Online Computation with Advice

Yuval Emek[1,⋆], Pierre Fraigniaud[2,⋆⋆], Amos Korman[2,⋆⋆⋆], and Adi Rosén[3,†]

[1] Tel Aviv University, Tel Aviv, 69978 Israel
[2] CNRS and University Paris Diderot, France
[3] CNRS and University of Paris 11, France

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 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 $\Theta(1) \leq b \leq \log k$.

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

⋆ This work was partially done during this author's visit at LIAFA, CNRS and University Paris Diderot, supported by Action COST 295 DYNAMO. The author is also partially supported by The Israel Science Foundation, grants 664/05 and 221/07.

⋆⋆ Additional support from the ANR project ALADDIN, by the INRIA project GANG, and by COST Action 295 DYNAMO.

⋆⋆⋆ Additional support from the ANR project ALADDIN, by the INRIA project GANG, and by COST Action 295 DYNAMO.

† Research partially supported by ANR projects AlgoQP and ALADDIN.

S. Albers et al. (Eds.): ICALP 2009, Part I, LNCS 5555, pp. 427–438, 2009.

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 optimal algorithm: if the optimal 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 often infinite and in any case would typically be larger than the advice space $\mathcal{U}$, 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 classical and 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* $1 \leq b \leq \log n$*, 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* $\Theta(1) \leq b \leq \log k$*, there exists a deterministic online algorithm for the k-server problem that receives b bits of advice per request and whose competitive ratio is* $k^{O(1/b)}$.

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.

Organization. The rest of the paper is organized as follows. In Section 2 we give the necessary preliminaries. The lower bound for metrical task systems is presented in Section 3; the matching upper bound is established in Section 4. In Section 5 we prove Theorem 3 regarding the k-Server problem. Due to space limitations we only give outlines of these three results, while the proof of Theorem 4 is omitted entirely. 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} request is denoted by $\sigma[t] \in \mathcal{S}$. The online algorithm has to perform an action upon the receipt of each request, that is, at

[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.

round t, $1 \leq t \leq |\sigma|$, this action has to be performed when the online algorithm only knows the requests $\sigma[1], \ldots, \sigma[t]$. For this action it incurs some *cost* (in the case of minimization problems).

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 the competitive ratio of $\texttt{Alg}$, the advice complexity $\log |\mathcal{U}|$, and the interplay between these values.

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

[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.

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.

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 total transition cost in all rounds and the total processing cost 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, each server resides in some node $v \in V$. A subset $X \subseteq V$, $|X| = k$, where the servers reside is called a *configuration*. 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 request 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 total cost in all rounds.

3 A Lower Bound for MTS

In this section we sketch the proof of 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 a stronger model for the online algorithms, where the whole advice is provided at the beginning of the execution rather than round by round.

Before dwelling into the details of the lower bound, we describe a key ingredient in the proof that relates to a basic two-player zero-sum game, referred to as *generalized matching pennies (GMP)*. In GMP both the min-player and the max-player have the same discrete action space $\{1, \ldots, k\}$. The cost incurred by the min-player is 0 if both players play the same action $1 \leq i \leq k$; otherwise, it is 1. Let S be the random variable that takes on the action of the max-player (S reflects the mixed strategy of the max-player). Clearly, if $\mathrm{H}(S) = \log k$, namely, if the entropy of S is maximal (which means that the max-player chooses its action uniformly at random), then the expected cost incurred by the min-player

is $1 - 1/k$. The following lemma, whose proof is omitted from this extended abstract, provides a lower bound on the expected cost incurred by the min-player when the entropy in S is not maximal.

Lemma 1. *For every mixed strategy of the max-player, if* $\mathrm{H}(S) \geq \delta \log k$*, where* $0 < \delta < 1$*, then the expected cost incurred by the min-player is greater than* $\delta - 1/\log k$*.*

Given some integer $1 \leq b \leq \Theta(\log n)$ (the size of the advice per round), fix $\phi = 2^{\Theta(b)}$ and $\tau = \lfloor \log_\phi(n/2) \rfloor$. The hidden constants in the Theta notations are chosen to guarantee the following properties: (P1) $4 \leq \phi < \sqrt{n/2}$; and (P2) $1 < \tau = \Omega(\log(n)/b)$.

Let L be a sufficiently large integer. By repeating the GMP game with $k = \phi$ for $L\tau$ rounds, we obtain the following *online GMP problem*. Each round $1 \leq t \leq L\tau$ of the online GMP problem is characterized by some *letter* ℓ_t in the alphabet $[\phi]$. This letter (chosen by an oblivious adversary) is revealed immediately after round t. The alphabet $[\phi]$ also serves as the action space of the algorithms for the online GMP problem, that is, the action of an algorithm in round t is characterized by some letter $\ell'_t \in [\phi]$. The cost incurred by the algorithm in round t is 0 if $\ell'_t = \ell_t$ and 1 if $\ell'_t \neq \ell_t$. The online GMP problem is defined such that any algorithm incurs additional L units of *dummy cost* regardless of its choices of letters ℓ'_t. Clearly, an optimal (offline) algorithm for the online GMP problem does not incur any cost other than the dummy cost as its action in round t is $\ell'_t = \ell_t$ for every $1 \leq t \leq L\tau$.

The remainder of the proof consists of two parts. First, we employ an information theoretic argument to show that if the request sequence $\sigma \in [\phi]^{L\tau}$ of the online GMP problem is chosen uniformly at random among all $L\tau$-letter words over the alphabet $[\phi]$, then the expected cost incurred on σ by any deterministic online algorithm that receives $bL(\tau + 1)$ bits of advice in advance is $L(1 + \Omega(\tau))$. Therefore by Yao's principle, it follows that for every randomized online algorithm $\texttt{Alg}$ that receives $bL(\tau + 1)$ bits of advice in advance, there exists a request sequence $\sigma \in [\phi]^{L\tau}$ such that $\mathbb{E}[\texttt{Alg}(\sigma)] = L(1 + \Omega(\tau))$. Second, we reduce the online GMP problem to uniform n-node MTS showing that a request sequence of length $L\tau$ for the former problem can be implemented as a request sequence of length $L(\tau + 1)$ for the latter. By combining these two parts, and since $\texttt{Opt}(\sigma) = L$ for every $\sigma \in [\phi]^{L\tau}$, we conclude that the competitive ratio of any randomized algorithm for uniform n-node MTS with advice of b bits per round is $1 + \Omega(\tau) = 1 + \Omega(\log(n)/b)$ (even if the whole advice is provided to the algorithm at the beginning of the execution).

The information theoretic argument. Let $\sigma = (\sigma_1, \ldots, \sigma_{L\tau})$ be a sequence of $L\tau$ letters of the alphabet ϕ chosen independently and uniformly at random. Consider some deterministic online algorithm $\texttt{Alg}$ for the online GMP problem that receives $bL(\tau + 1) = \Theta(L\tau \log \phi)$ bits of advice, denoted by U, at the beginning of the execution.

The entropy in σ is $\mathrm{H}(\sigma) = L\tau \log \phi$. By definition, the advice U is a random variable which is fully determined by σ, thus $\mathrm{H}(\sigma \mid U) = \mathrm{H}(\sigma, U) - \mathrm{H}(U) =$

$\mathrm{H}(\sigma) - \mathrm{H}(U) = \Omega(L\tau \log \phi)$. A straightforward variant of the chain rule of conditional entropy implies that $\mathrm{H}(\sigma \mid U) = \mathrm{H}(\sigma_1 \mid U) + \mathrm{H}(\sigma_2 \mid \sigma_1, U) + \cdots + \mathrm{H}(\sigma_{L\tau} \mid \sigma_1, \ldots, \sigma_{L\tau-1}, U)$. Since $\mathrm{H}(\sigma_t) = \log \phi$ for all t, it follows by an averaging argument that in a constant fraction of the rounds t the entropy that remains in σ_t after `Alg` saw the advice U and the outcomes $\sigma_1, \ldots, \sigma_{t-1}$ of the previous rounds is $\Omega(\log \phi)$. Lemma 1 can now be used to deduce that the expected cost incurred by `Alg` in each such round t is $\Omega(1)$. Therefore the expected cost (including the dummy cost) incurred by `Alg` throughout the execution is $L(1 + \Omega(\tau))$.

The reduction. We now turn to reduce the online GMP problem to the MTS problem. The online GMP problem is implemented as an n-node MTS $(\mathcal{M}, \mathcal{R})$, where $\mathcal{R} = \{0, \infty\}^n$. That is, each task $r \in \mathcal{R}$ has 0 processing cost for some states and infinite processing cost for the rest. (It is assumed that in every task, at least one state has 0 processing cost.) Clearly, a competitive algorithm for $(\mathcal{M}, \mathcal{R})$ must have 0 total processing cost.

Given a request sequence of length $L\tau$ over the alphabet $[\phi]$ (for the online GMP problem), fix $n' = \phi^\tau$. The corresponding MTS request sequence is divided into L *cycles*, where each cycle consists of $\tau + 1$ rounds, so the total length of the request sequence is $L(\tau + 1)$. (Consequently, the advice provided to the online algorithm at the beginning of the execution contains $bL(\tau + 1)$ bits.) A request $r = \langle r(1), \ldots, r(n) \rangle$ in odd (respectively, even) cycles satisfies $r(i) = \infty$ for every $n'+1 \leq i \leq 2n'$ (resp., for every $1 \leq i \leq n'$). For states $2n'+1 \leq i \leq n$, we always have $r(i) = \infty$. Therefore, throughout an odd (respectively, even) cycle, every algorithm must be in state i for some $1 \leq i \leq n'$ (resp., for some $n'+1 \leq i \leq 2n'$). This means that between cycle c and cycle $c+1$ every algorithm must move from some state in $\{1, \ldots, n'\}$ to some state in $\{n' + 1, \ldots, 2n'\}$ or vice versa, which sums up to L units of cost referred to as the *dummy cost*.

In what follows we describe the structure of cycle c for some odd c. The structure of the even cycles is analogous. States $1, \ldots, n'$ are organized in contiguous *ranges*. For every round $1 \leq t \leq \tau + 1$ of cycle c, there exists some range $R_t \subseteq \{1, \ldots, n'\}$ such that the processing cost of state i is 0 if $i \in R_t$; and ∞ if $i \notin R_t$. Clearly, every competitive algorithm must process the request of round t in some state of R_t. In round 1 all states have zero processing cost, i.e., $R_1 = \{1, \ldots, n'\}$. For $t = 1, \ldots, \tau$, we have $R_{t+1} \subseteq R_t$. Specifically, the range R_t is partitioned into ϕ equally-sized contiguous *subranges*; range R_{t+1} will be one of these subranges (determined by the letter in $[\phi]$ that characterizes the corresponding round in the online GMP request sequence). Eventually, in the last round of the cycle, the range $R_{\tau+1}$ consists of a single state. This is consistent with our choice of parameters since $\tau = \log_\phi n'$.

Note that the only unknown in each round $1 \leq t \leq \tau$ of cycle c is which of the ϕ subranges of R_t corresponds to the range r_{t+1}. If at the end of round t the algorithm is located in some state of R_{t+1}, then no cost is incurred in round $t+1$; otherwise, a unit cost is incurred. So, we have $L\tau$ rounds characterized by some letter in $[\phi]$ and L additional rounds in which every algorithm incurs a unit cost. Therefore a ρ-competitive online algorithm for $(\mathcal{M}, \mathcal{R})$ implies a ρ-competitive online GMP algorithm.

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 $\lceil \frac{\lceil \log n \rceil}{b} \rceil = O(\log(n)/b)$. We call this algorithm `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, 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, and let OPT_i be the cost of the optimal algorithm during cycle i. Let OPT be the cost of the optimal algorithm on the whole request sequence.

Definition of `Follow`. Before starting to serve cycle i, $i \geq 0$, `Follow` places itself at state s_i. This is possible for 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. 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 serving 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.

We first give a lower bound on the cost of the optimal algorithm in each cycle (proof omitted), and then give the main theorem of this section.

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

Theorem 5. *`Follow` is $O(\alpha)$-competitive.*

Proof sketch: We consider the cost incurred by `Follow` cycle by cycle. We denote by C_i the cost of `Follow` during cycle i. $C_i = C_i^s + C_i^t + C_i^*$, 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). First note that by the triangle inequality $C_i^* \leq C_i^t + d(s_i, s_{i+1})$. Then, for each cycle we consider two cases.

Case 1: The last request of cycle i is in phase $k = 0$. In this case $C_i^s + C_i^t \leq OPT_i$ because `Follow` does not incur any transition cost, and the optimal algorithm either moved away from s_i, incurring a cost of at least 1, or serves the whole cycle in s_i, incurring processing cost equal to that of `Follow`.
Case 2: The last request of cycle i is in phase $k > 0$. In this case we have $C_i^s + C_i^t \leq 8\alpha \cdot OPT_i$. This is because (1) by Lemma 2 we have that $OPT_i \geq 2^{k-2}$; and (2) $C_i^s < 2^k$ by the definition of the phases, and $C_i^t \leq (\alpha - 1)2^{k+1}$ since `Follow` does not leave $B_i(k)$ during the phase.

Since $\sum_i d(s_i, s_{i+1}) \leq OPT$, summing over all cycles gives the result. □

5 An Upper Bound for the k-Server Problem

In this section we present a deterministic algorithm for the k-server problem that receives 4 bits of advice per request and admits a competitive ratio of $O(\sqrt{k})$, thus establishing Theorem 3.

The algorithm, denoted `Partition`, works in *iterations*, where each iteration consists of k requests. Fix some optimal (offline) algorithm `Opt` and let A_i denote the configuration of `Opt` at the beginning of iteration i. `Partition` uses *two* bits of advice per round of iteration i to identify A_{i+1} as follows. The first bit of advice, received in round $1 \leq j \leq k$ of iteration i, determines whether the node corresponding to the current request belongs to A_{i+1}. The second bit of advice, received in round $1 \leq j \leq k$ of iteration i, determines whether the j^{th} (according to some predefined order) node of A_i is still occupied in A_{i+1}. Based on these two bits of advice, and based on the knowledge of A_i, `Partition` computes the configuration A_{i+1} and moves to it at the end of iteration i. The cost incurred by this move is bounded from above by the sum of the total cost incurred by `Partition` in iteration i (tracing the steps of `Partition` back to configuration A_i) and the total cost incurred by `Opt` in iteration i (tracing the steps of `Opt` from A_i to A_{i+1}), thus increasing the overall competitive ratio of `Partition` only by a constant factor.

In order to serve the requests of iteration i, the k servers are partitioned into *heavy* servers and *light* servers according to the role they play in `Opt` during iteration i: those serving (in `Opt`) at least $\sqrt{k}$ requests are classified as heavy servers; the rest are classified as light servers. The third bit of advice, received in round $1 \leq j \leq k$ of iteration $i - 1$, determines whether the server that occupies the j^{th} node of configuration A_i is heavy or light. Consequently, at the beginning of iteration i `Partition` knows the heavy/light classification of all the servers. (The details related to the first iteration are omitted from this extended abstract.) The fourth bit of advice, received in round $1 \leq j \leq k$ of iteration i, determines whether `Opt` serves the current request with a heavy server or with a light one. Note that this partitions the requests of iteration i into *heavy* requests (served in `Opt` by a heavy server) and *light* requests (served in `Opt` by a light server).

To actually serve the requests of iteration i, `Partition` invokes the Work Function Algorithm (`WFA`) [9], with the heavy servers, on the subsequence consisting of the heavy requests of iteration i. `WFA` has a competitive ratio of $2k' - 1$, where k' is the number of servers in the problem [24]. Moreover, it is shown in

[11] that WFA is in fact strictly $O(k')$-competitive. In our case $k' \leq \sqrt{k}$ as there cannot be more than $\sqrt{k}$ heavy servers. Thus the cost incurred by Partition on the heavy requests of iteration i is at most $O(\sqrt{k})$ times larger than that of Opt.

The subsequence consisting of the light requests of iteration i is served by the light servers according to the following greedy strategy: each light request is served by the closest light server, which then immediately returns to its initial position in A_i. This strategy is strictly $O(\ell)$-competitive, where ℓ is the maximum number of requests served by any (light) server in Opt. By the definition of the light servers, ℓ in our case is at most $\sqrt{k}$, which implies that the cost incurred by Partition on the light requests of iteration i is at most $O(\sqrt{k})$ times larger than that of Opt. By summing over all iterations, we conclude that the competitive ratio of Partition is $O(\sqrt{k})$, thus establishing Theorem 3.

6 Conclusions

We define a model for online computation with advice. The advice provides the online algorithm with some (limited) information regarding 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. Albers, S.: A competitive analysis of the list update problem with lookahead. Theor. Comput. Sci. 197(1–2), 95–109 (1998)
2. Bartal, Y., Bollobás, B., Mendel, M.: Ramsey-type theorems for metric spaces with applications to online problems. J. Comput. Syst. Sci. 72, 890–921 (2006)
3. Bartal, Y., Mendel, M., Linial, N., Naor, A.: On metric Ramsey-type phenomena. Annals of Mathematics 162, 643–710 (2005)
4. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
5. Borodin, A., Irani, S., Raghavan, P., Schieber, B.: Competitive paging with locality of reference. In: STOC, pp. 249–259 (1991)
6. Borodin, A., Linial, N., Saks, M.E.: An optimal on-line algorithm for metrical task system. J. ACM 39, 745–763 (1992)

7. Breslauer, D.: On competitive on-line paging with lookahead. Theor. Comput. Sci. 209(1-2), 365–375 (1998)
8. Cohen, R., Fraigniaud, P., Ilcinkas, D., Korman, A., Peleg, D.: Label-guided graph exploration by a finite automaton. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 335–346. Springer, Heidelberg (2005)
9. Chrobak, M., Larmore, L.L.: 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, pp. 11–64 (1991)
10. Dobrev, S., Královič, R., Pardubská, D.: How much information about the future is needed? In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 247–258. Springer, Heidelberg (2008)
11. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: On the additive constant of the k-server work function algorithm. A manuscript, `http://arxiv.org/abs/0902.1378v1`
12. Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. J. Comput. Syst. Sci. 69, 485–497 (2004)
13. Fiat, A., Karlin, A.: Randomized and multipointer paging with locality of reference. In: STOC, pp. 626–634 (1995)
14. Fiat, A., Mendel, M.: Better algorithms for unfair metrical task systems and applications. SIAM J. Comput. 32, 1403–1422 (2003)
15. Fraigniaud, P., Gavoille, C., Ilcinkas, D., Pelc, A.: Distributed computing with advice: information sensitivity of graph coloring. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 231–242. Springer, Heidelberg (2007)
16. Fraigniaud, P., Korman, A., Lebhar, E.: Local MST computation with short advice. In: SPAA, pp. 154–160 (2007)
17. Fraigniaud, P., Ilcinkas, D., Pelc, A.: Oracle size: a new measure of difficulty for communication tasks. In: PODC, pp. 179–187 (2006)
18. Fraigniaud, P., Ilcinkas, D., Pelc, A.: Tree exploration with an oracle. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 24–37. Springer, Heidelberg (2006)
19. Grove, E.F.: Online bin packing with lookahead. In: SODA, pp. 430–436 (1995)
20. Fusco, E.G., Pelc, A.: Trade-offs between the size of advice and broadcasting time in trees. In: SPAA (2008)
21. Irani, S., Seiden, S.S.: Randomized algorithms for metrical task systems. Theor. Comput. Sci. 194, 163–182 (1998)
22. Korman, A., Kutten, S.: Distributed verification of minimum spanning trees. In: PODC, pp. 26–34 (2006)
23. Korman, A., Kutten, S., Peleg, D.: Proof labeling schemes. In: PODC, pp. 9–18 (2005)
24. Koutsoupias, E., Papadimitriou, C.H.: On the k-server conjecture. J. ACM 42(5), 971–983 (1995)
25. Manasse, M.S., McGeoch, L.A., Sleator, D.D.: Competitive algorithms for server problems. Journal of Algorithms 11, 208–230 (1990)
26. Nisse, N., Soguet, D.: Graph searching with advice. In: Prencipe, G., Zaks, S. (eds.) SIROCCO 2007. LNCS, vol. 4474, pp. 51–65. Springer, Heidelberg (2007)
27. Peleg, D.: Informative labeling schemes for graphs. Theor. Comput. Sci. 340(3), 577–593 (2005)
28. Thorup, M., Zwick, U.: Approximate distance oracles. In: STOC, pp. 183–192 (2001)