Online Metric Allocation and Time-Varying Regularization

Nikhil Bansal
University of Michigan, Ann Arbor, MI, USA

Christian Coester
University of Sheffield, UK

Abstract

We introduce a general online allocation problem that connects several of the most fundamental problems in online optimization. Let M be an n-point metric space. Consider a resource that can be allocated in arbitrary fractions to the points of M. At each time t, a convex monotone cost function $c_t\colon [0,1] \to \mathbb{R}_+$ appears at some point $r_t \in M$. In response, an algorithm may change the allocation of the resource, paying movement cost as determined by the metric and service cost $c_t(x_{r_t})$, where x_{r_t} is the fraction of the resource at r_t at the end of time t. For example, when the cost functions are $c_t(x) = \alpha x$, this is equivalent to randomized MTS, and when the cost functions are $c_t(x) = \infty \cdot \mathbb{1}_{x<1/k}$, this is equivalent to fractional k-server.

Because of an inherent *scale-freeness* property of the problem, existing techniques for MTS and k-server fail to achieve similar guarantees for metric allocation. To handle this, we consider a generalization of the online multiplicative update method where we decouple the rate at which a variable is updated from its value, resulting in interesting new dynamics. We use this to give an $O(\log n)$-competitive algorithm for weighted star metrics. We then show how this corresponds to an extension of the online mirror descent framework to a setting where the regularizer is time-varying. Using this perspective, we further refine the guarantees of our algorithm.

We also consider the case of non-convex cost functions. Using a simple ℓ_2^2-regularizer, we give tight bounds of $\Theta(n)$ on tree metrics, which imply deterministic and randomized competitive ratios of $O(n^2)$ and $O(n \log n)$ respectively on arbitrary metrics.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → K-server algorithms

Keywords and phrases Online algorithms, competitive analysis, k-server, metrical task systems, mirror descent, regularization

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.13

Related Version *Full Version*: https://arxiv.org/abs/2111.15169

Funding *Nikhil Bansal*: Supported in part by the NWO VICI grant 639.023.812.
Christian Coester: Supported in part by the Israel Academy of Sciences and Humanities & Council for Higher Education Excellence Fellowship Program for International Postdoctoral Researchers.

Acknowledgements We thank Ravi Kumar, Manish Purohit and Erik Vee for many useful discussions that inspired this work.

1 Introduction

We introduce a natural online problem that generalizes and is closely related to several fundamental and well-studied problems in online computation such as Metrical Task Systems (MTS), the k-server problem and convex body chasing. We call this the *metric allocation* problem (MAP) and it is defined as follows.

There is an underlying metric space M on n points with distances $d(i,j)$ between points i and j. An algorithm maintains an allocation of a resource to the points of M, represented by a vector $x = (x_1, \ldots, x_n)$ in the simplex $\Delta = \{x \in \mathbb{R}_+^M \mid \sum_{i \in M} x_i = 1\}$, where x_i denotes the

amount of resource at point i. At each time step t, tasks arrive at the points of M. The task at $i \in M$ is specified by a non-increasing and convex cost function $c_{t,i} : [0,1] \to \mathbb{R}_+$, which describes the cost of completing the task as function of resource available at i.[1] Given the tasks at time t, the algorithm can modify its previous allocation $x(t-1) \in \Delta$ to $x(t) \in \Delta$. It then incurs a *service cost* $c_t(x(t)) = \sum_{i \in M} c_{t,i}(x_i(t))$ and a *movement cost* of modifying $x(t-1)$ to $x(t)$ according to the distances in M (i.e., sending an ϵ amount of resource from i to j incurs cost $\epsilon \cdot d(i,j)$).

The problem is already very interesting when M is a uniform metric. In particular, this case already goes beyond the reach of existing techniques and highlights a key issue of *scale-freeness* (details in Section 1.2), which seems closely related to current barriers for improving bounds for k-server. For this reason, we will mostly focus on uniform metrics and, more generally, on weighted star metrics. Later, we also describe some results for non-convex cost functions on general metrics.

Besides the connections to other classical problems, that we describe below, MAP also has a natural motivation on its own. For example, the resource may represent workers that can be allocated to various locations. At step t, one could transfer extra workers to locations with high cost to execute tasks more efficiently. This also motivates our assumption on the cost functions being non-increasing (having more resources can only help) and convex (adding extra resources has diminishing returns). If M is a uniform or weighted star metric, this means there is a central depot that workers must return to between switching tasks.

Connections. MAP generalizes several fundamental and well-studied problems in online computation. We describe these next, as they play a key role in our discussion below.

- **Metrical Task Systems.** Here, there is a metric space (M,d) on n points, and the algorithm resides at some point in M at any time. At time t, a cost vector $\alpha_t \in \mathbb{R}_+^M$ arrives. The algorithm can then move from its old location $i_{t-1} \in M$ to a new $i_t \in M$, paying movement cost $d(i_{t-1}, i_t)$ and service cost α_{t,i_t}.

  The state of a *randomized* algorithm for MTS is given by a probability distribution $x(t) = (x_1(t), \ldots, x_n(t))$ on the n points. Its expected service cost at time t is given by $\sum_i \alpha_{t,i} \cdot x_i(t)$ and its expected movement cost is measured just like in MAP. Thus, randomized MTS is the special case of MAP with cost functions of the form $c_{t,i}(x_i) = \alpha_{t,i} x_i$, i.e., linear and increasing. (We show in the full version that MAP with non-decreasing cost functions is equivalent to MAP with non-increasing cost functions.)
- **k-server.** Here, there is a metric space (M,d) and k servers that reside at points of M. At time t, some point r_t is requested, which must be served by moving a server to r_t. The goal is to minimize the total movement cost. The *fractional* k-server problem is the relaxation[2] of the randomized k-server problem where points can have a fractional server mass. A request at r_t is served by having a server mass of at least 1 at r_t.

  Observe that fractional k-server is the special case of MAP with cost functions $c_{t,i} = 0$ for $i \neq r_t$ and $c_{t,r_t}(x_{r_t}) = \infty$ if $x_{r_t} < 1/k$ and 0 if $x_{r_t} \geq 1/k$, by viewing $k x_i(t)$ as the fractional server mass at location i at time t. Also notice that these $c_{t,i}$ are convex and non-increasing. If M is a uniform metric (or weighted star), this is equivalent to randomized (weighted) paging.

[1] Wlog, we can assume that a task appears at only one point r_t at time t, i.e., $c_{t,i} = 0$ for $i \neq r_t$. See Section 2.

[2] All known randomized k-server algorithms with poly-logarithmic competitive ratios use this relaxation.

- **Convex function chasing.** Here, the request at time t is a convex function $f_t : \mathbb{R}^n \to \mathbb{R}_+ \cup \{\infty\}$. The algorithm maintains a point in $\mathbb{R}^n$, and given f_t it can move from its old position $x(t-1) \in \mathbb{R}^n$ to a new $x(t) \in \mathbb{R}^n$, incurring cost $\|x(t) - x(t-1)\| + f_t(x(t))$.

  MAP is a special case of convex function chasing where the norm $\|\cdot\|$ is induced by a metric, cost functions are supported on the unit simplex and have separable form (i.e., $f_t(x) = \sum_{i=1}^n f_{ti}(x_i)$ with f_{ti} monotone).

The convex function chasing problem has seen tremendous progress recently, with the first competitive algorithm for arbitrary dimension given in [14], and $O(n)$-competitive algorithms shown in [27, 1]; this implies a trivial $O(n)$ upper bound for MAP with convex cost functions.

In recent years there has also been remarkable progress on obtaining polylogarithmic-competitive algorithms for special cases of MAP such as MTS [8, 26, 6, 7, 22, 3, 11, 19, 21] and the k-server problem [20, 4, 5, 2, 12, 25, 16, 24]. However, these solutions are based on rather ad hoc and problem-specific formulations (e.g., considering anti-server mass for k-server vs. server mass for MTS). Due to these inconsistencies, it is also still open, for example, whether for k-server one can achieve the same competitive ratios as for MTS.[3]

One of our key goals for studying MAP is to develop a systematic and unified approach for understanding a wide class of online problems.[4]

Our contribution. Below we list the results we obtain for MAP, and the new algorithmic design and analysis techniques that we develop. We give more details in Section 1.3.

- We give a tight $O(\log n)$-competitive algorithm for uniform metrics.
- This result already requires new algorithmic techniques to handle a *scale-freeness* property of the problem. In particular, our algorithm differs from classical multiplicative update algorithms by decoupling the rate at which a variable is updated from its value.
  The analysis also requires new ideas including a scale-mismatch potential function to handle the differences in the algorithm's perceived scale from the true scale.
- Next, we generalize the $O(\log n)$ bound above to weighted stars, and also refine this guarantee to be $(1+\epsilon)$-competitive with respect to the service cost.
- To achieve the refinement, we extend the online mirror descent framework pioneered by Bubeck et al. [12, 11] to a setting with a *time-varying regularizer*.
  The time-varying nature of the regularizer causes various complications for Bregman divergence based analysis techniques of prior works, and handling them requires several modifications.
- For the generalization of MAP where cost functions can be non-convex, we show an $\Omega(n)$ lower bound for arbitrary metrics. We give a matching $O(n)$ upper bound on tree metrics. This implies an $O(n^2)$ deterministic and $O(n \log n)$ randomized bound on general metrics.
- The $O(n)$ upper bound is also based on the mirror descent framework, but in contrast to all prior works in this framework, we do not use an entropic regularizer, but work with a *simple weighted ℓ_2^2-regularizer* instead.

[3] This contrasts with the deterministic setting, where the competitive ratio of MTS (which is $2n-1$) is known to be achievable for k-server (where $2k - 1 \le 2n - 1$ is known). Indeed, this is achieved by the same algorithm for both problems (the work function algorithm; see the book [9] for details) rather than by problem-specific algorithms as in the randomized setting.

[4] k-server on an n-point metric is also a special case of MTS on a $N = \binom{n}{k}$ point metric. But as the competitive ratio of MTS depends on N, this does not give any interesting bounds for k-server. In contrast, MAP generalizes both fractional k-server and randomized MTS in the *same* metric space.

We next discuss the relevance of allocation problems on star metrics and associated refined guarantees, and then describe the issue of scale-freeness that arises in MAP.

1.1 Allocation problems on star metrics and refined guarantees

Certain special cases of *allocation problems* on star metrics have been studied previously, either implicitly or explicitly, as they capture a lot of the difficulty of *general* metrics. This idea already goes back to Bartal et al. [7]; roughly, one can approximate a general metric space by a hierarchically-separated tree (HST), and recursively run the star algorithm at the internal nodes of the HST to decide how to much server mass to allocate to each child subtree. In this way, known algorithms for MTS on general metrics [7, 22, 11, 19] are obtained by using an algorithm for stars as central building blocks.

For k-server, a certain allocation problem on weighted stars was studied in [4] as a first step towards obtaining polylog(n,k)-competitive algorithms for k-server on general metrics. This allocation problem corresponds to the special case of MAP where the convex functions $c_{t,i}$ are piece-wise linear determined by values $c_{t,i}(j)$ at $j = 0, 1/k, 2/k, \ldots, 1$. In subsequent work [2], this step was completed to obtain the first polylog(n,k)-competitive algorithm for k-server on general metrics.

Refined guarantees. We say that an algorithm has α*-competitive service cost* and β*-competitive movement cost* if, up to some fixed additive constant, its service cost is at most α times the *total* (movement plus service) offline cost and its movement cost is at most β times the total offline cost.

For k-server and MTS, polylog-competitive algorithms on general metrics rely on star algorithms with $(1+\epsilon)$-competitive service cost and poly$(\log n, 1/\epsilon)$-competitive movement cost, for $\epsilon \approx 1/\log n$. The reason is that when the algorithm for stars is used recursively to obtain an algorithm on HSTs, then roughly, the service cost guarantee multiplies across levels and the movement cost guarantee increases additively. See, e.g., [7, 2] for more details.

The algorithm for the special case of MAP considered in [4] has $(1+\epsilon)$-competitive service cost and $O(\log k/\epsilon)$-competitive movement cost. However, the general cost functions that we consider for MAP (in this paper) correspond to this problem as $k \to \infty$; for this case, the $O(\log k/\epsilon)$ bound of [4] becomes unbounded and does not give anything useful.

1.2 Scale-freeness

The reason for the failure of the algorithm in [4] when $k \to \infty$ is not just technical, but an inherent one: roughly, for general cost functions, it is unclear how to do *multiplicative updates*, as there is no inherent notion of *scale* in the resulting problem. We elaborate on this issue now, as handling this *scale-freeness* is one of the key conceptual and technical contribution of this paper.

Multiplicative updates. A key underlying idea, sometimes used implicitly, for achieving poly-logarithmic guarantees for k-server, MTS and various other problems (e.g., those based on the online primal dual-framework [18, 17]) is that of multiplicative updates.

Let us see how this works for k-server and MTS on a star metric. For k-server, if a point r is requested, the fractional server amount z_i at other points i is decreased at rate proportional to $1 - z_i + \delta$ (i.e., the amount of server already missing at i plus a small constant δ). On the other hand for MTS, if a cost is incurred at point r, then the other points are increased at rate proportional to $x_i + \delta$.

Multiplicative update for MAP? As MAP generalizes these problems, clearly we also need to do some kind of multiplicative update. However, after some thought one soon realizes that completely unclear is "multiplicative update with respect to what?".

In particular, if we model k-server as MAP (as described above), then the update rule above becomes $x_i' \propto (1/k - x_i) + \delta$. This is natural as $1/k$ is a fixed parameter with a special meaning as $c_t(x) = \infty$ for $x_{r_t} < 1/k$ and 0 otherwise. On the other hand, for MTS the reason why $x_i' \propto (x_i + \delta)$ is natural is that the $c_{t,i}$ always have x-intercept at 0. In contrast, cost functions in MAP are lacking such an intrinsic *scale*.

We give a more concrete and instructive example to show the difficulty due to this lack of scale.

Example. We saw above how to model k-server via MAP by interpreting $z_i := kx_i$ as the amount of server mass at i, and a request to point i corresponds to the cost function $c_t(x) = \infty \cdot \mathbb{1}_{\{x_i < 1/k\}}$. However, this correspondence between the server mass z_i and the variable x_i is quite arbitrary.

A different way of modeling k-server via MAP is to choose any *offset* vector $a \in [0,1]^n$ with $s := 1 - \sum_i a_i > 0$, and interpret $z_i := k \cdot (x_i - a_i)/s$ as the server mass at i. Then, a request to page i corresponds to the cost function $c_t(x) = \infty \cdot \mathbb{1}_{\{x_i < a_i + s/k\}}$, and we can additionally intersperse cost functions $c_t(x) = \infty \cdot \mathbb{1}_{\{x_j < a_j\}}$ for each j to ensure that $z_j \geq 0$. In other words, an adversary can simulate a k-server request sequence in various *regions* of the simplex and at different *scales*.

Active region and active scale. Thus, the challenge for an algorithm is to find out the "active region" and "active scale". Since the adversary can keep changing this region and scale arbitrarily over time, any online algorithm for MAP needs to learn this region dynamically and determine how to do multiplicative updates with respect to the scale and offset of the current region.

At a higher level, the difficulty of learning an "active region" also relates to the difficulty in obtaining a polylog(k)-competitive algorithm for k-server on general metrics, which seems to require learning a region of poly(k) many points where the adversary is currently playing its strategy. A step in this direction was made recently in [12].

1.3 Results and techniques

We will first show the following tight bound for uniform metrics.

▶ **Theorem 1.** *There is an $O(\log n)$-competitive algorithm for MAP on uniform metrics.*

This bound is the best possible, due to the $\Omega(\log n)$ lower bound for the special case of randomized MTS [10]. Our algorithm is deterministic as randomization does not help for MAP.[5] The proof of Theorem 1 also extends to weighted stars (and we give a stronger result in Theorem 2 below). However, to introduce the key ideas in a modular way and avoid notational overhead, we focus on uniform metrics first.

To show Theorem 1, a key new idea is to handle the scale-freeness of MAP by decoupling the position x_i and its rate of change x_i' by using separate *rate* variables ρ_i for x_i'. The update of ρ_i is driven by trying to learn the active region and scale (details in Section 3).

[5] Any randomized algorithm for MAP can be derandomized by tracking its expected location. As cost functions are convex, this can only decrease the algorithm's cost.

Remark. Theorem 1 has an interesting consequence for the natural case of convex function chasing described above (with separable cost functions supported on the simplex and $\|\cdot\| = \|\cdot\|_1$). The competitive ratio of $O(\log n)$ improves exponentially on the $O(n)$ bound that follows from [1, 27] and breaks the $\Omega(\sqrt{n})$ lower bound that holds for the general case [23, 13].

The following theorem refines the previous guarantee via an improved algorithm for weighted stars.

▶ **Theorem 2.** *For any $\epsilon > 0$, there exists an algorithm for MAP on weighted stars with $(1+\epsilon)$-competitive service cost and $O\left(\frac{1}{\epsilon} + \log n\right)$-competitive movement cost.*

As explained above, a possible application of such refined guarantees is an extension to general metrics.

Time-varying regularization. To achieve the $(1+\epsilon)$-competitive service cost, we extend the powerful framework of *regularization* and *online mirror descent* to a setting where the regularizer is *time-varying*. This contrasts with previous works in this framework [15, 12, 11, 16, 19], which all used a *static* regularizer. Also as discussed in Section 1.1, a possible application of such refined guarantees is an extension to general metrics.

A time-varying regularizer is necessary as the regularizer must adapt to the current scale and region over time. This leads to substantial complications in the analysis. In particular, the default potential function for mirror descent analyses – the *Bregman divergence* – is not well-behaved when the offline algorithm moves (actually, it is not even well-defined), and changes of the regularizer lead to uncontrollable changes of the potential. We show how to adapt the Bregman divergence in several ways to obtain a modified potential function that has all the desired properties necessary to carry out the analysis.

Non-convex costs and arbitrary metric spaces. We also consider the version of MAP where the cost functions can be non-convex. Here, the competitive ratio must be exponentially worse.

▶ **Theorem 3.** *On any n-point metric space, any deterministic algorithm for MAP with non-convex cost functions has competitive ratio at least $\Omega(n)$.*

On tree metrics, we provide a matching upper bound:

▶ **Theorem 4.** *There is an $O(n)$-competitive deterministic algorithm for MAP on tree metrics, even if the cost functions are non-convex.*

By known tree embedding techniques, this implies the following result for general metrics:

▶ **Corollary 5.** *There is an $O(n^2)$-competitive deterministic and $O(n \log n)$-competitive randomized algorithm for MAP on arbitrary metric spaces, even for non-convex cost functions.*

ℓ_2^2-regularization. Our algorithm achieving the tight guarantee on trees is also based on mirror descent, but again with a crucial difference to previous mirror-descent based online algorithms in the literature. While previous algorithms all used some version of an entropic regularizer, our regularizer is a weighted ℓ_2^2-norm. Here, again, the Bregman divergence is not suitable as a potential function, but the issues are more fundamentally rooted in the non-convex structure of cost functions, and addressing them with changes to the Bregman divergence seems unlikely to work. Instead, our analysis uses two different potential functions, one of which resembles ideas of "weighted depth potentials" used in [12, 11] and the other one is a kind of "one-sided matching".

1.4 Organization

In Section 2, we define an equivalent version of MAP that will be easier to work with. We will give a first algorithm for uniform metrics in Section 3, where we also describe the ideas to overcome scale-freeness. In Section 4, we discuss a modified algorithm for weighted stars via mirror descent with a time-varying regularizer. However due to space constraints, most of the details are only given in the full version. Our upper and lower bounds for non-convex cost functions on general metrics are proved in the full version, which includes in particular the algorithm based on ℓ_2^2-regularization.

2 Preliminaries

For $a \in \mathbb{R}$, we write $[a]_+ := \max\{a, 0\}$. A metric space M is called a *weighted star* if there are weights $w_i > 0$ for $i \in M$ and the distance between two points $i \neq j$ is given by $d(i, j) = w_i + w_j$.

Continuous-time model and simplified cost functions. It will be more convenient to work with the following continuous-time version of MAP. Instead of cost functions being revealed at discrete times $t = 1, 2, \ldots$, we think of cost functions c_t arriving continuously over time, and that c_t changes only finitely many times. At any time $t \in [0, \infty)$, the algorithm maintains a point $x(t) \in \Delta$, where $\Delta := \{x \in \mathbb{R}_+^M : \sum_{i \in M} x_i = 1\}$, and the dynamics of the algorithm is specified by the derivative $x'(t)$ at each time t. The movement cost and the service cost are given by $\int_0^\infty \|x'(t)\| dt$ and $\int_0^\infty c_t(x(t)) dt$ respectively, where the norm $\|\cdot\|$ is induced by the metric.[6] On a weighted star metric, the norm $\|\cdot\|$ is given by $\|z\| := \sum_i w_i |z_i|$.

Further, we assume that $c_{t,i}$ is non-zero for only a single location $r_t \in M$ at any time, and $c_{t,i}$ is linear with slope -1 and truncated at 0 (see Figure 1(a)). Formally, $c_t(x) = [s_t - x_{r_t}]_+$ for some $s_t \in [0, 1]$ and $r_t \in M$. A useful consequence of this view is that if the service cost $c_t(x(t))$ incurred by the online algorithm is α_t, then the (one-dimensional) cost function c_{t,r_t} intercepts the x-axis (becomes 0) at the point $x_{r_t}(t) + \alpha_t$. The cost of an offline algorithm at $y \in \Delta$ is then given by $[\alpha_t + x_{r_t}(t) - y_{r_t}]_+$. For a cost function c_t of this form, we will say that $x(t)$ *is charged cost* α_t *at* r_t.

To simplify the description and analysis of our algorithms, we will further allow them to decrease x_i to a negative value. In the full version, we show that these assumptions are all without loss of generality.

3 A first algorithm for uniform metrics

We describe here an $O(\log n)$-competitive algorithm for MAP on uniform metrics, proving Theorem 1. Although the algorithm also extends to weighted stars, we assume in this section that all weights are 1 to avoid technicalities and focus on the key ideas.

3.1 Overview

Before we state the formal algorithm, we first give an overview and intuition behind the ideas needed to handle the difficulties due to scale-freeness.

Fix a time t, and let $x = x(t)$ and $y = y(t)$ denote the online and offline position and suppose a cost of $\alpha = \alpha_t$ is received (charged) at point $r = r_t$. We drop t from now for notational ease, as everything is a function of t. Clearly, an algorithm that wishes to be

[6] In general, $\|z\| = \min_f \sum_{i,j \in M} f(i, j) d(i, j)$, where the minimum is taken over all flows $f : M \times M \to \mathbb{R}_+$ satisfying $z_i = \sum_{j \in M} (f(j, i) - f(i, j))$ for all $i \in M$.

competitive must necessarily increase x_r (otherwise the offline algorithm can move to some y with $y_r > x_r$ and keep giving such cost functions forever). So, the key question is how to decrease other coordinates x_i for $i \neq r$ to offset the increase of x_r (and maintain $\sum_i x_i = 1$).

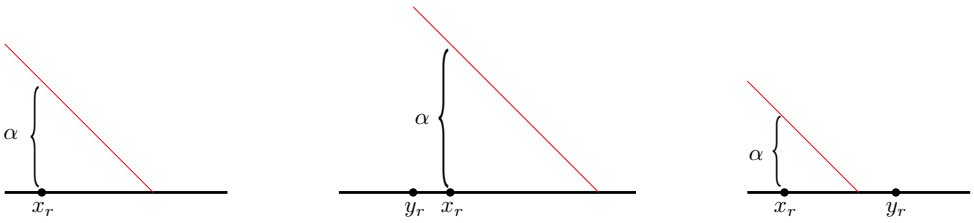
Separate rate variables and how to update them. As discussed in Section 1.2, due to scale-freeness, the rate of update of x_i, denoted x_i', cannot simply be some function of x_i, as in standard multiplicative update algorithms. So we maintain separate *rate* variables ρ_i (decoupled from x_i) that specify the rate at which to reduce x_i, i.e., $x_i' = -\rho_i$ (plus a small additive term and suitably normalized, but we ignore this technicality for now). Now the key issue becomes how to update these ρ_i variables themselves?

Consider the following two scenarios, which suggest two conflicting updates to ρ_r, depending upon the offline location y_r.

(i) $x_r < y_r$. Here, the adversary can make us incur the service cost while possibly not paying anything itself. However this is not problematic, as we will increase x_r and hence get closer to y_r. Also, decreasing ρ_r is good as it will prevent us in future from decreasing x_r again too fast and move away from y_r when requests arrive at locations $i \neq r$.

(ii) $y_r < x_r$. Here, increasing x_r is fine, as even though we are moving *away* from y_r, the offline algorithm is paying a higher service cost than online. However, decreasing ρ_r is very bad, as this makes it much harder for online to catch up with the offline position y_r later.

To summarize, in case (i), we should decrease ρ_i and in case (ii), we should leave it unchanged or increase it. However, the algorithm does not know the offline location y_r, and hence which option to choose.



(a) Linear Cost Function with slope -1

(b) $y_r < x_r$; adversary prefers large α

(c) $y_r > x_r$: adversary prefers small α

Figure 1 Illustration of cost functions.

Indirectly estimating y_r. We note that even though the algorithm does not know y_r, it can reasonably estimate whether $y_r < x_r$ or not, by looking at the structure of requests from the adversary's point of view. Suppose $y_r < x_r$ (see Figure 1(b)). In this case, the adversary will want to give us requests with *large* α; because for small α, the offline algorithm pays much higher cost in proportion to that of online; indeed, as α gets larger, the ratio between the offline to online service cost tends to 1. On the other hand, if $y_r > x_r$ (see Figure 1(c)), the adversary will tend to give requests so that $x_r + \alpha \leq y_r$ (so offline incurs no service cost at all), and hence keep α *small*.

A problem however is that even though the online algorithm sees α when the request arrives, whether this α is large or small has no intrinsic meaning as this depends on the current scale at which the adversary is giving the instance. The final piece to make this idea work is that the algorithm will also try to learn the scale at which the adversary is playing its strategy. We describe this next.

A region estimate. At each time t, and for each point i, we maintain a real number $b_i \geq 0$ where $b_i \geq x_i$. Intuitively, we can view b_i as an (online) estimate of the "region" where the adversary is playing the strategy, and we set the rate variable $\rho_i = b_i - x_i$. The observations above now suggest the following algorithm. For an incoming request at point r,

(i) if $x_r + \alpha \leq b_r$ (this corresponds to small α in the discussion above, which suggested that $y_r > x_r$) the algorithm increases x_r and decreases b_r at roughly the same rate (and hence decreases $\rho_r = b_r - x_r$ at roughly twice the rate), and

(ii) if $x_r + \alpha > b_r$ (this corresponds to α being large, which suggested that $y_r < x_r$), the algorithm increases both b_r and x_r at roughly the same rate (and ρ_r only increases slowly). Note that even though the location r is incurring a service cost, we do not necessarily decrease ρ_r.

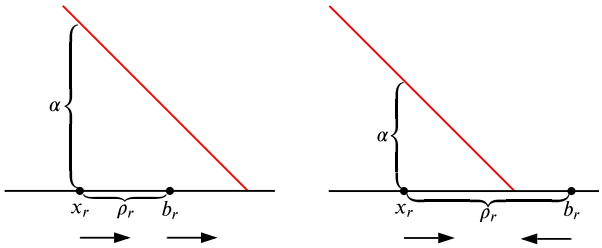


Figure 2 Illustration of the update rule for ρ_r. If $\alpha > \rho_r$ (left), then x_r and b_r increase and ρ_r changes slowly. If $\alpha < \rho_r$ (right), then x_r and b_r move towards each other and ρ_r decreases.

For other points $i \neq r$, b_i stays fixed. So as x_i decreases, this corresponds to increasing the rate ρ_i. This step is analogous to multiplicative updates, but where the update rate is given by distance of x_i from b_i, where b_i itself might change over time. We also call b_i the "baseline".

A complication (in the analysis) will be that the b_i themselves are only estimates and could be wrong. E.g., even if b_i is accurate at some given time, the offline algorithm can move y_i somewhere far at the next step, and start issuing requests in that region. The current b_i would be completely off now and the algorithm may make wrong moves. What will help here in the analysis is that the algorithm is quickly trying learn the new b_i.

3.2 Formal description of the algorithm

At any instantaneous time t, the algorithm maintains a point $x(t) \in \Delta$. In addition, it also maintains a point $b(t) \in \mathbb{R}^M$, where $b_i(t) \geq x_i(t)$ for all $i \in M$. Let $\rho_i(t) = b_i(t) - x_i(t)$, for each $i \in M$. We specify the formal algorithm by describing how it updates the points $x(t)$ and $b(t)$ in response to a cost function c_t. Again, we drop t from the notation hereafter.

Suppose the cost function at a given time charges cost α at point r. Then, we increase x_r at rate α and simultaneously decrease all x_i (including x_r) at rate

$$-\alpha \cdot \frac{\rho_i + \delta S}{2S},$$

where $\delta = 1/n$ and $S := \sum_i \rho_i$. Intuitively, one can think of S as the current *scale* of the problem (which is changing over time).

Then the overall update of x can be summarized as

$$x_i' = \alpha \left(\mathbb{1}_{i=r} - \frac{\rho_i + \delta S}{2S} \right) \qquad \text{for all } i \in M. \tag{1}$$

The baseline vector b is updated as follows:

(i) For $i \neq r$, b_i stays fixed.

(ii) For $i = r$, if $x_r + \alpha > b_r$ (or equivalently $\alpha > \rho_r$), then $b'_r = \alpha$.

(iii) For $i = r$, if $x_r + \alpha \leq b_r$ (or equivalently $\alpha \leq \rho_r$), then $b'_r = -\alpha$. [7]

See Figure 2 for an illustration. Notice that the update rules for x'_i and b'_i also ensure that $b_i \geq x_i$. Finally, using that $\rho_i = b_i - x_i$ and writing compactly, this gives the following update rule for ρ_i.

$$\rho'_i = \alpha \left(\frac{\rho_i + \delta S}{2S} - 2 \cdot \mathbb{1}_{i=r \text{ and } \rho_r > \alpha} \right). \tag{2}$$

This completes the description of the algorithm.

Feasibility. Notice that $\sum_i (\rho_i + \delta S)/(2S) = 1$ and hence $\sum_i x'_i = 0$ and the update for x maintains that $\sum_i x_i = 1$. In the algorithm description above, we do not explicitly enforce that $x_i \geq 0$. As we show in the full version, allowing the online algorithm to decrease x_i to negative values is without loss of generality. It is possible to enforce this directly in the algorithm, but this would make the notation more cumbersome.

3.3 Analysis sketch

Due to space constraints, and because our proof of Theorem 2 yields a stronger result anyway, we only provide a brief sketch of the analysis of our algorithm here. It is based on potential functions. Specifically, we define a (bounded) potential Θ that is a function of the online and offline states, and show that at any time t it satisfies

$$\mathrm{On}' + \Theta' \leq O(\log n) \cdot \mathrm{Off}', \tag{3}$$

where On′ (resp. Off′) denote the change in cost of the online (resp. offline) algorithm, and Θ' is the change in the potential. The potential function Θ consists of two parts defined as

$$\text{(Primary potential)} \qquad P := \sum_{i:\, x_i \geq y_i} (b_i - y_i) \log \frac{(1+\delta)(b_i - y_i)}{\rho_i + \delta(b_i - y_i)}$$

$$\text{(Scale-mismatch potential)} \qquad Q := \sum_i [\rho_i + 2(x_i - y_i)]_+ ,$$

where $y \in \Delta$ is the position of the offline algorithm. The overall potential is given by

$$\Theta = 12P + 6Q.$$

One can verify, by taking derivative with respect to y_i and using $\delta = 1/n$ and $\rho_i = b_i - x_i \geq 0$, that Θ is $O(\log n)$-Lipschitz in y_i. Thus, the potential increases by at most $O(\log n)$ times the offline movement cost when y changes. It therefore suffices to show (3) for the case that y stays fixed and only the online algorithm moves (i.e., while x, ρ and b are changing).

Recall that $S = \sum_i \rho_i$ is the algorithm's estimate of the current "scale", and define $L := \sum_i |x_i - y_i|$, which we may think of as the true scale of the error between the online position x and the (unknown) offline position y. If the current estimate of the scale is accurate, one would expect $S \approx L$, but in general this need not be true. In the full version, we prove the following two lemmas:

[7] Strictly speaking, if $x_r + \alpha = b_r$ both rules (ii) and (iii) apply simultaneously and b_r stays fixed.

▶ **Lemma 6** (Change of the primary potential). *When y is fixed and the online algorithm moves, the change of P is bounded by*

$$P' \le O(\log n)[\alpha + x_r - y_r]_+ - (\alpha/2)\min\{L/2S, 1\}.$$

▶ **Lemma 7** (Change of the scale-mismatch potential). *When y is fixed and the online algorithm moves, the change of Q is bounded by*

$$Q' \le 2\alpha \cdot \mathbb{1}_{y_r \le x_r + \alpha/2} - (\alpha/2)\,[1 - L/S]_+.$$

Note that the offline algorithm incurs service cost at rate $[\alpha + x_r - y_r]_+$, and the online algorithm incurs both service and movement cost at rate $O(\alpha)$. In the bound on P' in Lemma 6, the positive term can be charged against the offline service cost. If $S = O(L)$, then the negative term can be used to pay for the online cost. However, if $S \gg L$, then the negative term may be negligibly small. Intuitively, this corresponds to the case that the algorithm's estimate of the scale is far off from the true scale, and this possibility is the reason why we need the scale-mismatch potential.

If $y_r \le x_r + \alpha/2$, the offline service cost is at least $\alpha/2$, so the positive part in the change of Q is at most 4 times the offline service cost. The negative part in Lemma 7 pays for the online cost if $S \gg L$, which is precisely the case not covered by the primary potential. So,

$$\begin{aligned}\Theta' = 12P' + 6Q' &\le O(\log n)\cdot \mathrm{Off}' - 3\alpha\,(\min\{L/S, 2\} + [1 - L/S]_+)\\ &\le O(\log n)\cdot \mathrm{Off}' - 3\alpha,\end{aligned}$$

which yields the desired inequality (3) because the online service cost is α and online movement cost is at most 2α.

4 Time-varying regularization and refined guarantees on weighted stars

We now turn to an improved algorithm for weighted stars, achieving the refined guarantees of Theorem 2 that the service cost is $(1+\epsilon)$-competitive. To do so, we first reinterpret our previous algorithm through the lens of mirror descent.

4.1 Online mirror descent

The online mirror descent framework has been useful to derive optimally competitive algorithms for problems where the state of an algorithm can be described by a point in a convex body (e.g., set cover, k-server, MTS [15, 12, 11, 16, 19]). That is, the algorithm can be described by a path $x\colon [0,\infty) \to K$ for a convex body $K \subset \mathbb{R}^n$, such that $x(t)$ describes the state of the algorithm at time t (in our case, $K = \Delta$ is the simplex). In the framework, the dynamics of an algorithm x is specified by a differential equation of the form

$$\nabla^2\Phi(x(t))\cdot x'(t) = f(t) - \lambda(t) \tag{4}$$

where $\Phi\colon K \to \mathbb{R}$ is a suitable convex function called the *regularizer*, $\nabla^2\Phi(x(t))$ is its Hessian at $x(t)$, $f\colon [0,\infty) \to \mathbb{R}^n$ is called a *control function*, and $\lambda(t)$ is an element of the normal cone of K at $x(t)$, given by

$$N_K(x(t)) := \{\lambda \in \mathbb{R}^n\colon \langle \lambda, y - x(t)\rangle \le 0,\ \forall y \in K\}.$$

Under suitable conditions (which are all satisfied here), the path x is uniquely defined by (4) and absolutely continuous in t [12]. The equation (4) is easiest to read if we imagine that $\nabla^2\Phi(x(t))$ were the identity matrix. Then (4) says that x tries to move in direction $f(t)$, and the normal cone element λ ensures that $x(t)$ does not leave the body K. The case when $\nabla^2\Phi(x(t))$ is different from the identity matrix corresponds to imposing a different (e.g., non-Euclidean) geometry on K. When Φ is fixed, the framework allows a black-box way to prove 1-competitive service cost using the Bregman divergence $D_\Phi(y\|x) := \Phi(y) - \Phi(x) + \langle \nabla\Phi(x), x - y\rangle$ as a potential function.

We show in the full version that the dynamics (1) of x defined in the previous section is precisely equivalent to equation (4) when choosing the *time-varying* regularizer

$$\Phi_t(x) := \sum_i (b_i(t) - x_i + \delta S(t)) \log\left(\frac{b_i(t) - x_i}{S(t)} + \delta\right) \tag{5}$$

for $S(t) := \sum_i b_i(t) - 1$, and the control function

$$f(t) := \frac{\alpha_t}{b_{r_t}(t) - x_{r_t}(t) + \delta S(t)} e_{r_t},$$

where e_{r_t} denotes the 0-1-vector with a 1 only in the r_t-coordinate, and α_t is the cost charged at r_t at time t.

To achieve the refined guarantees, we replace α_t by $b_{r_t}(t) - x_{r_t}(t)$ in the control function, and use a very similar regularizer that incorporates a scaling factor and the weights of the star. The most subtle parts in the proof of Theorem 2 are the way the baseline vector b_i is updated (which differs from how it was done in the previous section) and several modifications to the Bregman divergence in order to handle the time-varying nature of Φ_t. Due to space constraints, we defer all details to the full version.

References

1. C. J. Argue, Anupam Gupta, Guru Guruganesh, and Ziye Tang. Chasing convex bodies with linear competitive ratio (invited paper). In *STOC '21*, 2021. doi:10.1145/3406325.3465354.
2. Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. *J. ACM*, 62(5), 2015. doi:10.1145/2783434.
3. Nikhil Bansal, Niv Buchbinder, and Joseph Naor. Metrical task systems and the k-server problem on HSTs. In *ICALP' 10*, 2010. doi:10.1007/978-3-642-14165-2_25.
4. Nikhil Bansal, Niv Buchbinder, and Joseph Naor. Towards the randomized k-server conjecture: A primal-dual approach. In *Symposium on Discrete Algorithms, SODA*, pages 40–55, 2010.
5. Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. *J. ACM*, 59(4):19:1–19:24, 2012.
6. Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *FOCS '96*, 1996. doi:10.1109/SFCS.1996.548477.
7. Yair Bartal, Avrim Blum, Carl Burch, and Andrew Tomkins. A polylog(n)-competitive algorithm for metrical task systems. In *STOC '97*, 1997. doi:10.1145/258533.258667.
8. Avrim Blum, Howard J. Karloff, Yuval Rabani, and Michael E. Saks. A decomposition theorem for task systems and bounds for randomized server problems. *SIAM J. Comput.*, 30(5), 2000. doi:10.1137/S0097539799351882.
9. Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.
10. Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4), 1992. doi:10.1145/146585.146588.

11 Sébastien Bubeck, Michael B. Cohen, James R. Lee, and Yin Tat Lee. Metrical task systems on trees via mirror descent and unfair gluing. In *SODA '19*, 2019. doi:10.1137/1.9781611975482.6.

12 Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. k-server via multiscale entropic regularization. In *STOC '18*, 2018. doi:10.1145/3188745.3188798.

13 Sébastien Bubeck, Bo'az Klartag, Yin Tat Lee, Yuanzhi Li, and Mark Sellke. Chasing nested convex bodies nearly optimally. In *SODA '20*, 2020. doi:10.1137/1.9781611975994.91.

14 Sébastien Bubeck, Yin Tat Lee, Yuanzhi Li, and Mark Sellke. Competitively chasing convex bodies. In *STOC '19*, 2019. doi:10.1145/3313276.3316314.

15 Niv Buchbinder, Shahar Chen, and Joseph (Seffi) Naor. Competitive analysis via regularization. In *Symposium on Discrete Algorithms, SODA*, pages 436–444, 2014.

16 Niv Buchbinder, Anupam Gupta, Marco Molinaro, and Joseph (Seffi) Naor. k-servers with a smile: Online algorithms via projections. In *Symposium on Discrete Algorithms, SODA*, pages 98–116, 2019.

17 Niv Buchbinder and Joseph Naor. The design of competitive online algorithms via a primal-dual approach. *Found. Trends Theor. Comput. Sci.*, 3(2-3):93–263, 2009.

18 Niv Buchbinder and Joseph Naor. Online primal-dual algorithms for covering and packing. *Math. Oper. Res.*, 34(2):270–286, 2009.

19 Christian Coester and James R. Lee. Pure entropic regularization for metrical task systems. In *COLT '19*, 2019. URL: http://proceedings.mlr.press/v99/coester19a.html.

20 Aaron Cote, Adam Meyerson, and Laura J. Poplawski. Randomized k-server on hierarchical binary trees. In *STOC '08*, 2008. doi:10.1145/1374376.1374411.

21 Farzam Ebrahimnejad and James R. Lee. Multiscale entropic regularization for MTS on general metric spaces. In *ITCS '22*, 2022.

22 Amos Fiat and Manor Mendel. Better algorithms for unfair metrical task systems and applications. *SIAM J. Comput.*, 32(6), 2003. doi:10.1137/S0097539700376159.

23 Joel Friedman and Nathan Linial. On convex body chasing. *Discret. Comput. Geom.*, 9, 1993. doi:10.1007/BF02189324.

24 Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. A hitting set relaxation for k-server and an extension to time-windows. In *FOCS '21*, 2021.

25 James R. Lee. Fusible HSTs and the randomized k-server conjecture. In *FOCS '18*, pages 438–449, 2018. doi:10.1109/FOCS.2018.00049.

26 Steve Seiden. Unfair problems and randomized algorithms for metrical task systems. *Inf. Comput.*, 148(2), 1999. doi:10.1006/inco.1998.2744.

27 Mark Sellke. Chasing convex bodies optimally. In Shuchi Chawla, editor, *SODA '20*, 2020. doi:10.1137/1.9781611975994.92.