$, respectively.

We now observe that $C \triangle C^* \subseteq A$, and that $C_h \triangle C_h^* \subseteq A$ and $C_l \triangle C_l^* \subseteq A$. This is because the points in A are the first points in the order used to define the bits of advice. Therefore the bits of advice may "err" only by replacing one point of A by another point of A.

We can now prove by induction on j that $\text{cost}(i) \leq 9k^{1/(j+1)}\text{cost}^*(i) + O(5^{j-1}k\Delta)$ (we make no attempt to minimize the additive constant). The base case of $j = 1$ is established by arguments similar to those in the proof of Lemma 5.2. We get that $cost(i) \leq 9\lfloor\sqrt{k}\rfloor\text{cost}^*(i) + O(k\Delta)$.

Now consider some $j > 1$ and suppose that the lemma holds for every $j' < j$. We first consider the cost incurred by $\texttt{Alg}_j$ on serving the light requests. As each light server serves at most $\lfloor k^{1/(j+1)} \rfloor$

requests, and the total number of (light) requests in the iteration is at most k, we can repeat the line of arguments which establishes inequality (6) together with the arguments in the proof of Lemma 5.2 to conclude that

$$\mathrm{cost}_l(i) \leq 2 \left\lfloor k^{1/(j+1)} \right\rfloor \mathrm{cost}_l^*(i) + 2k\Delta \ .$$

As to the heavy requests, observe that there are $\lceil \frac{k}{\lfloor k^{1-1/(j+1)} \rfloor} \rceil$ iterations of $\mathtt{Alg}_{j-1}$ in iteration i of $\mathtt{Alg}_j$. By the inductive hypothesis on $\mathtt{Alg}_{j-1}$, and since $\mathtt{Alg}_{j-1}$ is invoked with $\lfloor k^{1-1/(j+1)} \rfloor$ servers,

$$\begin{aligned} \mathrm{cost}_h(i) &\leq 9 \cdot \left(\left\lfloor k^{1-1/(j+1)} \right\rfloor \right)^{1/j} \mathrm{cost}_h^*(i) + \lceil \frac{k}{\lfloor k^{1-1/(j+1)} \rfloor} \rceil \cdot O(5^{j-2} \left\lfloor k^{1-1/(j+1)} \right\rfloor \Delta) \\ &\leq 9 \cdot k^{1/(j+1)} \mathrm{cost}_h^*(i) + O(5^{j-2} 2k\Delta) \ . \end{aligned}$$

It remains to bound from above the configuration matching cost $\mathrm{cost}_m(i)$. The crucial observation is that the heavy servers of $\mathtt{Alg}_j$ are exactly the servers used by $\mathtt{Alg}_{j-1}$, hence the cost of placing them in the desired configuration is already accounted for in the cost incurred by $\mathtt{Alg}_{j-1}$. The $k - \lfloor k^{1-1/(j+1)} \rfloor$ light servers of $\mathtt{Alg}_j$ reside in their initial configuration at the end of iteration i, thus by the line of arguments that establishes inequality (7) together with the arguments in the proof of Lemma 5.2, we conclude that

$$\mathrm{cost}_m(i) \leq \mathrm{cost}_l^*(i) + (k - \left\lfloor k^{1-1/(j+1)} \right\rfloor)\Delta \ .$$

It follows that the total cost incurred by $\mathtt{Alg}_j$ in iteration i is

$$\begin{aligned} \mathrm{cost}_h(i) + \mathrm{cost}_l(i) + \mathrm{cost}_m(i) &\leq 9 \cdot k^{1/(j+1)} \mathrm{cost}_h^*(i) + O(5^{j-2} 2k\Delta) + 2 \cdot k^{1/(j+1)} \ \mathrm{cost}_l^*(i) + 2k\Delta + \\ &\quad \mathrm{cost}_l^*(i) + (k - \left\lfloor k^{1-1/(j+1)} \right\rfloor)\Delta \\ &= 9 \cdot k^{1/(j+1)} \mathrm{cost}_h^*(i) + \left(2 \cdot k^{1/(j+1)} + 1\right) \mathrm{cost}_l^*(i) + O(5^{j-2} 2k\Delta) + 2k\Delta + \\ &\quad (k - \left\lfloor k^{1-1/(j+1)} \right\rfloor)\Delta \\ &\leq 9 \cdot k^{1/(j+1)} \mathrm{cost}^*(i) + O(5^{j-1} k\Delta) \ . \end{aligned}$$

□

Using Lemma 5.4 and 5.5 we can now conclude with the following theorem.

Theorem 5.6. *For any $1 \leq j \leq \log k$, there exists a $O(k^{1/(j+1)})$-competitive k-server algorithm with $6j$ bits of advice.*

Proof. Assume we use $6j$ bits of advice per request and use $\mathtt{Alg}_j$ to serve request sequence ρ starting at configuration A. Let $\Delta = \max\{\delta(u, v) \mid u, v \in A\}$. Assume $\mathtt{Alg}_j$ has ℓ iterations in total, and let $\mathrm{cost}(i)$ and $\mathrm{cost}^*(i)$ be the cost of $\mathtt{Alg}_j$ and of $\mathtt{Opt}$, respectively, during iteration i, for $1 \leq i \leq \ell$.

By Lemma 5.4 we have that $\text{cost}(i) \leq 9k^{1/(j+1)}\text{cost}^*(i)$ for $2 \leq i \leq \ell$. By Lemma 5.5 we have that $\text{cost}(1) \leq 9k^{1/(j+1)}\text{cost}^*(1) + \beta$, where β is a constant that does not depend on the request sequence. In addition, $\texttt{Alg}_j$ moves its heavy servers at the end of the first iteration to "correct" its configuration so that it coincides with that of `Opt`. This involves moving servers only between the points of A. Therefore the cost incurred by $\texttt{Alg}_j$ for these moves is at most $k\Delta$. □

6 Conclusions

We define a model for online computation with advice. The advice provides the online algorithm with some (limited) information regarding the future requests. Our model quantifies the amount of this information in terms of the size b of the advice measured in bits per request. This model does not depend on the specific online problem.

The applicability and usefulness of our model is demonstrated by studying, within its framework, two of the most extensively studied online problems: metrical task systems (MTS) and the k-server problem. For general metrical task systems we present a deterministic algorithm whose competitive ratio is $O(\log(n)/b)$. We further show that any online algorithm, even randomized, for MTS has competitive ratio $\Omega(\log(n)/b)$ if it receives b bits of advice per request. This lower bound is proved on uniform metric spaces. For the k-server problem we present a deterministic online algorithm whose competitive ratio is $k^{O(1/b)}$. Whether this is best possible is left as an open problem.

We believe that employing our model of online computation with advice may lead to other results, thus enhancing our understanding of the exact impact of the amount of knowledge an online algorithm has regarding the future on its competitive ratio.

References

[1] S. Albers, A Competitive Analysis of the List Update Problem with Lookahead, *Theor. Comput. Sci.*, Vol. 197, No. 1–2, pp. 95–109, 1998.

[2] Y. Bartal , B. Bollobás, and M. Mendel, Ramsey-type theorems for metric spaces with applications to online problems. *J. Comput. Syst. Sci.*, 72 (2006), pp. 890–921.

[3] Y. Bartal, M. Mendel, N. Linial, and A. Naor. On metric Ramsey-type phenomena. *Annals of Mathematics*, 162:643–710, 2005.

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

[5] A. Borodin, S.Irani, P. Raghavan, and B. Schieber, Competitive Paging with Locality of Reference, In *Proc. 23rd Ann. ACM symp. on Theory of computing (STOC)*, pp. 249–259, 1991.

[6] A. Borodin, N. Linial, and M. E. Saks, An optimal on-line algorithm for metrical task system., *J. ACM*, 39 (1992), pp. 745–763.

[7] D. Breslauer, On Competitive On-Line Paging with Lookahead, *Theor. Comput. Sci.*, Vol. 209, No. 1–2, pp. 365–375, 1998.

[8] R. Cohen, P. Fraigniaud, D. Ilcinkas, A. Korman, and D. Peleg. Label-Guided Graph Exploration by a Finite Automaton. In *32nd Int. Colloquium on Automata, Languages and Programming (ICALP)*, LNCS 3580, pp. 335-346, 2005.

[9] M. Chrobak and L.L. Larmore. The server problem and on-line games. In *On-line algorithms: Proc. of a DIMACS Workshop. DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 7, pages 11–64, 1991.

[10] Stefan Dobrev, Rastislav Královič, and Dana Pardubská. How much information about the future is needed? In *34th Int. Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, pp 247-258, 2008.

[11] Y. Emek, P. Fraigniaud, A.Korman, and A. Rosén. On the Additive Constant of the k-Server Work Function Algorithm. In *Proc. of WAOA 2009*. To appear. (also available at URL: http://arxiv.org/abs/0902.1378v1.)

[12] J. Fakcharoenphol , S. Rao, and K. Talwar, A tight bound on approximating arbitrary metrics by tree metrics., J. Comput. Syst. Sci., 69 (2004), pp. 485–497.

[13] A. Fiat and A. Karlin, Randomized and Multipointer paging with locality of reference, In *Proc. 27th Ann. ACM symp. on Theory of computing (STOC)*, pp. 626–634, 1995

[14] A. Fiat and M. Mendel, Better algorithms for unfair metrical task systems and applications., SIAM J. Comput., 32 (2003), pp. 1403–1422.

[15] P. Fraigniaud, C. Gavoille, D. Ilcinkas, and A. Pelc: Distributed Computing with Advice: Information Sensitivity of Graph Coloring. In *34th Int. Colloquium on Automata, Languages and Programming (ICALP)*, Springer LNCS 4596, pp 231-242, 2007.

[16] P. Fraigniaud, A. Korman, and E. Lebhar. Local MST Computation with Short Advice. In *19th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, pp 154-160, 2007

[17] P. Fraigniaud, D. Ilcinkas, and A. Pelc. Oracle size: a new measure of difficulty for communication tasks. In *25th ACM Symp. on Principles of Distributed Computing (PODC)*, pp. 179-187, 2006.

[18] P. Fraigniaud, D. Ilcinkas, and A. Pelc. Tree Exploration with an Oracle. In *31st Int. Symp. on Mathematical Foundations of Computer Science (MFCS)*, LNCS 4162, Springer, pp. 24-37, 2006

[19] E. F. Grove. Online Bin Packing with Lookahead, In *proc. of 6th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pp. 430–436, 1995.

[20] E. G. Fusco and A. Pelc. Trade-offs Between the Size of Advice and Broadcasting Time in Trees. In *20th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, 2008.

[21] S. Irani, and S. S. Seiden, Randomized algorithms for metrical task systems., Theor. Comput. Sci., 194 (1998), pp. 163–182.

[22] A. Korman, and S. Kutten. Distributed verification of minimum spanning trees. In *25th ACM Symp. on Principles of Distributed Computing (PODC)*, pp 26-34, 2006.

[23] A. Korman, S. Kutten, and D. Peleg. Proof labeling schemes. In *24th ACM Symp. on Principles of Distributed Computing (PODC)*, pp 9-18, 2005.

[24] E. Koutsoupias and C.H. Papadimitriou. On the k-server conjecture. *J. ACM*, 42(5):971–983, 1995.

[25] M.S. Manasse, L.A. McGeoch, and D.D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990.

[26] N. Nisse and D. Soguet. Graph searching with advice. In *14th Int. Colloquium on Structural Information and Communication Complexity (SIROCCO)*, Springer LNCS 4474, pp 51-65, 2007.

[27] D. Peleg. Informative labeling schemes for graphs. *Theoretical Computer Science 340(3)*, pages 577-593, 2005.

[28] M.Thorup, and U. Zwick. Approximate distance oracles. In *33rd ACM Symp. on Theory of Computing (STOC)*, pp 183-192, 2001.