The Infinite Server Problem

CHRISTIAN COESTER, CWI, Amsterdam, The Netherlands
ELIAS KOUTSOUPIAS, University of Oxford, United Kingdom
PHILIP LAZOS, Sapienza University of Rome, Italy

We study a variant of the k-server problem, the infinite server problem, in which infinitely many servers reside initially at a particular point of the metric space and serve a sequence of requests. In the framework of competitive analysis, we show a surprisingly tight connection between this problem and the resource augmentation version of the k-server problem, also known as the (h, k)-server problem, in which an online algorithm with k servers competes against an offline algorithm with h servers. Specifically, we show that the infinite server problem has bounded competitive ratio if and only if the (h, k)-server problem has bounded competitive ratio for some $k = O(h)$. We give a lower bound of 3.146 for the competitive ratio of the infinite server problem, which holds even for the line and some simple weighted stars. It implies the same lower bound for the (h, k)-server problem on the line, even when $k/h \to \infty$, improving on the previous known bounds of 2 for the line and 2.4 for general metrics. For weighted trees and layered graphs, we obtain upper bounds, although they depend on the depth. Of particular interest is the infinite server problem on the line, which we show to be equivalent to the seemingly easier case in which all requests are in a fixed bounded interval. This is a special case of a more general reduction from arbitrary metric spaces to bounded subspaces. Unfortunately, classical approaches (double coverage and generalizations, work function algorithm, balancing algorithms) fail even for this special case.

CCS Concepts: • **Theory of computation → K-server algorithms;**

Additional Key Words and Phrases: Online algorithms, competitive analysis, k-server, resource augmentation

ACM Reference format:
Christian Coester, Elias Koutsoupias, and Philip Lazos. 2021. The Infinite Server Problem. *ACM Trans. Algorithms* 17, 3, Article 20 (July 2021), 23 pages.
https://doi.org/10.1145/3456632

1 INTRODUCTION

In the k-server problem, one of the most central and well-studied online problems [21, 25], k servers serve a sequence of requests. The servers reside at points of a metric space M and requests are simply points of M. Serving a request entails moving one of the servers to the request. The objective is to minimize the total distance traveled by the servers. The most interesting variant of

A preliminary version of this work was first presented in Reference [14].
Partially supported by ERC Advanced Grants 321171 (ALGAME) and 788893 (AMDROMA), EPSRC awards 1493310 and 1652110, and MIUR PRIN project ALGADIMAR.
Authors' addresses: C. Coester, CWI, P.O. Box 94079, 1090 GB Amsterdam, Netherlands; email: christian.coester@cwi.nl; E. Koutsoupias, Department of Computer Science, 15 Parks Road, Oxford OX1 3QD, United Kingdom; email: elias@cs.ox.ac.uk; P. Lazos, Sapienza University of Rome, Via Cavour 124, 00184, Rome, Italy; email: lazos@diag.uniroma1.it.

1549-6325/2021/07-ART20 $15.00
https://doi.org/10.1145/3456632

the problem is its online version, in which the requests appear one by one and the online algorithm must decide how to serve a request without knowing the future requests. It is known that the deterministic k-server problem has competitive ratio between k and $2k - 1$ for every metric space with at least $k + 1$ points [23, 25].

We study the *infinite server problem* (hereafter written ∞-*server problem*),[1] the variant of the k-server problem where $k = \infty$. Of course, this problem would be trivial if each point of the metric space was already occupied by a server in the initial configuration. To avoid this, we assume that all infinitely many servers initially reside at the same point, the *source*. At first glance it may appear that the lower bound of k for the k-server problem would imply an unbounded competitive ratio for the ∞-server problem. But consider, for example, a uniform metric space (i.e., the distance between any two points is 1), and observe that the ∞-server problem has competitive ratio 1 for this case.

The ∞-server problem is closely related to the (h, k)-server problem, the resource augmentation version of the k-server problem in which the online algorithm has k servers and competes against an offline algorithm with $h \leq k$ servers. This refinement of the k-server problem is also known as the *weak adversaries* model [3, 20]. One major open problem in competitive analysis is whether the (h, k)-server problem has bounded competitive ratio when $k \gg h$. Here, we show a, perhaps surprising, tight connection between the ∞-server problem and the (h, k)-server problem, which also allows us to improve lower bounds for the latter.

The ∞-server problem is also a considerable generalization of the ski-rental problem, since the ski-rental problem is essentially a special case of the ∞-server problem when the metric space is an isosceles triangle (with the source at the more remote vertex of the triangle). Conversely, we show that the general ∞-server problem for arbitrary metric spaces can be reduced to a "ski-rental version" in which all requests are in a bounded-diameter metric space and the source is in fixed distance from every other point.

Besides its theoretical appeal, our main reasons for investigating the ∞-server problem are the following: The competitive ratio of the k-server problem goes to infinity as $k \to \infty$, but for $k = \infty$ it goes back to a small constant at least on some metric spaces (even on some infinite ones). Thus, the high competitive ratio of the k-server problem can be misleading for situations where the number of servers is so high that it is not a limitation in practice. A study of the ∞-server problem can provide a better understanding of these cases. Moreover, the ∞-server problem allows to model applications where more servers can be bought. A price for buying new servers can be modeled easily by appropriate placement of the source in the metric space. Finally, the relationship between the ∞-server problem and the (h, k)-server problem allows for new ways to tackle the latter.

1.1 Previous Work

The k-server problem was first formulated by Manasse et al. [25] to generalize a variety of online settings whose stepwise cost had a "metric"-like structure. They built on previous work by Sleator and Tarjan [26], the genesis of competitive analysis, on the paging problem. This problem can be easily recast as a k-server instance for the uniform metric and was already known to be k-competitive.

Manasse et al. [25] also showed that the deterministic competitive ratio of the k-server problem is at least k on any metric space with more than k points. They then proposed the renowned k-server conjecture, stating that this bound is tight. This has been shown to be true for $k = 2$ [25] and for several special metric spaces [11, 12, 22, 25, 26]. A stream of refinements [5, 18] led to

[1] We first learned about this problem from Kamal Jain.

better competitive ratios for general metric spaces until Reference [23] showed that a competitive ratio of $2k - 1$ can be achieved on any metric space. For randomized algorithms, the folklore *randomized k-server conjecture* states that the competitive ratio is $\theta(\log k)$ on general metric spaces. There has been tremendous progress on this question very recently, leading to polylogarithmically competitive algorithms for general metric spaces [1, 9, 24]. Chasing the competitive ratio for the k-server problem has been pivotal for the development of competitive analysis and is by many considered the "holy grail" of the field.

In the weak adversaries setting, significantly less is known. For the (h, k)-server problem, the exact deterministic competitive ratio is $\frac{k}{k-h+1}$ on uniform metrics (equivalent to paging) [26] and weighted stars (equivalent to weighted paging) [27]. More recently, Bansal et al. [4] showed that as $k/h \to \infty$, the competitive ratio on weighted trees is bounded by a constant depending exponentially on the depth of the tree. Using randomization, exponentially better competitive ratios can be achieved on uniform metrics [28] and weighted stars [2]. For trees of depth D and $k \geq (1+\epsilon) \cdot h$, the techniques of Reference [9] yield an $O(D \cdot \log(1/\epsilon))$-competitive *fractional* algorithm,[2] as shown in Reference [10].

On general metrics, the (h, k)-server problem is only poorly understood. No algorithm is known for general metrics that performs better than disabling the $k - h$ extra servers and using h servers only. In fact, for the line it was shown [3, 4] that the Double Coverage Algorithm and the Work Function Algorithm—despite achieving the optimal competitive ratio of h if $k = h$ [6, 11] – perform slightly *worse* in the resource augmentation setting than disabling the $k - h$ extra servers and applying the same algorithm to h servers only. For the case that h is not fixed, the Work Function Algorithm was shown to be $2h$-competitive simultaneously against any number $h \leq k$ of offline servers [20].

In terms of lower bounds, it is known that unlike for (weighted) paging, the competitive ratio does not converge to 1 on general metrics even as $k/h \to \infty$. Prior to this work, the best known lower bounds were 2 on the line, due to Bar-Noy and Schieber (see Reference [8, p. 175]), and 2.4 for general metrics, as shown recently by Bansal et al. [4].

The closest publication to this work is by Csirik et al. [16], which studies a problem that is essentially the special case of the ∞-server problem on the uniform metric space augmented by a faraway source. It is cast as a paging problem where new cache slots can be bought at a fixed price per unit and gives matching upper and lower bounds of $\approx$3.146 on the competitive ratio.

1.2 Our Results

Our main result is an equivalence theorem between the ∞-server problem and the (h, k)-server problem, presented in Section 2. It states that the ∞-server problem is competitive on every metric space if and only if the (h, k)-server problem is $O(1)$-competitive on every metric space as $k/h \to \infty$. We show further that it is not even necessary to let k/h tend to infinity, because in the positive case, there must also exist some $k = O(h)$. The theorem holds also if "every metric space" is replaced by "the real line."

In Section 3, we present upper and lower bounds on the competitive ratio of the ∞-server problem on a variety of metric spaces. Extending the work in Reference [16], we present a tight lower bound of approximately 3.146 for a class of metric spaces that includes every non-discrete space as well as some simple weighted stars. This lower bound is then turned into a lower bound of the same value for the (h, k)-server problem, improving on the previous lower bounds for this setting. We show how recent work by Bansal et al. [4] can be adapted to give an upper bound on the

[2]When the tree is an HST, this can be rounded to a randomized integral algorithm.

competitive ratio of the ∞-server problem on bounded-depth weighted trees. We also consider layered graph metrics, which are equivalent (up to a factor of 2) to general graph metrics. We have not settled the case for their competitive ratio, but we present a natural algorithm with tight analysis and a lower bound of 3 for general algorithms, which also beats the previous lower bounds on the (h, k)-server problem.

In Section 4, we show how a variety of known algorithms such as the work function and balancing algorithms fail for the ∞-server problem, even on the real line. We focus in particular on a class of speed-adjusted variants of the well-known double coverage algorithm.

Finally, we present a useful reduction from arbitrary metric spaces to bounded subspaces in Section 5. In particular, the ∞-server problem on the line is competitive if and only if it is competitive for the special case where requests are restricted to some bounded interval further away from the source.

1.3 Recent Developments

Following the original announcement of our work [14], Bienkowski et al. [7] showed very recently that the competitive ratio of the (h, k)-server problem on general metrics is $\Omega(\log \log h)$, even if $k/h \to \infty$. Thus, as implied by our equivalence theorem, there exists no competitive algorithm for the ∞-server problem on general metrics. In fact, it is stated in Reference [7] that they first found this lower bound by studying the ∞-server problem.

1.4 Preliminaries

Let $M = (M, d)$ be a metric space and let s be a point of M. In the *∞-server problem on* (M, s), an unbounded number of servers starts at point s and serves a finite sequence $\sigma = (\sigma_0 = s, \sigma_1, \sigma_2, \ldots, \sigma_m)$ of requests $\sigma_i \in M$. Serving a request entails moving one of the servers to it. The goal is to minimize the total distance traveled by the servers.

We drop s in the notation if the location of the source is not relevant or understood. We refer to the action of moving a server from the source to another point as *spawning*. Throughout this work, we use the letter d for the metric associated with the metric space.

In the online setting, the requests are revealed one-by-one and need to be served immediately without knowledge of future requests. All algorithms considered in this article are deterministic. An algorithm is called *lazy* if it moves only one server to serve a request at an unoccupied point and moves no server if the requested point is already covered. An algorithm is called *local* [15] if it moves a server from a to b only if there is no server at some other point c on a shortest path from a to b, i.e., with $d(a, b) = d(a, c) + d(c, b)$. It is easy to see that any algorithm can be turned into a lazy and local algorithm without increasing its cost (i.e., the total distance traveled by all servers).

For an algorithm ALG, we denote by $ALG(\sigma)$ its cost on the request sequence σ. Similarly, we write $OPT(\sigma)$ for the optimal (offline) cost.

An online algorithm ALG is *ρ-competitive* for $\rho \geq 1$ if $ALG(\sigma) \leq \rho OPT(\sigma) + c$ for all σ, where c is a constant independent of σ. The *competitive ratio* of an algorithm is the infimum of all such ρ. We say that an algorithm is *competitive* if it is ρ-competitive for some ρ. We also call an online problem itself (ρ-)competitive if it admits such an algorithm. If the additive term c in the definition is 0, then the algorithm is also called *strictly ρ-competitive* [17].

The *(h, k)-server problem on M* is defined like the ∞-server problem except that the number of servers is k for the online algorithm and h for the optimal (offline) algorithm against whom it is compared in the definition of competitiveness. For this problem, the servers are not required to start at the same point, although a different initial configuration would only affect the additive term c. The problem is interesting only when $k \geq h$, as otherwise the competitive ratio is infinite. The case $h = k$ is the standard k-server problem and the case $k \geq h$ is known as the *weak adversaries*

model. One major open problem is to determine the competitive ratio of the (h, k)-server problem, as k tends to infinity.

We will sometimes write OPT_h and OPT_∞ for the optimal offline algorithm, where the index specifies the number of servers available to it.

The following is a technical lemma that allows us to handle algorithmic "convergence" by consistently extending a sequence of parameterized algorithms (e.g., by the number of servers they use) to their limit:

Lemma 1.1. *Fix a metric space M. For each n, let ALG_n be an algorithm with cost $ALG_n(\sigma) \leq f_n(\sigma)$ for every request sequence σ on M, where f_n are functions such that*

$$\lim_{n\to\infty} f_n(\sigma) = f(\sigma).$$

Then there exists an algorithm ALG with $ALG(\sigma) \leq f(\sigma)$.

Proof. Assume, without loss of generality, that all algorithms ALG_n are lazy.

For every request sequence σ, consider the equivalence relation $\equiv_\sigma$ on natural numbers in which $n \equiv_\sigma n'$ if and only if $ALG_n(\sigma)$ and $ALG_{n'}(\sigma)$ serve σ in exactly the same way (i.e., make exactly the same moves). To every σ, we associate an equivalence class $H(\sigma)$ of $\equiv_\sigma$ such that

- $H(\sigma)$ is infinite,
- $H(\sigma r) \subseteq H(\sigma)$, for every request r.

This is done inductively on the length of σ (in a manner reminiscent of König's lemma) as follows: For the base case when σ is the empty request sequence, $H(\sigma) = \mathbb{N}$. For the induction step, suppose that we have defined $H(\sigma)$. Consider the equivalence classes of $\equiv_{\sigma r}$, a refinement of the equivalence classes of $\equiv_\sigma$. Since there are only finitely many possible ways to serve r, they partition $H(\sigma)$ into finitely many parts. At least one of these parts is infinite and we select it to be $H(\sigma r)$; if there is more than one such set, then we select one arbitrarily, say, the lexicographically first.

Given such a mapping H, we define the online algorithm ALG, which serves every σ in the same way as all the online algorithms ALG_n for $n \in H(\sigma)$. The second property of H guarantees that ALG is a well-defined online algorithm.

By construction, $ALG(\sigma) = ALG_n(\sigma) \leq f_n(\sigma)$ for all $n \in H(\sigma)$. Taking the limit as $n \to \infty$ along the subsequence $H(\sigma) \subseteq \mathbb{N}$ yields $ALG(\sigma) \leq f(\sigma)$. □

The following two propositions will be useful later in the article:

Proposition 1.2. *If for every metric space there exists a competitive algorithm for the ∞-server problem, then there exists a universal competitive ratio ρ such that the ∞-server problem is* strictly *ρ-competitive on every metric space.*

Proof. We first show the existence of ρ such that the ∞-server problem is ρ-competitive (strictly or not) on every metric space. Suppose such ρ does not exist, then for every $n \in \mathbb{N}$, we can find a metric space M_n containing some point s_n such that the ∞-server problem on (M_n, s_n) is not n-competitive. Consider the metric space obtained by taking the disjoint union of all spaces M_n and gluing all the points s_n together. The ∞-server problem would not be competitive on this metric space, in contradiction to the assumption.

Analogously, we can also find a universal constant c such that a ρ-competitive algorithm ALG with additive constant c in the definition of competitive ratio exists for general metrics. Let ALG_n be the algorithm that acts like ALG if all distances were scaled by a factor n. Then for each request sequence σ it holds that $ALG_n(\sigma) \leq \rho OPT(\sigma) + c/n$. Now, Lemma 1.1 implies the existence of a strictly ρ-competitive algorithm. □

With a very similar argument, we get:

Proposition 1.3. *Let $k = k(h)$ be a function of h. Suppose that for every metric space M and for all h there exists an $O(1)$-competitive algorithm for the (h, k)-server problem on M. Then there exists a universal competitive ratio ρ such that the (h, k)-server problem is strictly ρ-competitive on every metric space if all servers start at the same point.*

2 EQUIVALENCE OF ∞-SERVER AND WEAK ADVERSARIES

The main result of this section is the following tight connection between the ∞-server problem and the weak adversaries model. Although we provide a proof for deterministic algorithms only, we remark that the proof can be extended to randomized algorithms as well.

Theorem 2.1. *The following are equivalent:*

(a) The ∞-server problem is competitive.
(b) The (h, k)-server problem is $O(1)$-competitive as $k/h \to \infty$.
(c) For each h there exists $k = O(h)$ so the (h, k)-server problem is $O(1)$-competitive.

The three statements above are also equivalent if we fix the metric space to be the real line.

The implication "(c) $\Longrightarrow$ (b)" is trivial. The proof of the equivalence theorem consists in its core of two reductions. Theorem 2.2 contains the easier of the two reductions, which is from the ∞-server problem to the k-server problem against weak adversaries ("(b) $\Longrightarrow$ (a)"). This reduction follows directly from Lemma 1.1 proved in the last section. By Propositions 1.2 and 1.3, it suffices to consider only strictly competitive algorithms. Theorem 2.3 proves the converse reduction for general metric spaces ("(a) $\Longrightarrow$ (c)"), and Theorem 2.6 specializes it to the line.

Theorem 2.2. *Fix a metric space M and consider algorithms with all servers starting at some $s \in M$. If for every h there exists k such that the (h, k)-server problem on M is strictly ρ-competitive, for some constant ρ, then there exists a strictly ρ-competitive online algorithm for the ∞-server problem on M.*

Proof. For any request sequence σ, the optimal offline cost $OPT_\infty(\sigma)$ is equal to $OPT_h(\sigma)$ whenever h is at least the length of σ. Thus, the proof is immediate from Lemma 1.1 by choosing ALG_h to be a strictly ρ-competitive algorithm for the (h, k)-server problem and $f_h(\sigma) = \rho \cdot OPT_h(\sigma)$. □

We now show the reduction from the k-server problem against weak adversaries to the ∞-server problem on general metric spaces.

Theorem 2.3. *If the ∞-server problem on general metric spaces is strictly $\tilde{\rho}$-competitive, then there exists a constant ρ such that the (h, k)-server problem is ρ-competitive, for $k = O(h)$. In particular, for every $\epsilon > 0$, we can take $\rho = (3 + \epsilon)\tilde{\rho}$ and any $k \geq (1 + 1/\epsilon)\tilde{\rho}h$.*

Proof. Fix some metric space M and a point $s \in M$. We will describe a strictly ρ-competitive algorithm for the (h, k)-server problem on M for the case that all servers start at s. This implies a (not necessarily strictly) ρ-competitive algorithm for any initial configuration.

The idea is to simulate a strictly $\tilde{\rho}$-competitive ∞-server algorithm, but whenever it would spawn a $(k + 1)$st server, we bring all servers back to the origin and restart the algorithm. The problem is that the overhead cost for returning the servers to the origin may be very high. To compensate for this, we assume that every time the servers return to the origin, they pretend to start from a different point further away from the origin. This motivates the following notation:

Definition 2.4. Given a metric M, a point $s \in M$, and a value $w \geq 0$, we will use the notation $M_{s \oplus w}$ to denote the metric derived from M by increasing the distance from s to every other point (additively) by w; we will also denote the relocated point by $s \oplus w$.

Let ALG_∞ denote a strictly $\tilde{\rho}$-competitive online algorithm for the ∞-server problem. We now define an online algorithm ALG_k for k servers (all starting at s). We will make use of the notation $A(\sigma; s)$ to denote the cost of algorithm A to serve the request sequence σ when all servers start at s.

Definition 2.5 (ALG_k derived from ALG_∞). Algorithm ALG_k runs in phases with the initial phase being the 0th phase. At the beginning of every phase, all servers of ALG_k are at s. In every phase i, the algorithm simulates the ∞-server algorithm ALG_∞, whose servers start at $s \oplus w_i$ for some $w_i \geq 0$. The parameters w_i are determined online, and initially $w_0 = 0$. Whenever ALG_∞ spawns a server from $s \oplus w_i$, algorithm ALG_k spawns a server from s.

The phase ends just before ALG_∞ spawns its $(k+1)$st server or when the request sequence ends. In the former case, all servers of ALG_k return to s to start the $(i+1)$st phase. To determine the starting point of the simulated algorithm of the next phase, we set

$$w_{i+1} = \epsilon \frac{OPT_h(\sigma_i; s)}{h}, \tag{1}$$

where σ_i is the sequence of requests during phase i.

Let n be the number of phases. The cost of ALG_k for the requests in phase $i < n$ is $ALG_\infty(\sigma_i; s \oplus w_i) - kw_i$; the last term is subtracted because the k servers do not have to actually travel the distance between $s \oplus w_i$ and s. However, for the last phase, no such term can be subtracted, since we do not know how many servers are spawned during the phase, and we can only bound the cost from above by $ALG_\infty(\sigma_n; s \oplus w_n)$. The cost of returning the servers to s at the end of a phase can at most double the cost during the phase.

From this, we see that the total cost of ALG_k in phase i is

$$cost_i \leq \begin{cases} 2\,(ALG_\infty(\sigma_i; s \oplus w_i) - kw_i) & \text{for } i < n \\ ALG_\infty(\sigma_n; s \oplus w_n) & \text{for } i = n. \end{cases}$$

Since ALG_∞ is strictly $\tilde{\rho}$-competitive, we have

$$\begin{aligned} ALG_\infty(\sigma_i; s \oplus w_i) &\leq \tilde{\rho}\, OPT_\infty(\sigma_i; s \oplus w_i) \\ &\leq \tilde{\rho}\, OPT_h(\sigma_i; s \oplus w_i) \\ &\leq \tilde{\rho}\, (OPT_h(\sigma_i; s) + hw_i), \end{aligned}$$

and substituting this in the expression for the cost, we can bound the total cost by

$$\begin{aligned} ALG_k(\sigma; s) = \sum_{i=0}^{n} cost_i &\leq 2 \sum_{i=0}^{n-1} (\tilde{\rho}(OPT_h(\sigma_i; s) + hw_i) - kw_i) + \tilde{\rho}(OPT_h(\sigma_n; s) + hw_n) \\ &= 2 \sum_{i=0}^{n-1} (\tilde{\rho} OPT_h(\sigma_i; s) - (k - \tilde{\rho}h)w_i) + \tilde{\rho} OPT_h(\sigma_n; s) + \tilde{\rho} h w_n. \end{aligned}$$

The parameters w_i were selected so for $k \geq (1 + 1/\epsilon)\tilde{\rho}h$ the summation telescopes, and we are left with

$$\begin{aligned} ALG_k(\sigma; s) &\leq 2\,\tilde{\rho}\, OPT_h(\sigma_{n-1}; s) + \tilde{\rho}\, OPT_h(\sigma_n; s) + \tilde{\rho}\, \epsilon\, OPT_h(\sigma_{n-1}; s) \\ &\leq (3 + \epsilon)\, \tilde{\rho}\, OPT_h(\sigma; s). \end{aligned}$$

□

The previous reduction requires the ∞-server problem to be competitive on *every* metric space. The following variant only requires the ∞-server problem to be competitive on the line:

Theorem 2.6. *If the ∞-server problem on the line is ρ-competitive, then for every $h \in \mathbb{N}$ and $\epsilon > 0$, the (h,k)-server problem on the line is $(3+\epsilon)\rho$-competitive, when $k \geq 2\lceil(1+1/\epsilon)\rho h\rceil$.*

Proof. Given the premise, we will show that there is a $(3+\epsilon)\rho$-competitive algorithm for the (h,k)-server problem on the half-line $M := [0,\infty)$ when $k \geq (1+1/\epsilon)\rho h$. The theorem then follows by doubling the number of servers and using half of them in each half-line.

By a straightforward scaling argument, if the ∞-server problem on the line is ρ-competitive, then it is also strictly ρ-competitive. This in turn implies a strictly ρ-competitive online algorithm for the space $M_{0\oplus w}$, since this space is isometric to the subspace $\{-w\} \cup (0,\infty)$ of the line. A straightforward adaptation of the proof of the previous theorem now shows the existence of a $(3+\epsilon)\rho$-competitive algorithm for the (h,k)-server problem on M when $k \geq (1+1/\epsilon)\rho h$. □

In the next section, we give upper and lower bounds on the competitive ratio of the ∞-server problem on some particular metric spaces.

3 UPPER AND LOWER BOUNDS

Unlike the k-server problem, which is 1-competitive if and only if the metric space has at most k points and conjectured k-competitive otherwise, the situation is more diverse for the ∞-server problem. As mentioned earlier, the recent result of Reference [7] shows that there exists a metric space where no competitive algorithm exists. On the other extreme, on uniform metric spaces (where all distances are the same) the problem is trivially 1-competitive even if the metric space consists of uncountably many points: An optimal strategy in this case is to spawn a server to every requested point. More generally, this strategy achieves a finite competitive ratio on any metric space where distances are bounded from below and above by positive constants. This shows that statements about the competitive ratio for the ∞-server problem cannot be as simple as the (conjectured) dichotomy for the k-server problem, which depends only on the number of points of the metric space. In this section, we derive bounds on the competitive ratio for particular classes of metric spaces.

3.1 Weighted Trees

We consider the ∞-server problem on metric spaces that can be modeled by edge-weighted trees. The points of the metric space are the nodes of the tree, and the distance between two nodes is the sum of edge weights along their connecting path. We choose the source of the metric space as the root of the tree, and define the depth of the tree as the maximal number of edges from the root to a leaf. The number of nodes can be infinite (otherwise the ∞-server problem is trivially 1-competitive), but we assume the depth to be finite.

An upper bound on the competitive ratio of such trees follows easily from an upper bound for the (h,k)-server on such trees [4] and the equivalence theorem:

Theorem 3.1. *The competitive ratio of the ∞-server problem on trees of depth D is at most $O(2^D \cdot D)$.*

Proof. Bansal et al. [4] showed that the competitive ratio of the (h,k)-server problem on trees of depth D is at most $O(2^D \cdot D)$ provided that k/h is large enough. If all servers start at the root, then it is in fact strictly $O(2^D \cdot D)$-competitive (because the initial potential function value in Reference [4] is 0 in this case). Thus, Theorem 2.2 implies the result for the ∞-server problem. □

As an important special case, we see that the ∞-server problem is $O(1)$-competitive on weighted stars; rooting a weighted star at the source leaf, it becomes a tree of depth 2.

As shown in Reference [7], any algorithm for trees of depth D must have competitive ratio at least $\Omega(\log D)$, even if it uses randomization. Thus, there is a doubly-exponential gap between the upper and lower bounds.

3.2 Non-discrete Spaces and Spaces with Small Infinite Subspaces

The following theorem gives a lower bound of 3.146 on the competitive ratio of the ∞-server problem on any metric space containing an infinite subspace of a diameter that is small compared to the subspace's distance from the source. For example, every non-discrete metric space has this property, since non-discrete metric spaces contain infinite subspaces of arbitrarily small diameter. The theorem is a generalization of such a lower bound established in Reference [16] for a variant of the paging problem where cache cells can be bought. Crucial parts of the subsequent proof are as in Reference [16].

Theorem 3.2. *Let M be a metric space containing an infinite subspace $M_0 \subset M$ of finite diameter δ and a point $s \in M \setminus M_0$ such that the infimum Δ of distances between s and points in M_0 is positive. Let $\lambda > 3.146$ be the largest real solution to*

$$\lambda = 2 + \ln \lambda. \tag{2}$$

The competitive ratio of any deterministic online algorithm for the ∞-server problem on (M, s) is bounded from below by a value that converges to λ as $\Delta/\delta \to \infty$. In particular, the competitive ratio is at least λ if $M \setminus \{s\}$ contains a non-discrete part.

Proof. By scaling the metric, we can assume that $\delta = 1$. Let $p_1, p_2, p_3, \ldots$ be infinitely many distinct points in M_0.

Fix some lazy deterministic online algorithm *ALG*. We consider the request sequence that always requests the point p_i with i minimal such that p_i is not occupied by a server of *ALG*. We call a move of a server between two points in M_0 *local* (i.e., every move that does not spawn is local). Let f_j be the cumulative cost of local moves incurred to *ALG* until it spawns its jth server. Let σ_k be this request sequence that is stopped right after *ALG* spawns its kth server, for some large k. The total online cost is

$$ALG(\sigma_k) \geq k\Delta + f_k. \tag{3}$$

Let $h = \lceil k/\lambda \rceil$. We consider several offline algorithms that start behaving the same way, so we think of it as one algorithm initially that is forked into several algorithms later. The offline algorithms make use of only h servers and they begin by spawning them to the points $p_1, \ldots, p_h$. They do not need to move any servers until *ALG* spawns its hth server. Whenever *ALG* spawns its jth server for some $j \geq h$, every offline algorithm is forked to h distinct algorithms: Each of them moves a different server to p_{j+1} (to prepare for the next request, which will be at p_{j+1}). We will keep the invariant that each offline algorithm already has a server at the next request. To this end, whenever *ALG* does a local move from p to p', every offline algorithm that does not have a server at p moves a server from p' to p; note that the algorithm had a server at p' by the invariant, and the next request will be at p.

When *ALG* has j spawned servers ($j \geq h$), the offline algorithms are in $\binom{j}{h-1}$ different configurations, each of which occurs equally often among them. If *ALG* does a local move from p to p', then there are $\binom{j-1}{h-1}$ different offline configurations for which a local move is made in the opposite direction. Thus, for each local move by *ALG* while having spawned j servers in total, a portion $\binom{j-1}{h-1}/\binom{j}{h-1} = \frac{j-h+1}{j}$ of the offline algorithms move a server in the opposite direction for the same cost.

We use the average cost of all these offline algorithms as an upper bound on the optimal cost. The cost of spawning h servers is at most $h(\Delta+1)$, and the average cost while ALG has j spawned servers (for $j = h, \dots, k-1$) is at most $\frac{j-h+1}{j}(f_{j+1} - f_j) + 1$ (with the "+1" coming from the move when offline algorithms fork). Hence,

$$\begin{aligned} OPT(\sigma_k) &\leq h(\Delta+1) + k - h + \sum_{j=h}^{k-1} \frac{j-h+1}{j}(f_{j+1} - f_j), \\ &\leq h\Delta + k + \frac{k-h}{k-1} f_k - \frac{f_h}{h} - \sum_{j=h+1}^{k-1} \frac{h-1}{j(j-1)} f_j. \end{aligned}$$

Note that $\frac{f_k}{k}$ is bounded from above, since otherwise ALG would not be competitive (because an offline algorithm can spawn k servers at the start for cost at most $k(\Delta+1)$ and pay no additional cost in the period where ALG suffers cost f_k for its local moves), and it is bounded from below by 0. Thus, $L = \liminf_{k\to\infty} \frac{f_k}{k}$ is finite. In the following, we use the asymptotic notation $o(1)$ for terms (which might be negative) that disappear as $k \to \infty$. We can choose arbitrarily large values of k such that $\frac{f_k}{k} = L + o(1)$. Choosing k sufficiently large (so $h = \lceil k/\lambda \rceil$ is sufficiently large), we have $\frac{f_j}{j} \geq L + o(1)$ for all $j \geq h$. Moreover, $\sum_{j=h+1}^{k-1} \frac{1}{j-1} = \ln(\lambda) + o(1)$. This allows us to simplify the previous bound to

$$\begin{aligned} OPT(\sigma_k) &\leq \frac{k}{\lambda}(\Delta + \lambda + (\lambda - 1 - \ln(\lambda))L + o(1)) \\ &= \frac{k}{\lambda}(\Delta + L + \lambda + o(1)), \end{aligned}$$

where the last step uses Equation (2).

The competitive ratio is at least

$$\begin{aligned} \frac{ALG(\sigma_k) + O(1)}{OPT(\sigma_k)} &\geq \frac{k\Delta + f_k + O(1)}{\frac{k}{\lambda}(\Delta + L + \lambda + o(1))} \\ &= \lambda \cdot \frac{\Delta + L}{\Delta + L + \lambda} + o(1). \end{aligned}$$

The $O(1)$ term in the first expression ensures that the lower bound holds even for the non-strict competitive ratio. The fraction in the last term tends to 1 as $\Delta \to \infty$. □

This bound is tight due to a matching upper bound in Reference [16] that shows (translated to the terminology of the ∞-server problem) that a competitive ratio of λ can be achieved on metric spaces where all pairwise distances are 1 except that the source is at some larger distance Δ from the other points.

The previous theorem together with the equivalence theorem also allows us to obtain a new lower bound for the k-server problem against weak adversaries.

Corollary 3.1. *For sufficiently large h, there is no 3.146-competitive algorithm for the (h,k)-server problem on the line, even if $k \to \infty$.*

Proof. By a scaling argument it is easy to see that if the ∞-server problem on the line is ρ-competitive, then it is also *strictly* ρ-competitive. Thus, the statement follows from Theorems 2.2 and 3.2. □

This improves upon both the previous best known lower bounds of 2 for this problem on the line [8, p. 175] and 2.4 on general metric spaces [4].

3.3 Layered Graphs

A *layered graph of depth D* is a graph whose (potentially infinitely many) nodes can be arranged in layers $0, 1, \ldots, D$ so all edges run between adjacent layers and each node—except for a single node in layer 0—is connected to at least one node of the previous layer. The induced metric space is the set of nodes with the distance being the minimal number of edges of a connecting path. For the purposes of the ∞-server problem, the single node in layer 0 is the source. We assume $D \geq 2$ to avoid trivial cases.

Note that a connected graph is layered if and only if it is bipartite. Moreover, any graph can be embedded into a bipartite graph by adding a new node in the middle of each edge. So, essentially, layered graphs capture *all* (unweighted) graph metrics.

Let ***Move Only Outwards (MOO)*** be some lazy and local algorithm for the ∞-server problem on layered graphs that moves servers along edges only in the direction away from the source. Not surprisingly, the competitive ratio of this simple algorithm is quite bad, and we show that it is exactly $D - 1/2$. Nonetheless, at least for $D \leq 3$ this is actually the optimal competitive ratio.

Theorem 3.3. *The competitive ratio of MOO is exactly* $D - \frac{1}{2}$.

Proof.

Upper bound: Consider some final configuration of the algorithm. Let n_j be the number of servers in the jth layer. Then the cost of MOO is

$$cost = \sum_{j=1}^{D} j n_j.$$

To obtain a lower bound on *OPT*, observe that every node occupied by MOO in the final configuration must have been visited by an offline server at least once. We account an offline cost of 1 for each visit of a node on layers $1, \ldots, D-2$ and an offline cost of 2 for each visit of a node on layer D. This cost of 2 covers the last two edge-traversals before visiting the layer-D-node, so this may include serving a request on layer $D-1$. If $n_{D-1} > n_D$, then we can account another $n_{D-1} - n_D$ cost for visiting the remaining at least $n_{D-1} - n_D$ requested nodes on layer $D-1$. In summary,

$$OPT \geq \sum_{j=1}^{D-2} n_j + 2n_D + (n_{D-1} - n_D)^+,$$

where $(n_{D-1} - n_D)^+ := \max\{0, n_{D-1} - n_D\}$. The upper bound on the competitive ratio follows, since

$$\begin{aligned}
\frac{cost}{OPT} &\leq \frac{\sum_{j=1}^{D} j n_j}{\sum_{j=1}^{D-2} n_j + 2n_D + (n_{D-1} - n_D)^+} \\
&\leq \frac{(D-2)\sum_{j=1}^{D-2} n_j + (2D-1)n_D + (D-1)(n_{D-1} - n_D)^+}{\sum_{j=1}^{D-2} n_j + 2n_D + (n_{D-1} - n_D)^+} \\
&\leq D - \frac{1}{2}.
\end{aligned}$$

Lower bound: Let $k, n \in \mathbb{N}$ be some integers much larger than D. We construct the following graph: Layers $0, \ldots, D-2$ consist of one node each and layers $D-1$ and D consist of infinitely many nodes each, denoted $a_0, a_1, a_2, \ldots$ and $b_0, b_1, b_2, \ldots$, respectively. For each $i \in \mathbb{N}_0$, the k nodes $b_{ik}, b_{ik+1} \ldots, b_{(i+1)k-1}$ are adjacent to each of the $2k$ nodes $a_{ik}, a_{ik+1}, a_{(i+2)k-1}$ and to no other nodes. The set of remaining edges is uniquely determined by the fact that this is a layered graph of depth D.

The request sequence consists of n rounds $0, 1, \ldots, n-1$, where each request in round i is at a node from the list $a_{ik}, a_{ik+1}, \ldots, a_{(i+1)k-1}, b_{ik}, b_{ik+1}, \ldots, b_{(i+1)k-1}$. Round i starts with requests on the nodes $a_{ik}, a_{ik+1}, \ldots, a_{(i+1)k-1}$. Then, for $j = 0, \ldots, k-1$, the adversary first requests b_{ik+j} and then requests whichever node from $a_{ik}, a_{ik+1}, \ldots, a_{(i+1)k-1}$ has been left by a MOO-server to serve the request at b_{ik+j}. Note that by definition of MOO and the graph, the server it moves to b_{ik+j} does indeed come from $a_{ik}, a_{ik+1}, \ldots, a_{(i+1)k-1}$.

In round i, MOO first pays $k(D-1)$ to move k servers to $a_{ik}, a_{ik+1}, \ldots, a_{(i+1)k-1}$ and then, for each $j = 0, \ldots, k-1$, it pays 1 to move to b_{ik+j} and $D-1$ to spawn a new server at the group $a_{ik}, a_{ik+1}, \ldots, a_{(i+1)k-1}$. Over n rounds this makes a total cost of $n(k(D-1) + k(1+D-1)) = nk(2D-1)$.

An offline algorithm can serve requests as follows: The requests at $a_{ik}, \ldots, a_{(i+1)k-1}$ at the beginning of round i are served by spawning if $i = 0$ (for cost $(D-1)k$) and by sending servers from $b_{(i-1)k}, \ldots, b_{ik-1}$ if $i \geq 1$ (for cost k). The request at b_{ik} is served by spawning[3] a server (cost D) and the requests at $b_{ik+1}, \ldots, b_{ik+k-1}$ are served by sending a server from a node in $a_{ik}, \ldots, a_{(i+1)k-1}$ that will not be requested any more (cost 1 each, so $k-1$ per round). Over n rounds, this adds up to an offline cost of $(D-1)k + (n-1)k + n(D+k-1) = 2nk + (D-2)k + n(D-1)$. The ratio of online and offline cost is

$$\frac{nk(2D-1)}{2nk + (D-2)k + n(D-1)} = \frac{2D-1}{2 + \frac{D-2}{n} + \frac{D-1}{k}},$$

which gets arbitrarily close to $D - \frac{1}{2}$ for n and k large enough. □

Theorem 3.4. *The competitive ratio of the ∞-server problem on layered graphs of depth D is exactly* 1.5 *for $D = 2$, exactly* 2.5 *for $D = 3$ and at least* 3 *for $D \geq 4$.*

Proof. For $D = 2$, the only possibility to move a server closer to the source is from layer 2 to layer 1. But, since spawning to layer 1 is at least as good, we can restrict our attention to algorithms of the type MOO. The result follows from Theorem 3.3.

For $D = 3$, the upper bound follows from Theorem 3.3. It remains to show the lower bounds for $D \geq 3$.

Fix some large integers $k, n \in \mathbb{N}$. Consider the following layered graph of depth D: For $i = 0, \ldots, D-2$ there exists a single node v_i in layer i, and any pair of vertices (v_i, v_{i+1}) is connected by an edge. The remaining nodes are defined inductively as all nodes obtained by the following two rules:

- There exist a set S_0 of $2k$ nodes and sets A^{S_0} and B^{S_0} of k nodes.
- Let S be a set of $2k$ nodes such that A^S and B^S exist. Then for each $S' \subset S \cup B^S$ of size $2k$ there are sets $A^{S'}$ and $B^{S'}$ of k nodes.

The sets A^S and B^S are all disjoint from each other. The nodes in the sets A^S are in layer $D-1$, the nodes in S_0 and in the sets B^S are in layer D. For a node in some set A^S, the set of adjacent nodes is $S \cup B^S \cup \{v_{D-2}\}$ (see Figure 1).

For purposes of the analysis below, we further define a *generation* of a node as follows: The nodes $v_0, \ldots, v_{D-1}$ and the nodes in S_0 have generation 1. The generation of nodes in A^S and B^S is the maximal generation of any node in S plus 1.

Let *ALG* be some online algorithm. We assume without loss of generality that *ALG* is lazy and local.

[3]This is slightly sub-optimal, but it simplifies the description of the offline algorithm and suffices to obtain the tight lower bound.

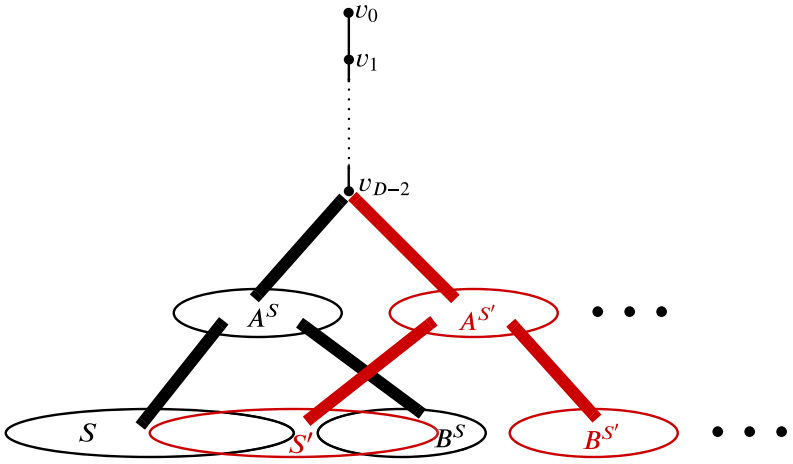

Fig. 1. Inductive construction of a layered graph for the lower bounds for $D \geq 3$ in Theorem 3.4. Thick bars between two sets of vertices indicate that all vertices in one set are connected to all vertices of the other set.

The adversary chooses the following request sequence against *ALG*: First, it requests the nodes in S_0 until *ALG* has a server at each of them. The adversary also moves $2k$ servers to these nodes. The adversary uses only these $2k$ servers for the entire sequence of requests. The remainder of the requests consists of several rounds. We will keep the invariant that at the beginning of the ith round, the $2k$ adversary servers occupy a set S for which A^S and B^S (with nodes of generation $i+1$) exist, and the online servers occupy nodes of generation at most i. Clearly this holds before the first round. Let $A^S = \{a_1, \ldots, a_k\}$ and $B^S = \{b_1, \ldots, b_k\}$.

The requests of the ith round are divided into part a and part b, consisting of *steps* a.1,…,a.k, b.1,…,b.k that are executed in this order. Step a.j consists of the following one or two requests: First request a_j. If *ALG* moves a server from some $b \in S$ towards a_j, then immediately request b. We can assume that online servers cover A^S after the end of part a (otherwise request nodes in A^S again at the end of part a until this is the case). Step b.j consists of the following two or three requests: First request b_j. Note that from each node of generation at most i, there exist shortest paths to b_j along any node from A^S. Thus, since *ALG* is local, it will move a server from some $a \in A^S$ towards b_j. The second request of step b.j is at this node a and, if *ALG* moves a server from some $b \in S \cup B^S$ towards a, then the step contains a third request at b.

The adversary cost per round is at most $2k+2$: For each $j = 1, \ldots, k$, there are at least j nodes in S that will *not* be requested during steps a.j, …, a.k, b.1, …, b.$(k-1)$. Hence, the adversary can serve all requests of part a for cost k by moving k servers from S towards A^S while keeping servers at all nodes of S that will be requested during the steps b.1,…,b.$(k-1)$. Similarly, it can serve the steps b.1,…,b.$(k-1)$ for cost $k-1$ by moving $k-1$ servers from A^S to B^S. The final step b.k of the round can be served at cost 3 using the last server in A^S to serve the requests and finish with all $2k$ offline servers in some set $S' \subseteq S \cup B^S$.

We analyze the online cost for the cases $D = 3$ and $D \geq 4$ separately.

If $D = 3$, then the cost for each step a.j is at least 2 and the cost for each step b.j is at least 3. Thus, the cost per round is at least $5k$. As k goes to infinity, the ratio of online and offline cost in each round converges to 2.5. As the number of rounds goes to infinity, the online and offline costs before the first round become negligible, which proves the lower bound of 2.5 for $D = 3$.

For $D \geq 4$, we use a potential Φ equal to the number of online servers in layer $D-1$. As we consider different subsequences of requests, we write $\Delta\Phi$ and $\Delta cost$ for the change of Φ and

the cost suffered by A, respectively, during this subsequence. During step a.j, either Φ does not change and the cost is at least 2, or Φ increases by 1 and the cost is at least 3. Thus, during step a.j, we have $\Delta cost \geq 2 + \Delta\Phi$ and hence during part a, we have $\Delta cost \geq 2k + \Delta\Phi$. During step b.$j$, either Φ decreases by 1 and the cost is at least 3, or Φ does not change and the cost is at least 4. Thus, during part b, we have $\Delta cost \geq 4k + \Delta\Phi$. In total, this adds up to $\Delta cost \geq 6k + \Delta\Phi$ during the round. Over n rounds, this makes $\Delta cost \geq 6nk + \Delta\Phi \geq 6nk$, since Φ starts at 0 before the first round and remains nonnegative. As k and n go to infinity, the ratio of our bounds on online and offline cost converges to 3. □

It remains an open problem to close the gap between the lower bound of 3 and the upper bound of 3.5 for $D = 4$. More importantly, we are interested in the question whether an algorithm better than MOO exists for large D, achieving a competitive ratio of less than $D - 1/2$ on layered graphs of depth D.

4 ALGORITHMS WITH UNBOUNDED COMPETITIVE RATIO

We examine the performance of classical algorithms known for the k-server problem when applied to the ∞-server problem. The main focus of this section is a generalization of the Double Coverage algorithm for the line with adjusted server speeds. This idea has proved successful for the (h, k)-server problem (and hence the ∞-server problem) on weighted trees [4]. However, neither of these algorithms is competitive for the ∞-server problem even on the line. We remark that it is not known whether the unbounded lower bound from Reference [7] was shown for a tree where every internal vertex has infinite degree, and it is not known whether this lower bound can be extended to the line metric.

4.1 Work Function Algorithm

The **Work Function Algorithm (WFA)** [13] for the k-server problem achieves a competitive ratio of at most $2k - 1$, which is the best known upper bound for general metric spaces [23]. Given a sequence of requests $r_1, r_2, \ldots$ and a configuration C (i.e., a multiset of server positions), the work function $w_t(C)$ is defined as the minimal cost of serving the first t requests and ending up in configuration C. If C_{t-1} is the server configuration before the tth request, then the algorithm moves to a configuration C_t that contains r_t and minimizes the quantity

$$w_t(C_t) + d(C_{t-1}, C_t), \tag{4}$$

where $d(C_{t-1}, C_t)$ is the cost of moving from C_{t-1} to C_t.

Even though WFA is the best known deterministic algorithm for the k-server problem and has proved useful for many other online problems, it fails for the ∞-server problem. Intuitively, the weakness of WFA is that it chases the configuration of the optimal algorithm, and thus it makes no attempt to outnumber the offline algorithm by using more servers. But if the online algorithm uses only as many servers as the offline algorithm, then the k-server lower bounds apply and yield an infinite competitive ratio for the ∞-server problem.

Proposition 4.1. *WFA is not competitive for the ∞-server problem on the line.*

Proof. Let the source be at 0 and, for some small $\delta > 0$, let $p_1, p_2, \cdots \in [1, 1 + \delta]$ be infinitely many distinct points. Consider the request sequence that always requests the point p_i with i minimal such that p_i is not occupied by an online server. Let σ_k be the prefix of this request sequence until WFA spawns its kth server.

We claim that the optimal way of serving σ_k is to bring k servers to the points $p_1, \ldots, p_k$: Since spawning a new server costs at least 1 and using an old server costs at most δ, WFA spawns a kth

server only if the cost of spawning k servers to the points $p_1, \ldots, p_k$ is at least $1 - \delta$ less than the optimal cost of using $k - 1$ servers to serve σ_k and end up at some fixed configuration of $k - 1$ points from $\{p_1, \ldots, p_k\}$. Moreover, the latter cost differs by at most δ from the optimal cost of serving σ_k with $k - 1$ servers (without the requirement to end up in a particular configuration). Hence, serving σ_k with k servers is at least $1-2\delta$ less than doing so with $k-1$ servers. In particular, the optimal way to service σ_k is to bring k servers to the points $p_1, \ldots, p_k$, as claimed. Therefore, $OPT(\sigma_k) = \sum_{i=1}^{k} p_i = k + o(1)$ as $\delta \to 0$.

Thus, the optimal offline cost increases by $1 + o(1)$ during the period when WFA has k spawned servers, and it increases by at least as much for an offline algorithm that is restricted to using k servers only. Let $cost_k$ be the cost incurred to WFA during this period. Due to the lower bound of k on the competitive ratio of the k-server problem,[4] $cost_k$ is at least k times this increase of the optimal cost (up to an additive error of $o(1)$ as $\delta \to 0$), i.e., $cost_k \geq k + o(1)$. Thus, the total cost of WFA given the request sequence σ_n is at least

$$\sum_{k=1}^{n-1} cost_k = \Omega(n^2).$$

Meanwhile, the optimal cost is $OPT(\sigma_n) = n + o(1)$. Letting n tend to infinity, we obtain an unbounded competitive ratio. □

4.2 Balance and Balance2

The algorithm *Balance* serves a request r by sending a server x that minimizes the quantity $D_x + d(x, r)$, where D_x is the cumulative distance traveled by x so far and $d(x, r)$ is the distance between x and r. For the k-server problem, Balance is k-competitive on metric spaces with $k+1$ points [25] and for weighted paging [11]. Young showed that for weighted paging against a weak adversary with h servers the competitive ratio of Balance is $k/(k - h + 1)$ [27]. On general metric spaces however, Balance has unbounded competitive ratio, even if $k = 2$ [25]. It is therefore unsurprising that it is also not competitive for the ∞-server problem.

Proposition 4.2. *Balance is not competitive for the ∞-server problem on the line.*

Proof. Suppose all servers start at source 0 and consider the request sequence $r_0, r_1, r_2, \ldots, r_n$ where $r_i = 1 - i\epsilon$. As $\epsilon \to 0$, the optimal cost tends to 1 whereas the cost of Balance tends to $n + 1$. Since n can be arbitrarily high, this shows an unbounded competitive ratio. □

The intuitive problem of Balance is that it is not greedy enough. The algorithm *Balance2* by Irani and Rubinfeld [19] compensates for this weakness by giving more weight to the distance between the server and the request: To serve request r, Balance2 sends a server x that minimizes the quantity $D_x + 2d(x, r)$. Irani and Rubinfeld showed that, unlike Balance, Balance2 is competitive for two servers (achieving a competitive ratio of at most 10) and they conjectured that it is also competitive for any other finite number of servers [19].

However, for the ∞-server problem this algorithm is also not competitive:

Proposition 4.3. *Balance2 is not competitive for the ∞-server problem on the line.*

Proof. Suppose the source is at 0 and fix some small constant $\epsilon > 0$. The request sequence consists of several phases, starting with phase 0. Phase i consists of alternating requests at $1 - 2i\epsilon$

[4]The lower bound k for the k-server problem is achieved on any $(k + 1)$-point space by the request sequence that always requests the one point not currently occupied by a server. Note that our constructed request sequence during the period when WFA has k spawned servers is of this type on the $(k + 1)$-point space $\{p_1, \ldots, p_{k+1}\}$.

and $1 - (2i + 1)\epsilon$. We will ensure that all requests of a phase are served by the same online server, and we call this the active server. As soon as the cumulative distance traveled by the active server exceeds $2 - (4i + 5)\epsilon$, the phase ends and a new phase begins. Note that this means that the active server of a phase will not be used to serve any request of a subsequent phase because, by definition of Balance2, the algorithm would rather spawn a new server. Thus, the first request of each phase is served by spawning a new server, which becomes the active server of that phase. While the cumulative distance of the active server is at most $2 - (4i + 5)\epsilon$ and, since its distance from the next request of the current phase is always exactly ϵ, its associated quantity $D_s + 2d(s, r)$ is at most $2 - (4i + 3)\epsilon$. Hence, Balance2 rather uses this server during the phase instead of spawning a new server. Thus, it is indeed the active server that serves *all* requests of its phase.

Let n be the number of phases and choose ϵ small enough so all requests are in the interval $[1/2, 1]$. Thus, the cost of Balance2 is $\Omega(n)$.

An offline algorithm could serve all requests with two servers only that move to 1 and $1 - \epsilon$ initially and then back towards $1/2$, always covering the two points that are requested during a phase, resulting in an offline cost of less than 3. As n goes to infinity, the ratio between online and offline cost becomes arbitrarily large. □

4.3 Double Coverage Variants

Perhaps more surprising than for WFA and balancing algorithms is that a class of algorithms extending the **Double Coverage (DC)** algorithm [11] is also not competitive for the ∞-server problem. The basic DC algorithm on the line serves each request by an adjacent server. If the request lies between two servers, then both servers move towards it at equal speed until one of them reaches the request. A sensible extension of this algorithm seems to be to give different speeds to servers, so they move away from the source faster than towards it. This is motivated by the idea that there is already an excess of servers at the source. It is similar to the algorithm of Reference [4] for bounded depth trees, which moves servers faster if they come from a subtree with many servers.

We consider here only the half-line $[0, \infty)$ with the source at the left border 0. Let x_i be the position of the ith server from the right. We use the notation x_i both for its position and for the server itself. As servers do not overtake each other, x_i is the ith spawned server. Let $\mathcal{S} = \{s_i \geq 1 \mid i = 2, 3, \dots\}$ be a sequence of speeds s_i for moving to the right. The algorithm $\mathcal{S}$-DC is defined as follows:

- If there exist servers x_{i+1} and x_i to the left and right of the request, then move them towards it with speeds s_{i+1} and 1, respectively, until one of the two reaches it.
- If a request does not have a server to its right, then move the rightmost server to the request.

If $s_i = 1$ for all i, then this is precisely the original DC algorithm.

We will prove that $\mathcal{S}$-DC is not competitive for any monotonic (non-decreasing or non-increasing) sequence of speeds. The intuitive reason is that servers move to the right either too slowly or too quickly: Imagine repeatedly requesting the same n points in some small interval away from the source, until $\mathcal{S}$-DC covers all n points. One case is that $\mathcal{S}$-DC spawns too slowly and is therefore defeated by an adversary covering these n positions immediately with n servers. In the other case, the adversary will also use n servers to cover the initial group of requests and then shift its group of servers slowly towards the source, always making requests at the new positions of these offline servers. As $\mathcal{S}$-DC tries to cover the new requests, it is tricked into spawning too many servers. Both cases lead to an unbounded competitive ratio.

The proof consists of several lemmas. The lemmas hold also for non-monotonic speeds and we use monotonicity only to easily combine the lemmas in the end.

A useful property of $\mathcal{S}$-DC is that its cost can be calculated using only the final positions of the servers.

Lemma 4.4. *Let $x_1 \geq x_2 \geq \ldots$ be the server positions of $\mathcal{S}$-DC after serving a sequence of requests. Then the cost paid is $\sum_{i=1}^{\infty} z_i x_i$ where*

$$z_1 = 1 \tag{5}$$

$$z_i = \frac{z_{i-1}}{s_i} + 1 + \frac{1}{s_i}. \tag{6}$$

Proof. The position x_i of each server can be written as $x_i = r_i - l_i$ where r_i and l_i are the cumulative distances traveled by that server while moving to the right and left, respectively. By definition of $\mathcal{S}$-DC, for all i, we have

$$l_i = \frac{r_{i+1}}{s_{i+1}},$$

since any right move (apart from the rightmost server) is accompanied by a left move of another server. Observe that the online cost is

$$\begin{aligned} cost &= \sum_{i=1}^{\infty} (r_i + l_i) = \sum_{i=1}^{\infty} \left(r_i + \frac{r_{i+1}}{s_{i+1}} \right) \\ &= \sum_{i=1}^{\infty} r_i + \sum_{i=2}^{\infty} \frac{r_i}{s_i} = r_1 + \sum_{i=2}^{\infty} r_i \left(1 + \frac{1}{s_i} \right). \end{aligned} \tag{7}$$

Similarly,

$$\begin{aligned} \sum_{i=1}^{\infty} z_i x_i &= \sum_{i=1}^{\infty} z_i (r_i - l_i) \\ &= \sum_{i=1}^{\infty} z_i r_i - \sum_{i=1}^{\infty} z_i \frac{r_{i+1}}{s_{i+1}} \\ &= z_1 r_1 + \sum_{i=2}^{\infty} r_i \left(z_i - \frac{z_{i-1}}{s_i} \right). \end{aligned} \tag{8}$$

By equating (7) and (8) term-by-term, we get the desired recurrence for z_i. □

The next lemma takes care of the case when online servers spawn too slowly.

Lemma 4.5. *If the speeds in $\mathcal{S}$ satisfy $\liminf_{n\to\infty} \sqrt[n]{\prod_{i=2}^{n} s_i} = 1$, then $\mathcal{S}$-DC is not competitive.*

Proof. For this lower bound, we have requests on n arbitrary positions in the interval $[1, 2]$, until $\mathcal{S}$-DC covers them all.

The optimal cost is at most $2n$. This can be achieved by spawning a fresh server for each requested position.

Since for every spawned online server we have $x_i \geq 1$, by Lemma 4.4 the online cost is $cost = \sum_{i=1}^{n} z_i x_i \geq \sum_{i=1}^{n} z_i$. Unraveling the recurrence, we get that $z_i = 1 + \frac{2}{s_i} + \frac{2}{s_i s_{i-1}} + \cdots + \frac{2}{s_i \cdot \ldots \cdot s_2}$. Thus,

$$\begin{aligned} cost &\geq n + \sum_{i=1}^{n-1} \sum_{j=1}^{n-i} \frac{2}{\prod_{k=j+1}^{j+i} s_k} \\ &\geq \sum_{i=1}^{f(n)} \sum_{j=1}^{n-i} \frac{2}{\prod_{k=j+1}^{j+i} s_k}, \end{aligned} \tag{9}$$

where

$$f(n) = \left\lfloor \frac{n}{2 + 2\log_2 \prod_{i=2}^{n} s_i} \right\rfloor \leq \frac{n}{2}.$$

We argue that for each $i = 1, \ldots, f(n)$, it holds for at least half of the values of $j = 1, \ldots, n - i$ that $\prod_{k=j+1}^{j+i} s_k \leq 2$. Indeed, suppose this were not the case for some i. Let us partition the set $J = \{1, \ldots, n - i\}$ of j-values into subsets $J_0, \ldots, J_{i-1}$, where J_m contains precisely those numbers from J that are congruent to m modulo i. By assumption, we have $\prod_{k=j+1}^{j+i} s_k > 2$ for at least half the values $j \in J$, so this must also be true for at least half the values $j \in J_m$ for some m. However, this would mean that

$$\prod_{k=2}^{n} s_k \geq \prod_{j \in J_m} \prod_{k=j+1}^{j+i} s_k > 2^{|J_m|/2} \geq 2^{\lfloor \frac{n-i}{i} \rfloor / 2} \geq 2^{\frac{n}{2f(n)} - 1} \geq \prod_{i=2}^{n} s_i,$$

a contradiction, because the second inequality is strict.

Thus, continuing from Equation (9), we can further bound the online cost as

$$cost \geq f(n) \frac{n - f(n)}{2} \geq \frac{n f(n)}{4}.$$

Since the optimal cost is at most $2n$, the competitive ratio is at least $f(n)/8$. However, $f(n)$ is unbounded because

$$\frac{n}{2 + 2\log_2 \prod_{i=2}^{n} s_i} = \frac{1}{\frac{2}{n} + 2\log_2 \sqrt[n]{\prod_{i=2}^{n} s_i}}$$

and the denominator in the last term gets arbitrarily close to 0. □

The case of servers being spawned too aggressively is handled by the following lemma:

Lemma 4.6. *If there exists an unbounded function $f(n)$ such that for each $k \in \mathbb{N}$ we have $\prod_{i=k}^{k+n} s_i \geq f(n)$, then $\mathcal{S}$-DC is not competitive. In particular, if $\liminf_{i \to \infty} s_i > 1$, then $\mathcal{S}$-DC is not competitive.*

Proof. Consider the setup of server and request locations depicted in Figure 2. We start by spawning n online servers grouped tightly, with the leftmost being at distance 1 from the source and a small gap δ between them. This is easily accomplished by repeating several requests on those points. Afterwards, we shift this group of n servers (by means of requests on new $n + 1$ points) to the left by v_1, chosen so the $n + 1$ points are covered exactly by the n old servers plus a newly spawned one, which occupies the leftmost requested position $1 - v_1 - \delta$.

Such a shift to the left is repeated again and again, shifting each time the leftmost n spawned servers a new distance v_k to the left via multiple requests on $n + 1$ positions. The goal each time is to pull a new server from the source *and* leave one behind forever, thus achieving an arbitrarily high competitive ratio for $\mathcal{S}$-DC variants that spawn servers too fast.

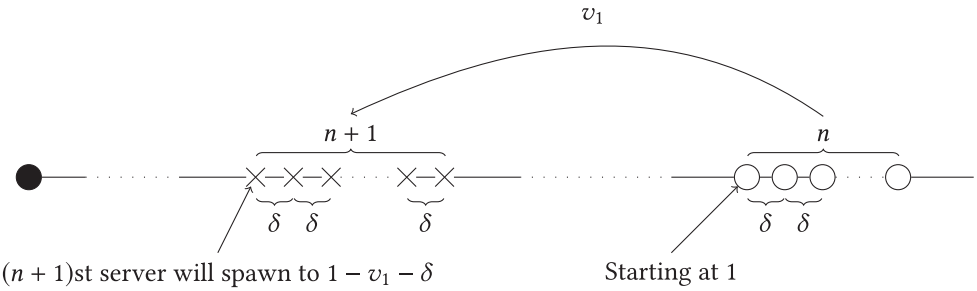

Fig. 2. Servers (denoted by circles) and request locations (denoted by crosses) in the first phase when $\mathcal{S}$-DC spawns too aggressively.

The offline cost can be calculated easily. The offline algorithm uses n servers to cover the first group of n requested points in the interval $[1, 1+n\delta]$. Then it adds one more server and moves the group of $n+1$ servers to the left to satisfy all of the following requests. At most, the group of offline servers will return close to the source, yielding an optimal cost of

$$OPT \le 2(n+1)(1+n\delta) = O(n), \tag{10}$$

where the last bound holds for δ sufficiently small.

To bound the online cost, we need to compute the values v_k first. Let ℓ_i^k and r_i^k denote the cumulative distance to the left and right, respectively, traveled by x_i during the left shift by v_k of the group $x_k, x_{k+1}, \ldots, x_{k+n-1}$. The nonzero values among these are

$$\begin{aligned}
\ell_k^k &= v_k \\
r_{k+1}^k &= v_k s_{k+1} \\
\ell_{k+1}^k &= v_k(1+s_{k+1}) \\
r_{k+2}^k &= v_k(s_{k+2}+s_{k+1}s_{k+2}) \\
\ell_{k+2}^k &= v_k(1+s_{k+2}+s_{k+1}s_{k+2}) \\
&\vdots \\
r_{k+n}^k &= v_k\left(s_{k+n}+s_{k+n-1}s_{k+n}+\cdots+\prod_{j=k+1}^{k+n} s_j\right) = v_k \sum_{i=k+1}^{k+n}\prod_{j=i}^{k+n} s_j.
\end{aligned} \tag{11}$$

However, the new position of the server x_{k+n} pulled from the source during these moves is $1-\sum_{i=1}^k v_i - k\delta$. Equating this with Equation (11) and solving for v_k yields (assuming that n is even)

$$v_k = \frac{1-\sum_{i=1}^k v_i - k\delta}{\sum_{i=k+1}^{k+n}\prod_{j=i}^{k+n} s_j} \le \frac{1}{\frac{n}{2}\prod_{j=k+\frac{n}{2}}^{k+n} s_j} \le \frac{2}{nf(\frac{n}{2})}.$$

We will calculate the number of repetitions before the left border of the group of servers (just) passes $\frac{1}{2}$. If l is the number of repetitions, then we have

$$\frac{1}{2} \le \sum_{k=1}^{l}(v_k+\delta) \le \frac{2l}{nf(\frac{n}{2})} + l\delta,$$

and for sufficiently small δ this means that

$$l \geq \frac{n}{5} f\left(\frac{n}{2}\right).$$

If we do $l-1$ repetitions, then each of them will pull a new server at least $1/2$ away from the source, resulting in an online cost of $\Omega(n) \cdot f(\frac{n}{2})$. As the offline cost is $O(n)$ and $f(n)$ is unbounded, the algorithm is not competitive. □

If the sequence of speeds s_i is monotonic (non-decreasing or non-increasing), then we have either $\lim_{i\to\infty} s_i = 1$, in which case Lemma 4.5 applies, or otherwise $\liminf_{i\to\infty} s_i > 1$ and Lemma 4.6 applies. In either case, the competitive ratio is unbounded:

Theorem 4.7. *Algorithm $\mathcal{S}$-DC is not competitive for any monotonic sequence of speeds.*

5 REDUCTION TO BOUNDED SPACES

In this section, we show a reduction from the ∞-server problem on general metric spaces to bounded subspaces. Specifically, a metric space can be partitioned into "rings" of points whose distance from the source is between r^n and r^{n+1}, where $r > 1$ is fixed and $n \in \mathbb{Z}$. We show that if the ∞-server problem is strictly ρ-competitive on each ring, then it is competitive on the entire metric space.

Theorem 5.1. *Let M be a metric space and $s \in M$ and let $r > 1$. For $n \in \mathbb{Z}$, let $M_n = \{s\} \cup \{p \in M \mid d(s,p) \in [r^n, r^{n+1})\}$. If for each n the ∞-server problem on (M_n, s) is strictly ρ-competitive, then on (M, s) it is strictly $\frac{3r-1}{r-1}\rho$-competitive.*

Proof. Let ALG_n be a ρ-competitive algorithm for the ∞-server problem on (M_n, s).

For a request sequence σ, let σ_n be the subsequence of requests in M_n. Let ALG be the algorithm for (M, s) that uses different servers for each of the subsequences σ_n and serves them independently according to ALG_n.

The total online cost is $ALG(\sigma) = \sum_n ALG_n(\sigma_n) \leq \rho \sum_n OPT(\sigma_n)$. To finish the proof, it suffices to show that

$$\sum_n OPT(\sigma_n) \leq \frac{3r-1}{r-1} OPT(\sigma). \tag{12}$$

Thus, we only need to analyze the offline cost. We do this for each offline server separately. Fix some offline server x. Let N_0 and N_1 be the minimal and maximal values of n such that x visits M_n. We can assume without loss of generality (by adding virtual points to the metric space) that whenever x moves from M_n to $M_{n'}$ for some $n < n'$, it travels across points $p_{n+1}, p_{n+2}, \ldots, p_{n'}$ with $d(s, p_i) = r^i$, and similarly for $n > n'$.

The movements of server x can be tracked by separate servers for the different sets M_n. We denote the servers responsible for M_n by $x_{n,1}, x_{n,2}, \ldots$ in the order in which they are spawned. When server x is in M_n, the last spawned server $x_{n,t}$ is exactly at the same position tracking the movement of x. When server x exits M_n at some point p at the boundary to M_{n-1} or M_{n+1}, this server $x_{n,t}$ freezes at p. When x later re-enters M_n at a point p' at the same boundary, then either $x_{n,t}$ moves to p' or a new server $x_{n,t+1}$ is spawned to p'—whichever is cheaper. The movement cost of the servers $x_{n,1}, x_{n,2}, \ldots$ can be partitioned into the *first spawn cost* to spawn $x_{n,1}$ at some (possibly virtual) point on the boundary of M_n and M_{n-1}, the *tracking cost* to follow the movement of x within M_n by the last spawned server $x_{n,t}$, and the *re-entering cost* incurred to relocate $x_{n,t}$ or spawn $x_{n,t+1}$ when x re-enters M_n.

The total tracking cost for all n is bounded by the distance traveled by x. The sum of first spawn costs for all n is $\sum_{n=N_0}^{N_1} r^n \leq \sum_{n=-\infty}^{N_1} r^n = r^{N_1+1}/(r-1)$, which is at most $\frac{r}{r-1}$ times the total distance travelled by x, because the latter is at least r^{N_1}.

Let us now consider the re-entering cost incurred when x enters M_n at p' after it previously exited at p. Then p and p' are at the boundary of M_n and M_{n+u} for $u \in \{-1, +1\}$. The re-entering cost is $\min\{d(p,p'), d(s,p')\}$. Let b be the distance traveled by x in M_{n+u} between the times when it is entered at p and when it is next exited. If this exiting is at p', then the re-entering cost is at most $d(p,p') \leq b$ by the triangle inequality. Otherwise, x exits M_{n+u} at a point p'' at the boundary of M_{n+u} and M_{n+2u}. If $u = 1$, then $b \geq d(p,p'') \geq d(s,p'') - d(s,p) = r^{n+2} - r^{n+1} = (r-1)r^{n+1}$ and the re-entering cost is at most $d(s,p') = r^{n+1}$. If $u = -1$, then $b \geq d(p,p'') \geq d(s,p) - d(s,p'') = r^n - r^{n-1} = \frac{r-1}{r} r^n$ and the re-entering cost is at most $d(s,p') = r^n$. In all cases, the re-entering cost is at most $\frac{r}{r-1} b$. Thus, the total re-entering cost of all servers x_n is at most $\frac{r}{r-1}$ times the total distance traveled by x.

Thus, the sum of first spawn, tracking and re-entering cost of the servers x_n is at most $\frac{3r-1}{r-1}$ times the distance traveled by x. This shows Equation (12), giving the statement of the theorem. □

The last theorem can also be slightly generalized to the case where instead of *strict* ρ-competitiveness, an additive term proportional to r^n is allowed. It is not difficult to show the following specialization for the line, where the premise can be weakened to require competitiveness only on a single interval:

Corollary 5.1. *Let $0 < a < b$. The ∞-server problem is competitive on the line if and only if it is competitive on $(\{0\} \cup [a,b], 0)$.*

Another consequence of Theorem 5.1 is a reduction to spaces where the source is at a uniform distance from all other points. This models the case of a fixed cost for "buying" new servers.

Corollary 5.2. *Suppose there exists ρ so the ∞-server problem is strictly ρ-competitive on any metric space where the distance from the source to any other point is the same. Then the ∞-server problem on general metric spaces is competitive.*

Proof. Follows from Theorem 5.1 by increasing the distance from s to the other points in M_n to r^{n+1}, making a multiplicative error of at most r. □

Because of the lower bound of Reference [7], we now know that such a ρ as required by the premise of the corollary does not exist.

6 OPEN PROBLEMS

As stated earlier, the question whether the ∞-server problem is competitive on every metric space was resolved negatively in Reference [7]. For trees of depth D, they show that the competitive ratio is at least $\Omega(\log D)$. It seems likely that narrowing the doubly-exponential gap to the $O(2^D \cdot D)$ upper bound from Theorem 3.1 would correspond to narrowing the gap for the (h,k)-server problem as a function of h, currently between the $\Omega(\log\log h)$ shown in Reference [7] and the $O(h)$ following from the unaugmented setting.

The fact that the competitive ratio of the ∞-server problem is finite on some metric spaces and infinite on others poses the question of how to identify for a given metric space which of the two is the case. Moreover, are there non-trivial upper bounds on the competitive ratio as a function of the aspect ratio? Another interesting question is to determine the *strict* competitive ratio on n-point metrics as a function of n.

The lower bounds from Reference [7] give the same value for both deterministic and randomized algorithms, whereas typically one may expect the deterministic competitive ratio to be exponentially larger than the randomized one. A combinatorial argument, extending our lower bounds for layered graphs to larger depths, may be a way to obtain larger lower bounds for the deterministic setting.

REFERENCES

[1] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. 2011. A polylogarithmic-competitive algorithm for the k-server problem. In *Proceedings of the 52nd Symposium on Foundations of Computer Science (FOCS'11)*. 267–276.

[2] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. 2015. A polylogarithmic-competitive algorithm for the k-server problem. *J. ACM* 62, 5 (2015), 1–49.

[3] Nikhil Bansal, Marek Eliás, Lukasz Jez, Grigorios Koumoutsos, and Kirk Pruhs. 2018. Tight bounds for double coverage against weak adversaries. *Theor. Comput. Syst.* 62, 2 (2018), 349–365.

[4] Nikhil Bansal, Marek Eliéš, Łukasz Jeż, and Grigorios Koumoutsos. 2019. The (h, k)-server problem on bounded depth trees. *ACM Trans. Alg.* 15, 2 (2019), 28.

[5] Yair Bartal and Eddie Grove. 2000. The harmonic k-server algorithm is competitive. *J. ACM* 47, 1 (2000), 1–15.

[6] Yair Bartal and Elias Koutsoupias. 2004. On the competitive ratio of the work function algorithm for the k-server problem. *Theoret. Comput. Sci.* 324, 2–3 (2004), 337–345.

[7] Marcin Bienkowski, Jaroslaw Byrka, Christian Coester, and Lukasz Jez. 2020. Unbounded lower bound for k-server against weak adversaries. In *Proceedings of the 52nd ACM SIGACT Symposium on Theory of Computing (STOC'20)*. 1165–1169.

[8] Allan Borodin and Ran El-Yaniv. 2005. *Online Computation and Competitive Analysis*. Cambridge University Press.

[9] Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. 2018. k-server via multiscale entropic regularization. In *Proceedings of the 50th ACM SIGACT Symposium on Theory of Computing (STOC'18)*. 3–16.

[10] Niv Buchbinder, Anupam Gupta, Marco Molinaro, and Joseph (Seffi) Naor. 2019. k-servers with a smile: Online algorithms via projections. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms (SODA'19)*. 98–116.

[11] Marek Chrobak, Howard Karloff, Tom Payne, and Sundar Vishwanathan. 1991. New results on server problems. *SIAM J. Disc. Math.* 4, 2 (1991), 172–181.

[12] Marek Chrobak and Lawrence L. Larmore. 1991. An optimal on-line algorithm for k servers on trees. *SIAM J. Comput.* 20, 1 (1991), 144–148.

[13] Marek Chrobak and Lawrence L. Larmore. 1992. The server problem and on-line games. In *On-line Algorithms (DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol.7)*. Citeseer.

[14] Christian Coester, Elias Koutsoupias, and Philip Lazos. 2017. The infinite server problem. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP'17)*. 14:1–14:14.

[15] Ilan R. Cohen, Alon Eden, Amos Fiat, and Łukasz Jeż. 2014. Pricing online decisions: Beyond auctions. In *Proceedings of the 26th ACM-SIAM Symposium on Discrete Algorithms (SODA'15)*. SIAM, 73–91.

[16] János Csirik, Csanád Imreh, John Noga, Steve S. Seiden, and Gerhard J. Woeginger. 2001. Buying a constant competitive ratio for paging. In *Proceedings of the European Symposium on Algorithms (ESA'01)*. Springer, 98–108.

[17] Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. 2009. On the additive constant of the k-server work function algorithm. In *Proceedings of the International Workshop on Approximation and Online Algorithms (WAOA'09)*. Springer, 128–134.

[18] Amos Fiat, Yuval Rabani, and Yiftach Ravid. 1990. Competitive k-server algorithms (extended abstract). In *Proceedings of the 31st Symposium on Foundations of Computer Science (FOCS'90)*. 454–463.

[19] Sandy Irani and Ronitt Rubinfeld. 1991. A competitive 2-server algorithm. *Inform. Proc. Lett.* 39, 2 (1991), 85–91.

[20] Elias Koutsoupias. 1999. Weak adversaries for the k-server problem. In *Proceedings of the 40th Symposium on Foundations of Computer Science (FOCS'99)*. 444–449.

[21] Elias Koutsoupias. 2009. The k-server problem. *Comput. Sci. Rev.* 3, 2 (2009), 105–118.

[22] Elias Koutsoupias and Christos Papadimitriou. 1996. The 2-evader problem. *Inform. Proc. Lett.* 57, 5 (1996), 249–252.

[23] Elias Koutsoupias and Christos H. Papadimitriou. 1995. On the k-server conjecture. *J. ACM* 42, 5 (1995), 971–983.

[24] James R. Lee. 2018. Fusible HSTs and the randomized k-server conjecture. In *Proceedings of the 59th IEEE Symposium on Foundations of Computer Science (FOCS'18)*. 438–449.

[25] Mark Manasse, Lyle McGeoch, and Daniel Sleator. 1988. Competitive algorithms for on-line problems. In *Proceedings of the 20th ACM Symposium on Theory of Computing (STOC'88)*. 322–333.

[26] Daniel D. Sleator and Robert E. Tarjan. 1985. Amortized efficiency of list update and paging rules. *Commun. ACM* 28, 2 (1985), 202–208.
[27] Neal Young. 1994. The k-server dual and loose competitiveness for paging. *Algorithmica* 11, 6 (1994), 525–541.
[28] Neal E. Young. 1991. On-line caching as cache size varies. In *Proceedings of the 2nd ACM/SIGACT-SIAM Symposium on Discrete Algorithms (SODA'91)*. 241–250.

Received January 2020; revised March 2021; accepted March 2021