Randomized k-Server in Polynomial Time

Christian Coester
University of Oxford, UK

Romain Cosson
New York University, NY, USA

Abstract

We study the design of computationally efficient randomized algorithms for the k-server problem. Existing randomized algorithms with the best known competitive ratios are, on the one hand, inherently implicit and, on the other hand, employ a rounding scheme that maintains a distribution over exponentially many configurations. In this work, we introduce a derandomization framework that transforms any randomized k-server algorithm on a hierarchically separated tree into one that uses only $O(\log k)$ random bits for request sequences of arbitrary length – hence maintaining a distribution over only polynomially many server configurations. Leveraging this black-box derandomization, we obtain the first polynomial-time randomized k-server algorithm on arbitrary n-point metrics with a polylogarithmic competitive ratio. Our results also have implications for the advice complexity of the k-server problem.

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

Keywords and phrases k-server, online algorithms, computational complexity, randomized complexity, advice complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2026.65

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2605.01497`

Funding *Christian Coester*: Funded by the European Union (ERC, CCOO, 101165139). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

Acknowledgements We thank the anonymous reviewers for their constructive feedback.

1 Introduction

The k-server problem is a cornerstone in online algorithms, sometimes described as the "holy grail of competitive analysis", and many techniques originally developed for k-server went on to impact various other problems in the field. While there has been limited progress on deterministic algorithms for the problem since a competitive ratio of $2k-1$ was established in [29], the study of randomized algorithms has seen recent breakthroughs, culminating in an $\mathcal{O}(\log n \log^2 k)$-competitive randomized algorithm on n-point metrics via an $\mathcal{O}(\log^2 k)$-competitive algorithm on hierarchically separated trees (HSTs) [15], and an $\Omega(\log^2 k)$ lower bound refuting the randomized k-server conjecture [14].

Despite the improved information-theoretic understanding of the problem, competitive algorithms that are also computationally efficient are lacking. Among algorithms for general metrics with polynomial (in the metric space encoding size) per-request running time, the best known competitive ratio is exponential in k, achieved by the randomized Harmonic algorithm [25, 6]. The work function algorithm [29], which achieves the best deterministic competitive ratio, has time-per-request either exponential in k or as an increasing function in the number of preceding requests (i.e., growing unbounded over time) [31].


For modern randomized algorithms, it is unclear how to even implement them at all while retaining competitiveness. Although the k-server problem is a discrete combinatorial online problem, recent progress relies on continuous methods such as existence results for solutions of differential inclusions, leading to implicit algorithms. The lack of efficiency of the algorithm in [15] was pointed out explicitly in Bubeck's TCS+ talk [13, minute 40:50]. This algorithm is defined by a continuous (both in time and space) projected gradient flow. A first step towards making the approach explicit – and towards obtaining a discrete algorithm that could be implemented – was taken by [17], which replaced the continuous gradient flow with a sequence of Bregman projections, effectively discretizing the algorithm in time. In this paper, we address the problem of discretization in space. Specifically, we show that any randomized k-server algorithm on hierarchically separated trees (HSTs) can be transformed into an algorithm that uses a finite number of random bits (more precisely, $\mathcal{O}(\log k)$ random bits, regardless of the length of the request sequence). While the basic idea of space discretizations has been present already in variants of paging [1, 5], prior techniques do not extend to k-server, as we discuss in Section 1.3.

Our derandomization resolves two problems that hinder the implementability of current algorithms: first, that they require computing the exact solutions of a sequence of convex optimization problems, whereas finite time/space iterative methods can only provide approximate solutions; second, that they maintain a probability distribution over exponentially many server configurations (specifically, up to $\binom{n}{k}$ server configurations). In contrast, our approach reduces the cardinality of the support of a randomized algorithm to $\mathcal{O}(k^2)$ – a quantity that is independent of both the number of points in the metric space and the number of requests. This leads us to the first polynomial-time randomized k-server algorithm that achieves a polylogarithmic competitive ratio, matching the state-of-the-art competitiveness guarantee of [15, 17]. Our results also have implications for the so-called "advice complexity" of the k-server problem [10].

k-server problem. The problem is defined for a metric space $\mathcal{M} = (M, d)$ with $n \geq k$ points. The input is a sequence of points $\rho(\cdot) = (\rho(1), \rho(2), \dots)$, where $\rho(t) \in M$ is called the *request* at time $t \in \mathbb{N}$. The output of the problem is a sequence of server configurations $\mathcal{C}(\cdot) = (\mathcal{C}(1), \mathcal{C}(2), \dots)$, where $\mathcal{C}(t)$ is a subset of k points of $\mathcal{M}$ that represent the position of k servers at time t. The output must serve all the requests, meaning that it should satisfy $\forall t\colon \rho(t) \in \mathcal{C}(t)$.

Deterministic algorithms. A deterministic (online) algorithm $\mathcal{C}(\cdot)$ for the k-server problem is a function that maps a sequence of revealed requests $\rho(\leq t) = (\rho(1), \dots, \rho(t))$ to a configuration $\mathcal{C}(\rho(\leq t)) \in M^k$ which contains the current request $\rho(t)$. In slight abuse of notation, we denote the configuration $\mathcal{C}(\rho(\leq t))$ by $\mathcal{C}(t)$. The cost of a deterministic algorithm $\mathcal{C}(\cdot)$ on the input sequence $\rho(\cdot)$ is defined by

$$\texttt{Cost}(\mathcal{C}(\cdot), \rho(\cdot)) = \sum_t d(\mathcal{C}(t), \mathcal{C}(t+1)),$$

where the distance between two configurations $d(\mathcal{C}, \mathcal{C}')$ is defined as the minimum weight of a matching between $\mathcal{C}$ and $\mathcal{C}'$.

Randomized algorithms. A randomized (online) algorithm $\mathcal{R}(\cdot)$ is a probability distribution over deterministic (online) algorithms. The cost of a randomized algorithm on an instance is the expected cost on that instance. For a parameter $m \in \mathbb{N}$, an *m-barely random algorithm* is defined as the uniform distribution over m deterministic algorithms $\mathcal{R}(\cdot) = (\mathcal{C}_1(\cdot), \dots, \mathcal{C}_m(\cdot))$, and its cost on an instance $\rho(\cdot)$ is therefore given by $\texttt{Cost}(\mathcal{R}(\cdot), \rho(\cdot)) = \frac{1}{m} \sum_{j \in [m]} \texttt{Cost}(\mathcal{C}_j(\cdot), \rho(\cdot))$. Note that if $m = 2^b$ for an integer b, an m-barely random algorithm can be sampled with just b random bits, independently of the length of the input sequence–hence the term *barely* random algorithm, introduced in [32].

Advised algorithm. An online algorithm with advice $\mathcal{A}(\cdot)$ is a deterministic algorithm whose input is augmented with some bits of advice supposedly provided by a third party that knows the problem instance. For any parameter $m \in \mathbb{N}$, we define an m-advised algorithm $\mathcal{A}(\cdot)$ as a collection of m deterministic algorithms $\mathcal{A}(\cdot) = (\mathcal{C}_1(\cdot), \dots, \mathcal{C}_m(\cdot))$. The cost of an m-advised algorithm is defined as $\texttt{Cost}(\mathcal{A}(\cdot), \rho(\cdot)) = \min_{j\in[m]} \texttt{Cost}(\mathcal{C}_j(\cdot), \rho(\cdot))$, and is therefore not greater than its barely random counterpart.

Competitive analysis. The performance of an online algorithm for the k-server problem is evaluated by its competitive ratio. We say that an online algorithm $\mathcal{A}(\cdot)$ is c-competitive if it satisfies for any input $\rho(\cdot)$ that

$$\texttt{Cost}(\mathcal{A}(\cdot), \rho(\cdot)) \leq c \cdot \mathrm{OPT}(\rho(\cdot)) + c_0, \tag{1}$$

where c is a positive real number, $\mathrm{OPT}(\rho(\cdot)) = \inf_{\mathcal{C}(\cdot)} \texttt{Cost}(\mathcal{C}(\cdot), \rho(\cdot))$ is called the offline optimum and c_0 is a constant independent of the sequence of requests.

Main result. The main result of this paper (Theorem 1) is a polynomial-time randomized k-server algorithm that enjoys state-of-the-art competitive guarantees. The main technical ingredient for this result is a polynomial-time derandomization scheme (Proposition 5 and Proposition 6) transforming a c-competitive randomized k-server algorithm on hierarchically separated trees (HSTs) into an $\mathcal{O}(c)$-competitive k-server algorithm that uses only $\mathcal{O}(\log k)$ random bits. As detailed in the related works section below, this framework extends previous results on the randomized complexity [11] and the advice complexity [21, 10] of the k-server problem.

1.1 Previous work

The k-server problem was introduced by [30] as a general framework for online problems. This initial work focused on the competitive ratio of deterministic algorithms, establishing a universal lower bound of k and conjecturing that a k-competitive algorithm exists for all metric spaces. The problem quickly became central in the domain of online algorithms, and, years after, the deterministic analysis of the k-server problem was nearly settled by [29], which proved that the work function algorithm is $(2k-1)$-competitive in all metric spaces. In this paper, we are primarily concerned with the randomized and the fractional variants of the k-server problem, and we refer to [28] for details on the deterministic variant.

The first randomized analysis of the k-server problem was by [23]. They introduced a simple (and discrete) "marking" algorithm that is $2H_k$-competitive on the uniform metric space (equivalent to the paging problem). In contrast, going beyond the uniform metric space later required the introduction of the fractional variant of the k-server problem. The idea is to have the algorithm maintain a (fractional) measure over the metric space, of total mass k, representing for each point the probability that a server is located at that point. This effectively allows the algorithm to move probability mass continuously in space, enabling the use of new methods. Any randomized algorithm straightforwardly induces a fractional algorithm, but a reciprocal transformation is only known for the line metric [20] and hierarchically separated trees (HSTs) [2] (as well as metrics that embed into HSTs with constant distortion, such as weighted stars, for which this result was established first [3]). This typically requires a nontrivial "rounding" step. The idea of studying the fractional variant of the k-server problem was first used by [9] to obtain refined randomized guarantees on the uniform metric space. In subsequent breakthroughs, first [3] used the approach to obtain an $\mathcal{O}(\log k)$-competitive algorithm on weighted stars with a primal-dual algorithm.

After, [2] obtained an $\tilde{\mathcal{O}}(\log^3 n \log^2 k)$-competitive algorithm n-point metrics, based on a fractional algorithm on HSTs. To date, the best randomized competitive ratio is due to [15], which presented an $\mathcal{O}(\log^2 k)$-competitive algorithm for HSTs. The focus on hierarchically separated trees is theoretically motivated by random tree embeddings [22], which allow to embed any n-point metric space into a hierarchically separated tree with a distortion of $\mathcal{O}(\log n)$. Therefore, the result of [15] directly translates into an $\mathcal{O}(\log n \log^2 k)$-competitive algorithm on arbitrary metrics. The question whether an $o(k)$-competitive randomized algorithm exists for the k-server problem on arbitrary metrics remains open.

The study of randomized complexity in online algorithms started with the work of [32] on the list update problem. They noticed that a finite number of random bits (independent of the length of the request sequence) is sufficient to improve the competitive ratio from 2 to 1.75 and coined the term "barely random algorithm". The notion is also covered in [12, Chap. 2]. In the case of the k-server problem, a result of this type is in [11, 27] (see also [26, Chap. 3]) and is restricted to the uniform metric space. Their result essentially says that there exists an $\mathcal{O}(\log k)$-competitive algorithm for paging that requires only $\mathcal{O}(\log k)$ random bits. This can be recovered as a consequence of our derandomization method since there exist $\mathcal{O}(\log k)$-competitive randomized algorithms on the uniform metric space [23]. Other works, such as [21, 10, 33], study the randomized complexity or advice complexity of the k-server problem, but in a weaker regime where the number of bits of advice or the number of random bits is allowed to scale with the size of the problem instance. For example, [21] introduces a deterministic $\mathcal{O}(\sqrt{k})$-competitive algorithm for the k-server problem that uses $\mathcal{O}(1)$ bits of advice *per request*.

One important technical tool that we use throughout the paper is the notion of m-barely fractional algorithms, which allows us to recover some of the power of fractional algorithms, while enforcing a discretization in space. Informally, an m-barely fractional algorithm is one that moves server mass in fractions that are multiples of $1/m$, and consequently, the size of its support is bounded by km. Furthermore, applying the rounding of [2], the cardinality of the support of the distribution over deterministic algorithms of the corresponding randomized algorithm can be restricted to m. We note that barely fractional algorithms are already instrumental in previous works, such as in [1, 5], which study variants of the k-server problem on weighted stars (cf. Section 1.3 for further discussion), and in [18], which studies a multi-agent version of metrical task systems.

1.2 Preliminaries

For $x \in \mathbb{R}$, we write $x^+ := \max\{x, 0\}$.

Weighted trees. Let V be the set of nodes of a rooted edge-weighted tree. We denote by $r \in V$ the root and by $L \subseteq V$ the set of leaves. For a node $u \in V \setminus \{r\}$, we denote by $p(u)$ its parent and by w_u the weight of the edge between u and $p(u)$. For a node $u \in V$, we denote by C_u the set of its children and by L_u the set of leaves in the subtree rooted at u.

Any weighted tree induces a metric space on its leaves L,[1] where the distance between two leaves is defined as the sum of edge weights on their connecting path.

[1] We do not consider internal nodes to be points of the metric space, and in particular we assume that k-server requests never come at internal nodes. This is without loss of generality, as new leaves could be attached to internal nodes with edges of length 0.

For $\tau \geq 1$, a τ-HST (hierarchically separated tree) is the metric space induced by a tree whose edge weights satisfy $w_u = \tau w_v$ for all $u \in V \setminus \{r\}$ and $v \in C_u$, and where all leaves are at the same depth. It is well-known that any n-point metric space can be probabilistically embedded into an HST such that all distances are distorted by a factor $O(\log n)$ in expectation [22].

Inner and leaf measures. An *inner measure* of total mass k on a tree with nodes V is a vector $\mathbf{z} \in \mathcal{Z}$, where

$$\mathcal{Z} = \left\{ \mathbf{z} \in \mathbb{R}_+^V \text{ s.t. } z_r = k \text{ and } \forall u \in V \colon z_u \geq \sum_{c \in C_u} z_c \right\}.$$

Here, z_u represents the amount of server mass in the subtree rooted at u. We write $z_{[u]} = z_u - \sum_{c \in C_u} z_c$ for the mass at u itself. If $z_{[u]} = 0$ for every $u \in V \setminus L$, then $\mathbf{z}$ is called a *leaf measure*. Thus, a leaf measure is a measure of mass k on the points L of the associated metric space, whereas an inner measure may have some of the mass on internal nodes, which we do not consider part of the metric space. We say that a measure is *m-barely fractional* if it only takes values in $\frac{1}{m}\mathbb{N} = \{0, \frac{1}{m}, \frac{2}{m}, \dots\}$.

Optimal transport. We define the transport distance between two measures $\mathbf{z}, \mathbf{z}' \in \mathcal{Z}$ by

$$\mathrm{OT}(\mathbf{z}, \mathbf{z}') = \sum_{u \in V \setminus \{r\}} w_u |z_u - z'_u|.$$

This definition matches the classical interpretation of optimal transport for measures on trees.

Fractional algorithm. A fractional (online) k-server algorithm on a tree metric L is a function $\mathbf{z}(\cdot)$ that takes as input a sequence $\rho(\leq t)$ and produces as output a leaf measure denoted by $\mathbf{z}(t)$, with total mass $\sum_{u \in L} z_u(t) = k$ and that services requests by satisfying $z_{\rho(t)}(t) = 1$. The cost of a fractional algorithm $\mathbf{z}(\cdot)$ on some instance $\rho(\cdot)$ is defined by

$$\texttt{Cost}(\mathbf{z}(\cdot), \rho(\cdot)) = \sum_t \mathrm{OT}(\mathbf{z}(t+1), \mathbf{z}(t)).$$

A fractional algorithm $\mathbf{z}(\cdot)$ is m-barely fractional if it only outputs m-barely fractional measures.

Computational complexity of algorithms. To properly define computational complexity, we restrict our attention to metric spaces where all distances are integral. This assumption is without loss of generality, up to arbitrary precision, because the k-server problem is scale-invariant. We use D to denote the diameter of the underlying metric space. Thus, distances are encoded in words of $O(\log D)$ bits. We aim for algorithms that process each new request in time polynomial in $\log D$ and the number of points n in the metric space.

1.3 Why past approaches do not extend to k-server

The idea of working with barely fractional algorithms is already present in [1], which applies it to weighted/generalized paging, corresponding to star metrics for k-servers. In this case, a barely fractional algorithm can be achieved very simply by rounding each (fully fractional)

evicted page mass x to $f(x) = \frac{1}{k}\lfloor \min(2kx, k) \rfloor$. However, discretizations of this form are not competitive for k-server, because tiny fluctuations in x may lead to large movements in $f(x)$ (because $\lfloor \cdot \rfloor$ is discontinuous!). This issue does not arise on stars, where the evicted page mass x is decreased only when set to 0, but on trees, it occurs at internal nodes. In fact, all approaches of the form $x \mapsto f(x)$ for a fixed function $f(\cdot)$ are condemned to fail.

Another possible perspective on the rounding of [1] for star metrics is to assume that the root can hold server mass and to consider the modified algorithm in which upward movements (from a leaf to the root) are twice those of the base algorithm[2] and where requests are served by pulling server mass down from the root to the requested leaf. One can show that the modified algorithm is valid and suffers only a factor-of-2 loss in the movement cost. Furthermore, this transformation naturally extends to metric spaces that can be embedded as the leaves of a tree, by assuming that all non-leaf nodes can hold server mass and that requests are served by pulling server mass down from the nearest ancestors of the requested leaf. In star metrics, the cumulative movements of the modified algorithm can be rounded down to the nearest multiple of $1/k$, yielding exactly the rounding of [1]. But in trees, that simple rounding is no longer possible: Consider a node u with two children, each of which sends mass $\frac{1}{2k}$ to its grandparent in the accelerated algorithm (i.e., to $p(u)$). Then the mass traveling from u to $p(u)$ is $\frac{1}{k}$, so we would want to send the same mass after discretization. But each movement from a child of u to u is only $\frac{1}{2k}$, which might be insufficient to cross the next multiple of $\frac{1}{k}$. Hence, the discretized algorithm would not have any movement into u that could be sent further upwards to $p(u)$.

We resolve these kinds of issues via a technique inspired by physics that we can refer to as *hysteresis* (see Section 2). Roughly, it ensures that the barely fractional configuration only changes when the proximity to the target configuration improves a lot, i.e., when moving is really worth it.

The question of computational complexity of variants of weighted paging was also raised in a more recent paper of Bansal et al. [5], which proposed a polynomial-time algorithm for a generalization of weighted paging at the expense of an overall competitiveness of $O(\log^2 k)$ instead of $O(\log k)$.

1.4 Exposition of the results and outline

Our main result is the following.

▶ **Theorem 1.** *There exists an $\mathcal{O}(\log^2 k)$-competitive randomized k-server algorithm for τ-HST metrics with $\tau \geq 10$ that serves requests in polynomial time in $(n, \log D)$. The algorithm uses only $O(\log k)$ random bits, regardless of the length of the request sequence.*

By classical polynomial time random HST embeddings [22], we obtain the following extension of the theorem to general metrics.

▶ **Corollary 2.** *There exists a randomized $\mathcal{O}(\log n \log^2 k)$-competitive k-server algorithm on arbitrary n-point metrics that serves requests in polynomial time.*

We note that this algorithm for general metrics is still barely random as it only samples a bounded number of random bits at the start of the program. Most of these bits are used to compute the HST embedding.

By observing that bits of advice are at least as useful as random bits (see discussion above), we also obtain the following corollary.

[2] Movements in the modified algorithm stop when there is no mass available at the source.

▶ **Corollary 3.** *For any τ-HST metric, with $\tau \geq 10$, there exists a deterministic k-server algorithm with advice that is $\mathcal{O}(\log^2 k)$-competitive and that only uses $\mathcal{O}(\log k)$ bits of advice.*

We now streamline the main arguments leading to Theorem 1. We start with the fractional algorithm of [17], that can be adapted to run in polynomial time albeit at the expense of an additive penalty, as stated below.

▶ **Proposition 4.** *Let $\epsilon > 0$. There exists a fractional k-server algorithm $\mathbf{z}(\cdot)$ for τ-HST metrics with $\tau \geq 10$ that serves each request in polynomial time in $(n, \log 1/\epsilon, \log D)$, and satisfying for all requests sequences $\rho(\cdot)$*

$$\texttt{Cost}(\mathbf{z}(\cdot), \rho(\cdot)) \leq \mathcal{O}(\log^2 k)\mathrm{OPT}(\rho(\cdot)) + \epsilon T + O(D \log^2 k),$$

where T is the length of the sequence $\rho(\cdot)$.

Note that this fractional algorithm is not competitive in the sense of Equation (1). The seemingly innocuous additive term in ϵT is, in fact, quite inconvenient because a malicious adversary can make the additive term ϵT very large, while keeping the offline cost bounded. Specifically, consider the case of a $(k+1)$-point metric with k points close to one another, and one far-away point, and assume only the k nearby points are requested. A natural fractional algorithm (and indeed the algorithm in [17]) would send only a very small (smaller than ϵ) fractional mass from the far-away point to the group of k points, per request. Therefore, a polynomial-time implementation that approximates fractional mass to within precision ϵ in each step would never move any mass from the far-away point to the group of k points, and thus, would not have a finite competitive ratio. The other crucial issue with this fractional algorithm is that, under the rounding of [2], it induces a randomized algorithm with exponential support, thereby losing its computational efficiency. We deal with both issues by discretizing the fractional algorithm (Proposition 5) before applying the rounding of [2] (Proposition 6). For further generality, the statements of the propositions are extended in Proposition 5 to arbitrary trees, and, in Proposition 6, to all metric spaces for which a rounding scheme is known. The combination of Proposition 5 and Proposition 6 entails our derandomization scheme.

▶ **Proposition 5** (Fractional to barely fractional)**.** *Let $m \geq 2k^2 + k$. Any fractional online algorithm for k-server on trees can be converted online into an m-barely fractional algorithm with at most a polynomial additive overhead in running time and a constant factor multiplicative overhead in cost.*

▶ **Proposition 6** (Barely fractional to barely randomized)**.** *For k-server on a line metric or τ-HSTs with $\tau > 5$, any m-barely fractional algorithm can be converted online into an m-barely random algorithm with at most a polynomial additive overhead in running time and a constant factor multiplicative overhead in cost.*

It is clear from the statement of these propositions why they lead to a polynomial time randomized algorithm (provided that $1/\epsilon$ and m are sufficiently "small"), but it is not immediately obvious why they also solve the issue that was initially raised by the additive term. For this, we note that we can pick $\epsilon = \Theta\left(\frac{1}{m}\right)$, and then directly apply the following result.

▶ **Proposition 7.** *Suppose that there is an m-barely fractional algorithm $\mathbf{z}(\cdot)$ satisfying for any request sequence $\rho(\cdot)$,*

$$\texttt{Cost}(\mathbf{z}(\cdot), \rho(\cdot)) \leq c\,\mathrm{OPT}(\rho(\cdot)) + \frac{1}{2m}T + c_0,$$

where T is the number of time-steps in the input sequence $\rho(\cdot)$, c_0 is a constant independent of $\rho(\cdot)$, and each non-zero distance is at least 1. Then the variant of $\mathbf{z}(\cdot)$ that ignores any request that it can serve without moving is $2c$-competitive.

We prove Propositions 5 and 7 in Section 2, Proposition 6 in Section 3, and Proposition 4 in Section 4.

2 Barely fractional k-server

In this section, we prove the main technical step of this paper of going from fractional to barely-fractional algorithms. Before presenting the proof, we start with a general discussion of how *hysteresis* can be used in competitive analysis, motivating our approach with a simple example inspired by dynamic data structures.

▶ **Proposition 5** (Fractional to barely fractional). *Let $m \geq 2k^2 + k$. Any fractional online algorithm for k-server on trees can be converted online into an m-barely fractional algorithm with at most a polynomial additive overhead in running time and a constant factor multiplicative overhead in cost.*

▶ Remark 8 (Collective interpretation of m-barely fractional algorithms). One way to interpret m-barely fractional k-server is as an online problem with mk deterministic agents, where each incoming request must be served by moving m agents to the requested point. This perspective connects to the metric allocation problem [4], which generalizes k-server by allowing a broader class of requests. It also relates to collective metrical task systems [18], where a team of m agents jointly faces a metrical task system, and to collective tree exploration [19], where a team of k agents must explore an unknown tree as quickly as possible.

Warm-up: competitiveness via hysteresis. Hysteresis is a phenomenon where a system's present state depends on its history. It is a common theme in physics, specifically in electromagnetism, where it describes the fact a magnet can resist a (mildly) opposing magnetic field.

In computer science, the concept appears in dynamic data structures [7], and in real-world implementations of resizable arrays (see e.g., [8, 34]). Specifically, consider the situation where a user wishes to store at time t an array of size $y(t) \in \mathbb{N}$. Since reallocating memory space is costly, a common technique is to allocate at any time t an array of size $x(t) \geq y(t)$, where the value of $x(t)$ is more "stable" than the value of $y(t)$ (i.e., it resists change). A typical update rule for $x(t)$ is as follows:

$$\begin{cases} \text{If } y(t) \geq x(t-1) \text{ then set } x(t) = 2x(t-1). \\ \text{If } y(t) \leq x(t-1)/4 \text{ then set } x(t) = x(t-1)/2. \end{cases} \tag{2}$$

Basic arguments involving geometric sums demonstrate the approach's competitiveness. As a warm-up to our proof of Proposition 5, we propose a new perspective on the update rule described in Equation (2), which can equivalently be formulated as

$$x(t) = \operatorname*{arg\,min}_{x \in \{2^i \,:\, i \in \mathbb{N}\}} |x - x(t-1)| + |x - 2y(t)|, \tag{3}$$

where ties are broken in favor of the solution maximizing $|x - x(t-1)|$. The idea is that the first term $|x - x(t-1)|$ resists changes in the value of x, whereas the other term $|x - 2y(t)|$

encourages x to take a value close to $2y$. We now show the competitiveness of Equation (3) without using geometric sums. By optimality of $x(t)$ in (3) we have that

$$|x(t) - x(t-1)| + |x(t) - 2y(t)| = |x(t-1) - 2y(t)|. \tag{4}$$

Subtracting $|x(t-1)-2y(t-1)|$ from both sides of Equation (4), and using on the right-hand side that $|x(t-1)-2y(t)|-|x(t-1)-2y(t-1)| \leq 2|y(t-1)-y(t)|$, and on the left-hand side, the telescopic sum $\sum_{t\in[T]} |x(t)-2y(t)|-|x(t-1)-2y(t-1)| = |x(T)-2y(T)|-|x(0)-2y(0)| \geq 0$, we get,

$$\sum_{t\in[T]} |x(t) - x(t-1)| \leq 2 \sum_{t\in[T]} |y(t) - y(t-1)|,$$

and thus, the competitiveness of $\boldsymbol{x}(\cdot)$. This analysis can be extrapolated to more complex settings, as we will see in the proof below. Specifically, the reader will notice the parallel between Equation (3) and our discretization technique for k-server in Equation (5), where the absolute value is replaced with an optimal transport cost, and where the set $\{2^i : i \in \mathbb{N}\}$ is replaced by a set of barely fractional configurations.

Proof of Proposition 5. Let $\boldsymbol{x}(\cdot)$ be the given fractional algorithm, and assume without loss of generality that it is initialized with an integral configuration $\boldsymbol{x}(0)$. The proof goes through a sequence of steps that progressively go from the initial fully fractional strategy $\boldsymbol{x}(\cdot)$ to the desired barely fractional strategy $\boldsymbol{y}(\cdot)$:

1. First, we define a sequence of inner measures $\mathbf{z}^{(1)}(\cdot)$ of total mass k, with a movement cost at most twice the movement cost of $\boldsymbol{x}(\cdot)$, and satisfying at all times and for all nodes $u \in V$: $z_u^{(1)} > 0 \implies x_u \geq 1/2$ as well as $x_u \geq 1 \implies z_u^{(1)} \geq 1$, where we drop the dependency on t for convenience.
2. Next, for $m' = 2m + 2k + 1$, we define a sequence of m'-barely fractional inner measures $\mathbf{z}^{(2)}(\cdot)$ of total mass k, of movement cost at most the movement cost of $\mathbf{z}^{(1)}(\cdot)$, and satisfying at all times and for all nodes $u \in V$: $z_u^{(2)} \geq z_u^{(1)} - (2k+1)/m'$.
3. Then, we define a sequence of $2m$-barely fractional inner measures $\mathbf{z}^{(3)}(\cdot)$ of total mass in the interval $[k, k+1/2]$ (the same mass in each step), of movement cost at most twice the movement cost of $\mathbf{z}^{(2)}(\cdot)$ and satisfying at all times, and for all nodes $u \in V$: $x_u \geq 1 \implies z_u^{(3)} \geq 1$.
4. After, we define a sequence of m-barely fractional inner measures $\mathbf{z}^{(4)}(\cdot)$ of total mass k, of movement cost at most twice the movement cost of $\mathbf{z}^{(3)}(\cdot)$, and satisfying at all times and for all nodes $u \in V$: $x_u \geq 1 \implies z_u^{(4)} \geq 1$.
5. Finally, observing that all prior steps can be performed online, we obtain our desired m-barely fractional online algorithm by converting the sequence of inner measures $\mathbf{z}^{(4)}(\cdot)$ into a sequence of leaf measures.

Step 1. We consider the function $\sigma : \mathbb{R}_+ \to \mathbb{R}_+$ such that $\sigma|_{[\ell,\ell+\frac{1}{2}]} = \ell$ for every $\ell \in \mathbb{Z}_+$ and σ is extended affinely to the rest of $\mathbb{R}_+$. Equivalently, $\forall x \in \mathbb{R}_+ : \sigma(x) = \lfloor x \rfloor + 2(x - \lfloor x \rfloor - 1/2)^+$. The definition is borrowed from [15]. We consider the strategy $\mathbf{z}^{(1)}(\cdot)$ defined at all times by $\mathbf{z}^{(1)}(t) = \sigma(\boldsymbol{x}(t))$ where $\sigma(\cdot)$ is applied element-wise (including to inner nodes of the tree). Note that $\mathbf{z}^{(1)}(\cdot)$ always defines an inner measure of total mass k because $\sigma(\cdot)$ is superadditive and $\forall t : z_r^{(1)}(t) = \sigma(k) = k$. Also, since $\sigma(\cdot)$ is 2-Lipschitz, the movement cost of $\mathbf{z}^{(1)}(\cdot)$ is at most twice the cost of $\boldsymbol{x}(\cdot)$.

Step 2. Next, we consider the strategy $\mathbf{z}^{(2)}(\cdot)$ defined by initializing $\mathbf{z}^{(2)}(0) = \mathbf{z}^{(1)}(0) = \boldsymbol{x}(0)$ and with the induction for $t \geq 1$,

$$\mathbf{z}^{(2)}(t) \in \operatorname*{arg\,min}_{\substack{\mathbf{z}\in\mathcal{Z} \\ \forall u\in V\colon z_u\in\frac{1}{m'}\mathbb{N}}} \mathrm{OT}(\mathbf{z}^{(2)}(t-1),\mathbf{z}) + \mathrm{OT}(\mathbf{z},\mathbf{z}^{(1)}(t)), \tag{5}$$

where we split ties in favor of a configuration maximizing $\mathrm{OT}(\mathbf{z}^{(2)}(t-1),\mathbf{z})$. Note that the old configuration $\mathbf{z} = \mathbf{z}^{(2)}(t-1)$ already minimizes (5), due to the triangle inequality. Starting from this $\mathbf{z}$, in a polynomial number of steps we can successively modify $\mathbf{z}$ until it satisfies the tie-breaking rule, as follows: While the tree contains an edge (u,v) such that $z_{[u]} \geq \frac{1}{m'}$ and the subtree V_u of nodes closer to u than v satisfies $\sum_{s\in V_u} z_{[s]} \geq \sum_{s\in V_u} z^{(1)}_{[s]} + \frac{1}{m'}$, we change $\mathbf{z}$ by removing mass $\frac{1}{m'}$ from u and adding it to v. This increases the first term in the optimization problem (5) by $\frac{1}{m'}$ times the length of the edge (u,v) and decreases the second term by the same amount. Once such an edge (u,v) no longer exists, we set $\mathbf{z}^{(2)}(t) := \mathbf{z}$. In other words, we gradually move $\mathbf{z}$ away from $\mathbf{z}^{(2)}(t-1)$ and towards $\mathbf{z}^{(1)}(t)$ as long as another m'-barely fractional configuration in that direction exists. Polynomial running time of the procedure follows from the fact that each edge (u,v) can be chosen at most $m'k$ times.

We first show that the movement cost of $\mathbf{z}^{(2)}(\cdot)$ is at most the movement cost of $\mathbf{z}^{(1)}(\cdot)$. We have by optimality of $\mathbf{z}^{(2)}(t)$ in (5) and triangle inequality that

$$\mathrm{OT}(\mathbf{z}^{(2)}(t-1),\mathbf{z}^{(2)}(t)) + \mathrm{OT}(\mathbf{z}^{(2)}(t),\mathbf{z}^{(1)}(t))) = \mathrm{OT}(\mathbf{z}^{(2)}(t-1),\mathbf{z}^{(1)}(t)).$$

By the triangle inequality $\mathrm{OT}(\mathbf{z}^{(2)}(t-1),\mathbf{z}^{(1)}(t)) \leq \mathrm{OT}(\mathbf{z}^{(2)}(t-1),\mathbf{z}^{(1)}(t-1)) + \mathrm{OT}(\mathbf{z}^{(1)}(t-1),\mathbf{z}^{(1)}(t))$, we then obtain the inequality

$$\mathrm{OT}(\mathbf{z}^{(2)}(t-1),\mathbf{z}^{(2)}(t))+\mathrm{OT}(\mathbf{z}^{(2)}(t),\mathbf{z}^{(1)}(t))-\mathrm{OT}(\mathbf{z}^{(2)}(t-1),\mathbf{z}^{(1)}(t-1)) \leq \mathrm{OT}(\mathbf{z}^{(1)}(t-1),\mathbf{z}^{(1)}(t)).$$

We therefore conclude that the movement cost of $\mathbf{z}^{(2)}(\cdot)$ is less than the movement cost of $\mathbf{z}^{(1)}(\cdot)$ by taking a telescopic sum in the above equation.

Second, we prove that for all nodes $u \in V\colon z^{(2)}_u(t) \geq z^{(1)}_u(t) - (2k+1)/m'$. For this, we denote by F_u the set of nodes $v \in V \setminus \{u\}$ satisfying $z^{(2)}_{[v]} > 0$ and such that $\mathbf{z}^{(2)}$ has no mass on the path from v to u (except at its endpoints). Denoting by $\pi(\cdot,\cdot)$ an optimal transport from $\mathbf{z}^{(1)}$ to $\mathbf{z}^{(2)}$, we have that

$$\begin{aligned}
z^{(2)}_u &= z^{(1)}_u + \sum_{v\in V}\pi(v,u) - \sum_{v\in V}\pi(u,v), \\
&\geq z^{(1)}_u - \sum_{v\in V\setminus\{u\}}\pi(u,v), \\
&= z^{(1)}_u - \sum_{v\in F_u}\left(\sum_{w\in V(v)}\pi(u,w)\right), &(6)\\
&\geq z^{(1)}_u - \frac{|F_u|}{m'}. &(7)
\end{aligned}$$

where in (6), we denote by $V(v)$ the tree composed of nodes that are separated from u by $v \in F_u$, and where for (7) we use that $\forall v \in F_u\colon \sum_{w\in V(v)}\pi(u,w) \leq 1/m'$, otherwise, the tie-breaking rule of (5) is contradicted by considering the configuration $\mathbf{z}$ where, starting from $\mathbf{z}^{(2)}$, we remove a mass of $1/m'$ from v and add it to u instead. Then, we observe that (a) at most one node of F_u is an ancestor of u, and we denote it by $\bar{u}$, and (b) the set $F'_u = F_u \setminus \{\bar{u}\}$ does not contain a pair of nodes where one is an ancestor of the other.

Since $\forall v \in F'_u \colon z_v^{(2)} > 0$ we have that $z_v^{(1)} > 0$ and thus that $x_v \geq 1/2$. Thus, all elements of F'_u correspond to a mass of at least $1/2$ for $\boldsymbol{x}$, which is of total mass k. This shows that $|F'_u| \leq 2k$, and thus that $|F_u| \leq 2k+1$.

Step 3. We then consider the sequence defined at all times by $\mathbf{z}^{(3)}(t) = \lambda \mathbf{z}^{(2)}(t)$, for $\lambda = \frac{1}{1-(2k+1)/m'}$, which defines $2m$-barely fractional measures of mass λk. We note that the constraint $4k^2 + 4k + 1 \leq m'$, i.e., $2k^2 + k \leq m$, effectively enforces that $\lambda k \in [k, k+1/2]$. Clearly, since $\lambda \leq 2$, the movement cost of $\mathbf{z}^{(3)}(\cdot)$ is at most twice the movement cost of $\mathbf{z}^{(2)}(\cdot)$. Also, for any node $u \in V$ for which $x_u \geq 1$, we have that $z_u^{(2)} \geq z_u^{(1)} - \frac{2k+1}{m'} \geq 1 - \frac{2k+1}{m'} = 1/\lambda$ and thus $z_u^{(3)} = \lambda z_u^{(2)} \geq 1$.

Step 4. We finally consider $\mathbf{z}^{(4)}(\cdot)$ defined by $\forall t \colon \mathbf{z}^{(4)}(t) = \sigma(\mathbf{z}^{(3)}(t))$. This defines a valid sequence of inner measures because $\sigma(\cdot)$ is super-additive, with mass k since we have that $z_r^{(4)} = \sigma(\lambda k) = k$. Also, we clearly have that whenever $x_u \geq 1$, $z_u^{(3)} \geq 1$ and thus, $z_u^{(4)} \geq 1$. Each measure $\mathbf{z}^{(4)}(t)$ is m-barely fractional because $\mathbf{z}^{(3)}(t)$ is $2m$-barely fractional and $\sigma\left(\frac{1}{2m}\mathbb{N}\right) \subset \frac{1}{m}\mathbb{N}$. By the same Lipschitzness argument as above, the movement cost of $\mathbf{z}^{(4)}(\cdot)$ is at most twice the movement cost of $\mathbf{z}^{(3)}(\cdot)$.

Step 5. We observe that all the definitions above can be implemented online and in polynomial time. The algorithm $\mathbf{z}^{(4)}(\cdot)$ satisfies almost all the desired properties: It is m-barely fractional, has a total mass k, serves the same requests as $\boldsymbol{x}(\cdot)$, and its movement cost is at most 8 times the movement cost of $\boldsymbol{x}(\cdot)$. However, it may have fractional mass at inner nodes of the tree. To prevent this, any movement of server mass can be deferred until it reaches a leaf, since placing server mass at internal nodes is never necessary for serving requests. ◀

We then observe that barely fractional algorithms can handle a small (compared to m) additive overhead, as exposed in the following proposition.

▶ **Proposition 7.** *Suppose that there is an m-barely fractional algorithm $\mathbf{z}(\cdot)$ satisfying for any request sequence $\rho(\cdot)$,*

$$\texttt{Cost}(\mathbf{z}(\cdot), \rho(\cdot)) \leq c\,\mathrm{OPT}(\rho(\cdot)) + \frac{1}{2m}T + c_0,$$

where T is the number of time-steps in the input sequence $\rho(\cdot)$, c_0 is a constant independent of $\rho(\cdot)$, and each non-zero distance is at least 1. Then the variant of $\mathbf{z}(\cdot)$ that ignores any request that it can serve without moving is $2c$-competitive.

Proof of Proposition 7. Let $\mathbf{z}'(\cdot)$ be the variant of $\mathbf{z}(\cdot)$ that ignores requests that are already satisfied. Concretely, we will have $\mathbf{z}'(t) = \mathbf{z}'(t-1)$ if $z'_{\rho(t)}(t-1) \geq 1$. We denote by $\rho'(\cdot)$ the subsequence of $\rho(\cdot)$ where we remove all such superfluous requests. The fractional algorithm $\mathbf{z}'(\cdot)$ is formally defined by the value of algorithm $\mathbf{z}(\cdot)$ on the request sequence $\rho'(\cdot)$. Denoting by T' the length of $\rho'(\cdot)$, i.e., the number of requests that are not superfluous, we have

$$\frac{1}{m}T' \leq \texttt{Cost}(\mathbf{z}'(\cdot), \rho(\cdot)) = \texttt{Cost}(\mathbf{z}(\cdot), \rho'(\cdot)) \leq c\mathrm{OPT}(\rho'(\cdot)) + \frac{1}{2m}T' + c_0, \tag{8}$$

where the first inequality comes from the fact that any movement entails a cost of at least $\frac{1}{m}$. Thus,

$$\frac{1}{2m}T' \leq c\mathrm{OPT}(\rho'(\cdot)) + c_0. \tag{9}$$

Therefore, observing that $\mathrm{OPT}(\rho'(\cdot)) \leq \mathrm{OPT}(\rho(\cdot))$ because $\rho'(\cdot)$ is a subsequence of $\rho(\cdot)$, we have

$$\texttt{Cost}(\mathbf{z}'(\cdot), \rho(\cdot)) \leq 2c\mathrm{OPT}(\rho(\cdot)) + 2c_0. \tag{10}$$

Since superfluous requests can be eliminated in constant time, the computational complexity of $\mathbf{z}'(\cdot)$ is no larger than the computational complexity of $\mathbf{z}(\cdot)$. ◀

3 Barely random k-server

▶ **Proposition 6** (Barely fractional to barely randomized)**.** *For k-server on a line metric or τ-HSTs with $\tau > 5$, any m-barely fractional algorithm can be converted online into an m-barely random algorithm with at most a polynomial additive overhead in running time and a constant factor multiplicative overhead in cost.*

Proof. We first prove the result for the line metric $\mathcal{M} = ([n], d(\cdot,\cdot))$, with $\forall u, v \in [n]^2 \colon d(u,v) = |u - v|$. We view this as a tree rooted at $1 \in [n]$, and for convenience, other than our usual convention, we consider internal nodes part of the metric space. (Recall that both models are equivalent by adding leaves at distance 0.) For a configuration $\mathbf{z}$, we have that z_u denotes the mass that is located "below" $u \in [n]$, whereas $z_{[u]} = z_u - z_{u+1}$ is the mass located at u (with the convention that $z_{n+1} = 0$). Then, given an m-barely fractional algorithm $\mathbf{z}(\cdot)$ for any $i \in \{1, \dots, m\}$, at any time t, we define the deterministic k-server configuration $\mathcal{R}_i(t)$ as

$$\mathcal{R}_i(t) = \left(\max\{u \in [n] \colon \left\lfloor z_u(t) + \frac{i-1}{m} \right\rfloor = h\} \right)_{h \in \{1,\dots,k\}}. \tag{11}$$

We observe that $\mathcal{R}_i(t)$ contains always exactly k points of $[n]$, because $z_1(t) = k$, and $z_{n+1}(t) = 0$. Also, we note that the randomized algorithm $\mathcal{R}(\cdot)$ effectively serves the incoming requests, i.e., $\forall i \in [m] \colon u \in \mathcal{R}_i(t)$. This is because, when the request is at $u = \rho(t)$, we must have $z_{[u]}(t) = 1$ and thus $z_u(t) = z_{u+1}(t) + 1$. Finally, we study the movement cost of $\mathcal{R}(\cdot)$. We note that when a unit mass of $\frac{1}{m}$ moves in $\mathbf{z}(t)$ along one edge, at most one configuration $\mathcal{R}_i(t)$ is affected, moving by a distance of at most one. This shows that the movement cost of $\mathcal{R}(\cdot)$ is less than the movement cost of $\mathbf{z}(\cdot)$, and it finishes the proof.

We then turn to the proof in the case where the tree is an HST. Adapting the proof of [2] is straightforward, and we only describe the main ideas. We will make sure that the m-barely random configuration $\mathcal{R}(\cdot) = (\mathcal{R}_1(\cdot), \dots, \mathcal{R}_m(\cdot))$ is *consistent* with $\mathbf{z}(\cdot)$ at all times, i.e., for all $t > 0$ and $u \in V$,

$$z_u(t) = \frac{1}{m} \sum_{i \in [m]} \sum_{v \in L_u} \mathbb{1}(v \in \mathcal{R}_i(t)).$$

We further say that a configuration $\mathcal{C}$ is *balanced* with respect to $\mathbf{z}$ if it satisfies $\forall u \in V \colon n_u(\mathcal{C}) = \sum_{v \in L_u} \mathbb{1}(v \in \mathcal{C}) \in \{\lfloor z_u \rfloor, \lceil z_u \rceil\}$, where $n_u(\mathcal{C})$ counts the number of servers below u in $\mathcal{C}$. The result is obtained by induction by applying the following lemma.

▶ **Lemma 9.** *Consider two m-barely fractional configurations $\mathbf{z}$ and $\mathbf{z}'$ and some m-barely random configuration $\mathcal{R} = (\mathcal{R}_1, \dots, \mathcal{R}_m)$ that is consistent and balanced with respect to $\mathbf{z}$. Then one can find in polynomial time an m-barely random configuration $\mathcal{R}' = (\mathcal{R}'_1, \dots, \mathcal{R}'_m)$ that is consistent and balanced with respect to $\mathbf{z}'$ and such that $\frac{1}{m} \sum_{i \in [m]} d(\mathcal{R}_i, \mathcal{R}'_i) = \mathcal{O}(\mathrm{OT}(\mathbf{z}, \mathbf{z}'))$.*

Proof sketch. Observe that it suffices to consider the elementary case where:

$$z'_{\ell'} \leftarrow z_{\ell'} + 1/m \quad \text{and} \quad z'_{\ell} \leftarrow z_{\ell} - 1/m,$$

for two leaves $\ell, \ell' \in L$. We first construct $\bar{\mathcal{R}}$ that is consistent with $\mathbf{z}'$, but imbalanced. If there exists $i \in [m]$ such that $\ell \in \mathcal{R}_i$ and $\ell' \notin \mathcal{R}_i$, we simply transfer the server at ℓ in $\mathcal{R}_i$ to ℓ'. Otherwise, since $z_\ell > 0$ and $z_{\ell'} < 1$, we can pick $i \neq j \in [m]$ such that $\ell, \ell' \in \mathcal{R}_i$ and $\ell, \ell' \notin \mathcal{R}_j$. We denote by u the lowest common ancestor of ℓ and ℓ'. Because both configurations are balanced with respect to $\mathbf{z}$, their numbers of servers below u differ by at most 1, and we can pick a leaf ℓ'' that is a descendant of u such that $\ell'' \in \mathcal{R}_j$ and $\ell'' \notin \mathcal{R}_i$. We then proceed to the following updates,

$$\bar{\mathcal{R}}_i \leftarrow \mathcal{R}_i \cup \{\ell''\} \setminus \{\ell\} \quad \text{and} \quad \bar{\mathcal{R}}_j \leftarrow \mathcal{R}_j \cup \{\ell'\} \setminus \{\ell''\}.$$

The total movement cost for this operation is at most $2\mathrm{OT}(\mathbf{z}, \mathbf{z}')$, and the configuration $\bar{\mathcal{R}}$ obtained is consistent with $\mathbf{z}'$. However, the configuration $\bar{\mathcal{R}}$ might not be balanced with respect to $\mathbf{z}'$, specifically, descendants of u might have become imbalanced after this transformation, where u is the lowest common ancestor of ℓ and ℓ'. We therefore need to transform $\bar{\mathcal{R}}$ into a configuration $\mathcal{R}'$ that is balanced, while incurring a limited movement cost. For this, the idea is to balance all descendants v of u, proceeding inductively from top to bottom. Concretely, this is achieved by exchanging points between pairs of configurations $(\mathcal{R}_i, \mathcal{R}_j)$ that satisfy $n_v(\mathcal{R}_i) > \lceil z_u \rceil$ and $n_v(\mathcal{R}_j) < \lfloor z_u \rfloor$, thus strictly reducing the balance gap

$$G(\mathcal{R}, \mathbf{z}) = \frac{1}{m} \sum_{v \in V} w_v \sum_{i \in [m]} \min\{|n_v(\mathcal{R}_i) - \lfloor z_v \rfloor|, |n_v(\mathcal{R}_i) - \lceil z_v \rceil|\},$$

while preserving the consistency property. We refer to [2] for complete details. This procedure runs in polynomial time because, when a node v is being balanced, the value of $\sum_{i \in [m]} \min\{|n_v(\mathcal{R}_i) - \lfloor z_v \rfloor|, |n_v(\mathcal{R}_i) - \lceil z_v \rceil|\}$ decreases by at least 2 at each balancing step. ◀

◀

4 Polynomial time algorithm with additive cost

In this section, we explain how we adapt the fractional k-server algorithm of [17] to run in polynomial time, albeit at the expense of an additive cost, as stated in Proposition 4.

▶ **Proposition 4.** *Let $\epsilon > 0$. There exists a fractional k-server algorithm $\mathbf{z}(\cdot)$ for τ-HST metrics with $\tau \geq 10$ that serves each request in polynomial time in $(n, \log 1/\epsilon, \log D)$, and satisfying for all requests sequences $\rho(\cdot)$*

$$\mathtt{Cost}(\mathbf{z}(\cdot), \rho(\cdot)) \leq \mathcal{O}(\log^2 k)\mathrm{OPT}(\rho(\cdot)) + \epsilon T + O(D \log^2 k),$$

where T is the length of the sequence $\rho(\cdot)$.

The algorithm in [17] operates by finding a solution to a constrained convex optimization problem, which is restated in Equation (12) below. As the solution can contain irrational numbers, we cannot solve it exactly in finite time or space, so we aim for an approximate solution. The constraint set, which is known in the literature as the assignment polytope [15]

or anti-server polytope [17], is defined by an exponential number of constraints, but admits a fast separation oracle (see Remark 16 in Appendix A). Thus, we can use the ellipsoid algorithm to find an approximate solution of this problem in polynomial time.

We first recall some definitions from [17] relevant to our discussion. Some of the notation in [17] is for the more general case where an online algorithm with k servers competes against an offline algorithm with h servers, for $h \leq k$. For our purposes, the special case $h = k$ suffices.

For a node $u \in V$, let $n_u = |L_u|$ be the number of leaves in the subtree rooted at u. Let $\chi := \{(u, j) \mid u \in V, j \in [n_u]\}$, and for $u \in V \setminus L$ let $\chi_u := \{(v, j) \in \chi \mid v \in C_u\} = \{(v, j) \mid v \in C_u, j \in [n_v]\}$. The anti-server polytope employed in [17], originally introduced in [15], is defined as

$$P := \left\{ \boldsymbol{x} \in [0,1]^{\chi} \;\middle|\; \begin{aligned} x_{rj} &\geq \mathbb{1}_{j>k} && \forall j \in [n_r], \\ \sum_{i \leq |S|} x_{ui} &\leq \sum_{(v,j) \in S} x_{vj} && \forall u \in V \setminus L,\, S \subseteq \chi_u \end{aligned} \right\}.$$

For intuition, an integral k-server configuration can be encoded by an "anti-server configuration" $\boldsymbol{x} \in P$ by setting $x_{uj} = 0$ if the subtree rooted at u contains at least j servers and $x_{uj} = 1$ otherwise. In particular, a leaf $\ell \in L$ contains a server if $x_{\ell 1} = 0$.

Let $\delta = \frac{1}{2k+1}$ and $P_\delta := \{\boldsymbol{x} \in P \mid x_{\ell 1} \geq \delta \text{ for all } \ell \in L\}$. The online algorithm in [17] maintains an anti-server configuration $\boldsymbol{x}(t)$ in P_δ that satisfies $\sum_{\ell \in L} x_{\ell 1}(t) = n - k$. This is then converted into a leaf measure $\mathbf{z}(t) \in \mathcal{Z}$ of mass $k + \frac{1}{2}$ by setting $z_\ell(t) := \frac{1 - x_{\ell 1}(t)}{1 - \delta}$ for $\ell \in L$ and $z_u = \sum_{\ell \in L_u} z_\ell$, thus yielding a fractional $\left(k + \frac{1}{2}\right)$-server algorithm. A simple transformation (first shown in [15], analogous to step 4 in our proof of Proposition 5) then shows that any fractional $\left(k + \frac{1}{2}\right)$-server algorithm can be converted into a fractional k-server algorithm while increasing the cost by only a constant factor.

When a request arrives at a leaf $\rho(t) \in L$ and the previous anti-server configuration is $\boldsymbol{x}(t-1) \in P$, the algorithm of [17] chooses the new anti-server configuration as

$$\boldsymbol{x}(t) := \operatorname*{arg\,min}_{\substack{\boldsymbol{x} \in P_\delta \\ x_{\rho(t)1} = \delta}} D(\boldsymbol{x} \parallel \boldsymbol{x}(t-1)), \tag{12}$$

where

$$D(\boldsymbol{x} \parallel \boldsymbol{x}') := \sum_{u \in V \setminus \{r\}} w_u \sum_j \left((x_{uj} + \delta) \log \frac{x_{uj} + \delta}{x'_{uj} + \delta} - x_{uj} + x'_{uj} \right). \tag{13}$$

Observe that the associated leaf measure $\mathbf{z}(t)$ serves the request, since $z_{\rho(t)} = \frac{1 - x_{\rho(t)1}}{1 - \delta} = 1$.

In the following Lemma,

$$\|\boldsymbol{x}(t) - \boldsymbol{x}(t-1)\|_w^+ := \sum_{u,i} w_u \left(x_{ui}(t) - x_{ui}(t-1)\right)^+$$

denotes the "positive movement cost" associated with the transition from $\boldsymbol{x}(t-1)$ to $\boldsymbol{x}(t)$. Over any sequence of steps, the total positive and negative movement are equal up to a bounded additive error, and hence the sum of these terms over all time steps upper bounds the movement cost of $\mathbf{z}(\cdot)$ up to a constant factor.

▶ **Lemma 10.** *Let the underlying tree be a τ-HST with $\tau \geq 10$. Let $\boldsymbol{y}(t) \in P$ be the anti-server configuration at time t corresponding to an optimal offline algorithm. For any configuration $\boldsymbol{x}(t-1) \in P_\delta$ at time $t-1$, the new configuration $\boldsymbol{x}(t)$ defined in (12) satisfies*

$$\|\boldsymbol{x}(t) - \boldsymbol{x}(t-1)\|_w^+ + Q(\boldsymbol{x}(t), \boldsymbol{y}(t)) - Q(\boldsymbol{x}(t-1), \boldsymbol{y}(t-1)) \leq \mathcal{O}\left(\log^2 k\right) \cdot \Delta_t \mathrm{OPT},$$

where $Q(\boldsymbol{x}, \boldsymbol{y})$ is a potential function that is $O(D \log^2 k)$-Lipschitz in any x_{ui}, and $\Delta_t \mathrm{OPT}$ is the cost of the offline algorithm at time t.

Proof. The function Q is a combination of several potential functions used in [17]. We prove our lemma by combining several inequalities that have been derived in [17]. Since the proof of some of these inequalities is rather involved, we do not repeat it here and instead refer to the original paper. When citing specific results from [17], we will refer to the numbering used in the corresponding full version of the paper [16].

First, define the potential function

$$\Phi(\boldsymbol{y} \parallel \boldsymbol{x}) := \sum_{u \in V \setminus \{r\}} w_u \sum_j (y_{uj} + \delta) \log \frac{y_{uj} + \delta}{x_{uj} + \delta}.$$

where $\boldsymbol{y} \in P$ and $\boldsymbol{x} \in P_\delta$ refer to the anti-server configurations of the offline and online algorithm, respectively. By [16, Lemma 3.6], when the offline algorithm moves from $\boldsymbol{y}(t-1)$ to $\boldsymbol{y}(t)$, the change in Φ can be bounded by

$$\Phi(\boldsymbol{y}(t) \parallel \boldsymbol{x}(t-1)) - \Phi(\boldsymbol{y}(t-1) \parallel \boldsymbol{x}(t-1)) \leq (1+\delta) \log\left(1 + \frac{1}{\delta}\right) \cdot \Delta_t \mathrm{OPT}. \tag{14}$$

Further, [16, Theorem 4.1] shows for some constants $c_1, c_2, c_3 > 0$ that

$$\|\boldsymbol{x}(t) - \boldsymbol{x}(t-1)\|_w^+ \leq c_1 \log \frac{k}{\delta} \cdot \sum_j A_{rj}^t + \Psi(\mathbf{z}(t)) - \Psi(\mathbf{z}(t-1)), \tag{15}$$

where $\mathbf{z}(t)$ is the leaf measure of mass $k + \frac{1}{2}$ corresponding to $\boldsymbol{x}(t)$ and

$$\Psi(\mathbf{z}) := c_2 \sum_u \frac{w_u}{1-\delta} \int_{z_u(0)}^{z_u} \ln\left(1 + \frac{z}{\delta}\right) dz - c_3 \sum_u w_u \left(z_u - \frac{2}{3} z_{p(u)}\right)^+$$

and A_{rj}^t are quantities that by [16, Inequality (3.13)] satisfy

$$\Phi(\boldsymbol{y}(t) \parallel \boldsymbol{x}(t-1)) - \Phi(\boldsymbol{y}(t) \parallel \boldsymbol{x}(t)) \geq \frac{1}{3} \sum_j A_{rj}^t + \frac{2}{3}(W(\boldsymbol{x}(t)) - W(\boldsymbol{x}(t-1))), \tag{16}$$

for

$$W(\boldsymbol{x}) := \sum_u \sum_j w_u x_{uj}.$$

Setting

$$Q(\boldsymbol{x}, \boldsymbol{y}) := 3c_1 \log \frac{k}{\delta} \cdot \Phi(\boldsymbol{y} \parallel \boldsymbol{x}) + 2c_1 \log \frac{k}{\delta} \cdot W(\boldsymbol{x}) - \Psi(\mathbf{z})$$

and combining inequalities (14), (15) and (16) yields

$$\begin{aligned}
&\|\boldsymbol{x}(t) - \boldsymbol{x}(t-1)\|_w^+ + Q(\boldsymbol{x}(t), \boldsymbol{y}(t)) - Q(\boldsymbol{x}(t-1), \boldsymbol{y}(t-1)) \\
&\qquad \leq 3c_1 \log \frac{k}{\delta} \cdot (1+\delta) \log\left(1 + \frac{1}{\delta}\right) \cdot \Delta_t \mathrm{OPT} \\
&\qquad = \mathcal{O}\left(\log^2 k\right) \cdot \Delta_t \mathrm{OPT}.
\end{aligned}$$

Using $w_u \leq D$ and $1/\delta = O(k)$, it is easy to check that Q is $O(D \log^2 k)$-Lipschitz in any x_{ui}. ◀

Equipped with Lemma 10, we can now conclude this section with the proof of Proposition 4.

Proof of Proposition 4. The algorithm is defined by induction by iteratively obtaining an approximate solution of Equation (12) with the Ellipsoid method (Corollary 15 in Appendix A) with a precision that we will specify later. This defines a sequence of points in the anti-server polytope $\boldsymbol{x}^\epsilon(t)$, which are then transformed into a fractional k-server configuration $\mathbf{z}(t)$ via the transformations explained above, with a loss of only a constant factor in the movement cost in the process.

By Corollary 15, the sequence $\boldsymbol{x}^\epsilon(t)$ satisfies the following approximate version of Lemma 10:

$$\|\boldsymbol{x}^\epsilon(t) - \boldsymbol{x}^\epsilon(t-1)\|_w^+ + Q(\boldsymbol{x}^\epsilon(t), \boldsymbol{y}(t)) - Q(\boldsymbol{x}^\epsilon(t-1), \boldsymbol{y}(t-1)) \\ \leq \mathcal{O}\left(\log^2 k\right) \cdot \Delta_t \mathrm{OPT} + \mathcal{O}(\epsilon' D \log^2 k),$$

where we used the fact that $Q(\boldsymbol{x}, \boldsymbol{y})$ is $O(D \log^2 k)$-Lipschitz in its first argument, and that $||\cdot||_w^+$ is $O(D)$-Lipschitz. By telescopic sum,

$$\sum_{t \leq T} \|\boldsymbol{x}^\epsilon(t) - \boldsymbol{x}^\epsilon(t-1)\|_w^+ \leq \mathcal{O}(\log^2 k)\mathrm{OPT}(\rho(\cdot)) + \mathcal{O}(\epsilon' D \log^2 kT) + \mathcal{O}(D \log^2 k),$$

where we use that $Q(\boldsymbol{x}, \boldsymbol{y})$ is bounded by $\mathcal{O}(D \log^2 k)$. For the right choice of precision in our application of Corollary 15 of order $\epsilon/(D \log^2 k)$, we thus have that

$$\mathtt{Cost}(\mathbf{z}(\cdot), \rho(\cdot)) \leq \mathcal{O}(\log^2 k)\mathrm{OPT}(\rho(\cdot)) + \epsilon T + \mathcal{O}(D \log^2 k). \qquad \blacktriangleleft$$

References

1. Anna Adamaszek, Artur Czumaj, Matthias Englert, and Harald Räcke. An $O(\log k)$-competitive algorithm for generalized caching. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1681–1689, 2012. doi:10.1137/1.9781611973099.133.
2. Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. *Journal of the ACM (JACM)*, 62(5):1–49, 2015. doi:10.1145/2783434.
3. Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. *Journal of the ACM (JACM)*, 59(4):1–24, 2012. doi:10.1145/2339123.2339126.
4. Nikhil Bansal and Christian Coester. Online metric allocation and time-varying regularization. In *30th Annual European Symposium on Algorithms, ESA*, 2022.
5. Nikhil Bansal, Joseph Naor, and Ohad Talmon. Efficient online weighted multi-level paging. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 94–104, 2021.
6. Yair Bartal and Eddie Grove. The harmonic k-server algorithm is competitive. *J. ACM*, 47(1):1–15, 2000. doi:10.1145/331605.331606.
7. Michael A Bender, Alex Conway, Martín Farach-Colton, William Kuszmaul, and Guido Tagliavini. Iceberg hashing: Optimizing many hash-table criteria at once. *Journal of the ACM*, 70(6):1–51, 2023. doi:10.1145/3625817.
8. Emery D Berger and Robert D Blumofe. Hoard: A fast, scalable, and memory-efficient allocator for shared-memory multiprocessors. Technical report, Technical Report UTCS-TR99-22, The University of Texas at Austin, 1999.
9. Avrim Blum, Carl Burch, and Adam Kalai. Finely-competitive paging. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 450–457. IEEE, 1999. doi:10.1109/SFFCS.1999.814617.

10 Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, and Richard Královič. On the advice complexity of the k-server problem. In *International Colloquium on Automata, Languages, and Programming*, pages 207–218. Springer, 2011.

11 Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. On the advice complexity of online problems. In *Algorithms and Computation: 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009. Proceedings 20*, pages 331–340. Springer, 2009. doi:10.1007/978-3-642-10631-6_35.

12 Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. cambridge university press, 2005.

13 Sébastien Bubeck. k-server via multiscale entropic regularization. TCS+ seminar. Video recording at https://www.youtube.com/watch?v=UAfo90a4Cg0&t=2452s, December 2017.

14 Sébastien Bubeck, Christian Coester, and Yuval Rabani. The randomized k-server conjecture is false! In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC*, 2023.

15 Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. k-server via multiscale entropic regularization. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC*, 2018.

16 Niv Buchbinder, Anupam Gupta, Marco Molinaro, and Joseph Naor. k-servers with a smile: Online algorithms via projections. Full version of [17] available at https://arxiv.org/pdf/1810.07508v3, 2018.

17 Niv Buchbinder, Anupam Gupta, Marco Molinaro, and Joseph (Seffi) Naor. k-servers with a smile: Online algorithms via projections. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, 2019.

18 Romain Cosson and Laurent Massoulié. Barely random algorithms and collective metrical task systems. *Advances in Neural Information Processing Systems*, 37:37337–37355, 2024.

19 Romain Cosson and Laurent Massoulié. Collective tree exploration via potential function method. In *15th Innovations in Theoretical Computer Science Conference (ITCS 2024)*, 2024.

20 Sina Dehghani, Soheil Ehsani, MohammadTaghi Hajiaghayi, Vahid Liaghat, and Saeed Seddighin. Stochastic k-Server: How Should Uber Work? In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 126:1–126:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.126.

21 Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. *Theoretical Computer Science*, 412(24):2642–2656, 2011. doi:10.1016/J.TCS.2010.08.007.

22 Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 448–455, 2003. doi:10.1145/780542.780608.

23 Amos Fiat, Richard M Karp, Michael Luby, Lyle A McGeoch, Daniel D Sleator, and Neal E Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991. doi:10.1016/0196-6774(91)90041-V.

24 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2. Springer Science & Business Media, 2012.

25 Edward F. Grove. The harmonic online k-server algorithm is competitive. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, STOC*, pages 260–266, 1991. doi:10.1145/103418.103448.

26 Dennis Komm. *Introduction to Online Computation*. Springer, 2016.

27 Dennis Komm and Richard Královič. Advice complexity and barely random algorithms. *RAIRO-Theoretical Informatics and Applications*, 45(2):249–267, 2011. doi:10.1051/ITA/2011105.

28 Elias Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009. doi:10.1016/J.COSREV.2009.04.002.

29 Elias Koutsoupias and Christos H Papadimitriou. On the k-server conjecture. *Journal of the ACM (JACM)*, 42(5):971–983, 1995. doi:10.1145/210118.210128.

30 Mark Manasse, Lyle McGeoch, and Daniel Sleator. Competitive algorithms for on-line problems. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 322–333, 1988.

31 Sharath Raghvendra and Rachita Sowle. A Scalable Work Function Algorithm for the k-Server Problem. In *18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022)*, volume 227 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SWAT.2022.30.

32 Nick Reingold, Jeffery Westbrook, and Daniel D Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11(1):15–32, 1994. doi:10.1007/BF01294261.

33 Marc P Renault and Adi Rosén. On online algorithms with advice for the k-server problem. *Theory of Computing Systems*, 56(1):3–21, 2015. doi:10.1007/S00224-012-9434-Z.

34 Stijn Schildermans, Kris Aerts, Jianchen Shan, and Xiaoning Ding. Ptlbmalloc2: Reducing tlb shootdowns with high memory efficiency. In *2020 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 76–83. IEEE Computer Society, 2020. doi:10.1109/ISPA-BDCLOUD-SOCIALCOM-SUSTAINCOM51426.2020.00036.

A Ellipsoid method

In this section, we briefly explain how the ellipsoid method allows us to provide approximate solutions for problems of the form of (12) in polynomial time. We start with some definitions from [24].

▶ **Definition 11** (Approximate oracles [24]). *An approximate separation oracle for a closed convex set $K \subset \mathbb{R}^d$ is an algorithm that takes as input a rational number $\gamma > 0$ and a rational point $\boldsymbol{y} \in \mathbb{Q}^d$, and that, in polynomial time in the encoding length of the inputs either reports that $\boldsymbol{y}$ is at distance at most γ from K, or returns a vector $\boldsymbol{c} \in \mathbb{Q}^d$ satisfying $||\boldsymbol{c}||_\infty = 1$ and $\forall \boldsymbol{x} \in K\colon \boldsymbol{c}^T\boldsymbol{x} \leq \boldsymbol{c}^T\boldsymbol{y} + \gamma$. An approximate first order oracle for a convex differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is an algorithm that takes as input a rational number $\gamma > 0$ and a rational point $\boldsymbol{y} \in \mathbb{Q}^d$, and that, in polynomial time in the encoding lengths of the inputs returns a vector $\boldsymbol{g} \in \mathbb{R}^d$ that satisfies $||\boldsymbol{g} - \nabla f(\boldsymbol{y})|| \leq \gamma$. An approximate zeroth-order oracle for $f(\cdot)$ is an algorithm that, for the same input, returns in polynomial time in the encoding lengths of the inputs a value v such that $|f(\boldsymbol{y}) - v| \leq \gamma$.*

▶ **Theorem 12** (Central cut ellipsoid, Theorem 3.2.1 of [24]). *There is an algorithm that takes as input a convex set $K \subset \mathbb{R}^d$ contained in a ball of radius R represented by an approximate separation oracle, and a precision $\epsilon > 0$, and in polynomial time in $(d,\ \log R,\ \log 1/\epsilon)$ either outputs a point that is at distance at most ϵ from K or an ellipsoid of volume at most ϵ^{2d} that contains K.*

▶ Remark 13. In the original statement of Theorem 12, the volume of the ellipsoid for the failure case is ϵ, but it can be strengthened to ϵ^{2d} for a computational cost of same order by running the algorithm with $\epsilon' = \epsilon^{2d}$. In practice, if the polytope is full-dimensional, its volume is lower-bounded by an exponential function of its encoding length (see [24, Lemma 3.1.35]), and the failure case is not raised.

We will utilize the following corollary, which concerns the approximate optimization of convex functions.

▶ **Corollary 14.** *There is an algorithm that takes as input a precision $\epsilon > 0$, a closed convex set $K \subset \mathbb{R}^d$ contained in a ball of radius R represented by an approximate separation oracle and satisfying that the volume of the intersection of K with any ball centered on a point of K of radius r is at least $\Omega(r^d)$, and a convex function $f(\cdot)$ of $\mathbb{R}^d$ represented by its approximate zeroth-order and first-order oracles, that is L-Lipschitz, α-strongly convex, and a closed interval of width w to which the minimum value of $f(\cdot)$ belongs, and that, in time polynomial in $(d, \log R, \log 1/\epsilon, \log 1/\alpha, \log L)$ outputs a point $\boldsymbol{x}^{*,\epsilon}$ satisfying $||\boldsymbol{x}^{*,\epsilon} - \boldsymbol{x}^*|| \leq \epsilon$ where $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x} \in K} f(\boldsymbol{x})$.*

Proof. Define $f^* = \min_{\boldsymbol{x} \in K} f(\boldsymbol{x})$. for a fixed threshold $A \in \mathbb{R}$, consider the closed convex set defined by

$$K_A = \{\boldsymbol{x} \colon f(\boldsymbol{x}) \leq A\} \cap K.$$

We start by assuming that we have $A \in [f^* + \epsilon', f^* + 2\epsilon']$, where $\epsilon' = \alpha\epsilon^2/8$, and that we are given $\boldsymbol{x}^{*,\epsilon}$ that is at a distance at most $\epsilon/2$ from a point $\boldsymbol{x}_A$ in K_A. Then, by strong convexity of $f(\cdot)$, we have $\alpha||\boldsymbol{x}^* - \boldsymbol{x}_A||^2 \leq f(\boldsymbol{x}_A) - f(\boldsymbol{x}^*) \leq \alpha\epsilon^2/4$ so that $||\boldsymbol{x}_A - \boldsymbol{x}^*|| \leq \epsilon/2$. And by triangle inequality, $||\boldsymbol{x}^{*,\epsilon} - \boldsymbol{x}^*|| \leq \epsilon/2 + \epsilon/2 = \epsilon$.

We now explain how to find a value of $A \in [f^* + \epsilon', f^* + 2\epsilon']$, and to obtain the corresponding point $\boldsymbol{x}^{*,\epsilon}$ in polynomial time. We will proceed by dichotomy. We know that the minimum value of the function is within an interval of width w. We maintain the interval $[A_i^-, A_i^+]$ of candidates for A, starting with $[A_0^-, A_0^+]$ being that interval of width w. We run the central cut ellipsoid algorithm for K_{A_i} where A_i is the midpoint of the interval, $A_i = (A_i^+ + A_i^-)/2$, and, if it returns a solution, we update the interval to $[A_i^-, A_i]$ and otherwise to $[A_i, A_i^+]$. We note that in only $\mathcal{O}(\log w + \log 1/\epsilon')$ steps, the width of the interval is smaller than $\epsilon'/2$. We also note that A_i^- cannot be greater than $f^* + \epsilon'$, because, for any $A \geq f^* + \epsilon'$, we have that the volume of K_A is in $\Omega((\epsilon'/L)^d)$ because it contains the intersection of K with a ball of radius ϵ'/L. Finally, running the central cut ellipsoid algorithm on $A_i^+ \in [f^* + \epsilon', f^* + 2\epsilon']$ returns the desired output $\boldsymbol{x}^{*,\epsilon}$.

To conclude, we need to argue that for any value of A, we can run the central cut ellipsoid algorithm for K_A. Note that K_A is contained in a closed convex set included in a ball of radius R and that an approximate separation oracle for K_A is given at $\boldsymbol{y} \in \mathbb{Q}^d$ by (1) an approximate separation oracle for K at $\boldsymbol{y}$ if $\boldsymbol{y} \notin K$ (2) an approximate gradient of $f(\cdot)$ at $\boldsymbol{y}$ if $f(\boldsymbol{y}) \geq A$. Thus, applying Theorem 12, the central cut ellipsoid method applied to K_A and $\epsilon/2$ outputs either a point that is at distance at most $\epsilon/2$ from K_A or a certificate that the volume of K_A is less than $(\epsilon/2)^{2d}$. ◀

We now apply this theorem to real functions of the form $f(\boldsymbol{x}) = D(\boldsymbol{x} \parallel \boldsymbol{x}(t-1))$ for the divergence defined in (13) and some fixed vector $\boldsymbol{x}(t-1)$ and to the anti-server polytope P_δ (where we drop the variable $x_{\rho(t)1}$ which is set equal to δ). The dimension d of the polytope is bounded by n^2, the function $f(\cdot)$ is L-Lipschitz with $L = O(D \log k)$ and α-strongly convex with $\alpha = \Omega(1)$. There exists a separation oracle for the anti-server polytope (Remark 16), and, furthermore, by Taylor expansions of the $\log(\cdot)$ on the interval $[\delta, 1+\delta]$, approximate zeroth and first-order oracle for $f(\cdot)$ are known. Finally, we also note that for any point $\boldsymbol{x}$ at a distance η from the antiserver polytope, there is a polynomial time procedure (specifically, enforcing the constraints from top to bottom in the tree) that returns a point $\boldsymbol{x}'$ in the antiserver polytope at distance at most $O(\eta)$ from $\boldsymbol{x}$. Therefore, applying Corollary 14, we have the following result.

▶ **Corollary 15.** *For any $\epsilon > 0$, for any HST with n nodes and diameter D, and for any element $\boldsymbol{x}(t-1)$ of the antiserver polytope, there is a polynomial time algorithm that provides an approximate solution $\boldsymbol{x}^\epsilon$ of (12) satisfying $||\boldsymbol{x}(t) - \boldsymbol{x}^\epsilon|| \leq \epsilon$ in time polynomial in $(\log 1/\epsilon, n, \log D)$.*

▶ Remark 16 (Separation oracle for the antiserver polytope). Let n_u be the number of leaves below u. Recall that $\chi(u)$ is the set of pairs (v, i) where v is a child of u and $i \leq n_v$. Let $(v_1, i_1), (v_2, i_2), \ldots$ be the list of elements of $\chi(u)$ in increasing order of their x-values. Note that a constraint between u and its children is violated if and only if the constraint of a set S that is a prefix of this list is violated. Since there are only n_u prefixes, for each u, we only need to check polynomially many constraints to find a violated constraint, if one exists.