Slaying Hydrae: Improved Bounds for Generalized k-Server in Uniform Metrics

Marcin Bienkowski 🆔
Institute of Computer Science, University of Wrocław, Poland
marcin.bienkowski@cs.uni.wroc.pl

Łukasz Jeż 🆔
Institute of Computer Science, University of Wrocław, Poland
lukasz.jez@cs.uni.wroc.pl

Paweł Schmidt
Institute of Computer Science, University of Wrocław, Poland
pawel.schmidt@cs.uni.wroc.pl

Abstract

The generalized k-server problem is an extension of the weighted k-server problem, which in turn extends the classic k-server problem. In the generalized k-server problem, each of k servers $s_1, \ldots, s_k$ remains in its own metric space M_i. A request is a tuple $(r_1, \ldots, r_k)$, where $r_i \in M_i$, and to service it, an algorithm needs to move at least one server s_i to the point r_i. The objective is to minimize the total distance traveled by all servers.

In this paper, we focus on the generalized k-server problem for the case where all M_i are uniform metrics. We show an $O(k^2 \cdot \log k)$-competitive randomized algorithm improving over a recent result by Bansal et al. [SODA 2018], who gave an $O(k^3 \cdot \log k)$-competitive algorithm. To this end, we define an abstract online problem, called Hydra game, and we show that a randomized solution of low cost to this game implies a randomized algorithm to the generalized k-server problem with low competitive ratio.

We also show that no randomized algorithm can achieve competitive ratio lower than $\Omega(k)$, thus improving the lower bound of $\Omega(k/\log^2 k)$ by Bansal et al.

1 Introduction

The k-server problem, introduced by Manasse et al. [18], is one of the most well-studied and influential cornerstones of online analysis. The problem definition is deceivingly simple: There are k servers, starting at a fixed set of k points of a metric space M. An input is a sequence of requests (points of M) and to service a request, an algorithm needs to move servers, so that at least one server ends at the request position. As typical for online problems, the k-server problem is sequential in nature: an online algorithm ALG learns a new request only after it services the current one. The cost of ALG, defined as the total distance traveled by all its servers, is then compared to the cost of an *offline* solution OPT; the ratio between them, called *competitive ratio*, is subject to minimization.

In a natural extension of the k-server problem, called the *generalized k-server problem* [16, 20], each server s_i remains in its own metric space M_i. The request is a k-tuple $(r_1, \ldots, r_k)$, where $r_i \in M_i$, and to service it, an algorithm needs to move servers, so that *at least one* server s_i ends at the request position r_i. The original k-server problem corresponds to the

case where all metric spaces M_i are identical and each request is of the form $(r, \ldots, r)$. The generalized k-server problem contains many known online problems, such as the weighted k-server problem [1, 7, 11, 12] or the CNN problem [8, 16, 19, 20] as special cases.

So far, the existence of an $f(k)$-competitive algorithm for the generalized k-server problem in arbitrary metric spaces remains open. Furthermore, even for specific spaces, such as the line [16] or uniform metrics [1, 2, 16], the generalized k-server problem requires techniques substantially different from those used to tackle the classic k-server problems. For these reasons, studying this problem could lead to new techniques for designing online algorithms.

1.1 Previous Work

After almost three decades of extensive research counted in dozens of publications (see, e.g., a slightly dated survey by Koutsoupias [14]), we are closer to understanding the nature of the classic k-server problem. The competitive ratio achievable by deterministic algorithms is between k [18] and $2k - 1$ [15] with k-competitive algorithms known for special cases, such as uniform metrics [21], lines and trees [9, 10], or metrics of $k + 1$ points [18]. Less is known about competitive ratios for randomized algorithms: the best known lower bound holding for an arbitrary metric space is $\Omega(\log k/\log\log k)$ [4] and the currently best upper bound of $O(\log^6 k)$ has been recently obtained in a breakthrough result [6, 17].

In comparison, little is known about the generalized k-server problem. In particular, algorithms attaining competitive ratios that are functions of k exist only in a few special cases. The case of $k = 2$ has been solved by Sitters and Stougie [20, 19], who gave constant competitive algorithms for this setting. Results for $k \geq 3$ are known only for simpler metric spaces, as described below.

A *uniform metric* case describes a scenario where all metrics M_i are uniform with pairwise distances between different points equal to 1. For this case, Bansal et al. [2] recently presented an $O(k \cdot 2^k)$-competitive deterministic algorithm and an $O(k^3 \cdot \log k)$-competitive randomized one. The deterministic competitive ratio is at least $2^k - 1$ already when metrics M_i have two points [16]. Furthermore, using a straightforward reduction to the metrical task system (MTS) problem [5], they show that the randomized competitive ratio is at least $\Omega(k/\log k)$ [2].[1]

A *weighted uniform metric* case describes a scenario where each metric M_i is uniform, but they have different scales, i.e., the pairwise distances between points of M_i are equal to some values $w_i > 0$. For this setting, Bansal et al. [2] gave an $2^{2^{O(k)}}$-competitive deterministic algorithm extending an $2^{2^{O(k)}}$-competitive algorithm for the weighted k-server problem in uniform metrics [12]. (The latter problem corresponds to the case where all requests are of the form $(r, \ldots, r)$.) This matches a lower bound of $2^{2^{\Omega(k)}}$ [1] (which also holds already for the weighted k-server problem).

1.2 Our Results and Paper Organization

In this paper, we study the uniform metric case of the generalized k-server problem. We give a randomized $O(k^2 \cdot \log k)$-competitive algorithm improving over the $O(k^3 \cdot \log k)$ bound by Bansal et al. [2].

---

[1] In fact, for the generalized k-server problem in uniform metrics, the paper by Bansal et al. [2] claims only the randomized lower bound of $\Omega(k/\log^2 k)$. To obtain it, they reduce the problem to the n-state metrical task system (MTS) problem and apply a lower bound of $\Omega(\log n/(\log\log n)^2)$ for MTS [3]. By using their reduction and a stronger lower bound of $\Omega(\log n/\log\log n)$ for n-state MTS [4], one could immediately obtain a lower bound of $\Omega(k/\log k)$ for the generalized k-server problem.

To this end, we first define an elegant abstract online problem: a *Hydra game* played by an online algorithm against an adversary on an unweighted tree. We present the problem along with a randomized, low-cost online algorithm HERC in Section 2. We defer a formal definition of the generalized k-server problem to Section 3.1. Later, in Section 3.2 and Section 3.3, we briefly sketch the structural claims concerning the generalized k-server problem given by Bansal et al. [2]. Using this structural information, in Section 3.4, we link the generalized k-server problem to the Hydra game: we show that a (randomized) algorithm of total cost R for the Hydra game on a specific tree (called *factorial tree*) implies a (randomized) $(R + 1)$-competitive solution for the generalized k-server problem. This, along with the performance guarantees of HERC given in Section 2, yields the desired competitiveness bound. We remark that while the explicit definition of the Hydra game is new, the algorithm of Bansal et al. [2] easily extends to its framework.

Finally, in Section 4, we give an explicit lower bound construction for the generalized k-server problem, which does not use a reduction to the metrical task system problem, hereby improving the bound from $\Omega(k/\log k)$ to $\Omega(k)$.

2 Hydra Game

The Hydra game[2] is played between an online algorithm and an adversary on a fixed unweighted tree T, known to the algorithm in advance. The nodes of T have states which change throughout the game: Each node can be either *asleep*, *alive* or *dead*. Initially, the root r_T is alive and all other nodes are asleep. At all times, the following invariant is preserved: all ancestors of alive nodes are dead and all their descendants are asleep. In a single step, the adversary picks a single alive node w, kills it (changes its state to dead) and makes all its (asleep) children alive. Note that such adversarial move preserves the invariant above.

An algorithm must remain at some alive node (initially, it is at the root r_T). If an algorithm is at a node w that has just been killed, it has to move to any still alive node w' of its choice. For such movement it pays $\mathsf{dist}(w, w')$, the length of the shortest path between w and w' in the tree T. The game ends when all nodes except one (due to the invariant, it has to be an alive leaf) are dead. Unlike many online problems, here our sole goal is to minimize the total (movement) cost of an online algorithm (i.e., without comparing it to the cost of the offline optimum).

This game is not particularly interesting in the deterministic setting: As an adversary can always kill the node where a deterministic algorithm resides, the algorithm has to visit all but one nodes of tree T, thus paying $\Omega(|T|)$. On the other hand, a trivial DFS traversal of tree T has the cost of $O(|T|)$. Therefore, we focus on randomized algorithms and assume that the adversary is oblivious: it knows an online algorithm, but not the random choices made by it thus far.

2.1 Randomized Algorithm Definition

It is convenient to describe our randomized algorithm HERC as maintaining a probability distribution η over set of nodes, where for any node u, $\eta(u)$ denotes the probability that HERC is at u. We require that $\eta(u) = 0$ for any non-alive node u. Whenever HERC decreases

---

[2] This is a work of science. Any resemblance of the process to the decapitation of a mythical many-headed serpent-shaped monster is purely coincidental.

the probability at a given node u by p and increases it at another node w by the same amount, we charge cost $p \cdot \mathsf{dist}(u, w)$ to HERC. By a straightforward argument, one can convert such description into a more standard, "behavioral" one, which describes randomized actions conditioned on the current state of an algorithm, and show that the costs of both descriptions coincide. We present the argument in Appendix A for completeness.

At any time during the game, for any node u from tree T, $\mathsf{rank}(u)$ denotes the number of non-dead (i.e., alive or asleep) leaves in the subtree rooted at u. As HERC knows tree T in advance, it knows node ranks as well. Algorithm HERC maintains η that is distributed over all alive nodes proportionally to their ranks. As all ancestors of an alive node are dead and all its descendants are asleep, we have $\eta(u) = \mathsf{rank}(u)/\mathsf{rank}(r_T)$ if u is alive and $\eta(u) = 0$ otherwise. In particular, at the beginning η is 1 at the root and 0 everywhere else.

While this already defines the algorithm, we still discuss its behavior when an alive node u is killed by the adversary. By HERC definition, we can think of the new probability distribution η' as obtained from η in the following way. First, HERC sets $\eta'(u) = 0$. Next, the probability distribution at other nodes is modified as follows.

Case 1. Node u is not a leaf. HERC distributes the probability of u among all (now alive) children of u proportionally to their ranks, i.e., sets $\eta'(w) = (\mathsf{rank}(w)/\mathsf{rank}(u)) \cdot \eta(u)$ for each child w of u.

Case 2. Node u is a leaf. Note that there were some other non-dead leaves, as otherwise the game would have ended before this step, and therefore $\eta(u) \leq 1/2$. HERC distributes $\eta(u)$ among all other nodes, scaling the probabilities of the remaining nodes up by a factor of $1/(1 - \eta(u))$. That is, it sets $\eta'(w) = \eta(w)/(1 - \eta(u))$ for any node w.

Note that in either case, η' is a valid probability distribution, i.e., all probabilities are non-negative and sum to 1. Moreover, η' is distributed over alive nodes proportionally to their new ranks, and is equal to zero at non-alive nodes.

▶ **Observation 1.** *At any time, the probability of an alive leaf u is exactly $\eta(u) = 1/\mathsf{rank}(r_T)$.*

2.2 Analysis

For the analysis, we need a few more definitions. We denote the height and the number of the leaves of tree T by h_T and L_T, respectively. Let $\mathsf{level}(u)$ denote the height of the subtree rooted at u, where leaves are at level 0. Note that $h_T = \mathsf{level}(r_T)$.

To bound the cost of HERC, we define a potential Φ, which is a function of the current state of all nodes of T and the current probability distribution η of HERC. We show that Φ is initially $O(h_T \cdot (1 + \log L_T))$, is always non-negative, and the cost of each HERC's action can be covered by the decrease of Φ. This will show that the total cost of HERC is at most the initial value of Φ, i.e., $O(h_T \cdot (1 + \log L_T))$.

Recall that $\eta(w) = 0$ for any non-alive node w and that $\mathsf{rank}(u)$ is the number of non-dead leaves in the subtree rooted at u. Specifically, $\mathsf{rank}(r_T)$ is the total number of non-dead leaves in T. The potential is defined as

$$\Phi = 4 \cdot h_T \cdot H(\mathsf{rank}(r_T)) + \sum_{w \in T} \eta(w) \cdot \mathsf{level}(w) \,, \tag{1}$$

where $H(n) = \sum_{i=1}^{n} 1/i$ is the n-th harmonic number.

▶ **Lemma 2.** *At any time, $\Phi = O(h_T \cdot (1 + \log L_T))$.*

Proof. Since $\mathsf{rank}(r_T) \leq L_T$ at all times, the first summand of Φ is $O(h_T \cdot \log L_T)$. The second summand of Φ is a convex combination of node levels, which range from 0 to h_T, and is thus bounded by h_T. ◀

▶ **Lemma 3.** *Fix any step in which an adversary kills a node u and in result* HERC *changes the probability distribution from η to η'. Let ΔHERC be the cost incurred in this step by* HERC *and let $\Delta\Phi$ be the resulting change in the potential Φ. Then, $\Delta\Phi \leq -\Delta$HERC.*

Proof. We denote the ranks before and after the adversarial event by rank and rank', respectively. We consider two cases depending on the type of u.

Case 1. The killed node u is an internal node. In this case, ΔHERC $= \eta(u)$ as HERC simply moves the total probability of $\eta(u)$ along a distance of one (from u to its children). As $\mathsf{rank}'(r_T) = \mathsf{rank}(r_T)$, the first summand of Φ remains unchanged. Let $C(u)$ be the set of children of u. Then,

$$\Delta\Phi = \sum_{w \in T} (\eta'(w) - \eta(w)) \cdot \mathsf{level}(w) = -\eta(u) \cdot \mathsf{level}(u) + \sum_{w \in C(u)} \eta'(w) \cdot \mathsf{level}(w)$$

$$\leq -\eta(u) \cdot \mathsf{level}(u) + \sum_{w \in C(u)} \eta'(w) \cdot (\mathsf{level}(u) - 1)$$

$$= -\eta(u) \cdot \mathsf{level}(u) + \eta(u) \cdot (\mathsf{level}(u) - 1) = -\Delta\text{HERC},$$

where the inequality holds as level of a node is smaller than the level of its parent and the penultimate equality follows as the whole probability mass at u is distributed to its children.

Case 2. The killed node u is a leaf. It is not the last alive node, as in such case the game would have ended before, i.e., it holds that $\mathsf{rank}(r_T) \geq 2$. HERC moves the probability of $\eta(u) = 1/\mathsf{rank}(r_T)$ (cf. Observation 1) along a distance of at most $2 \cdot h_T$, and thus ΔHERC $\leq 2 \cdot h_T/\mathsf{rank}(r_T)$.

Furthermore, for any $w \neq u$, $\eta'(w) = \eta(w)/(1 - \eta(u))$. Using $\eta(u) = 1/\mathsf{rank}(r_T)$, we infer that the probability at a node $w \neq u$ increases by

$$\eta'(w) - \eta(w) = \left(\frac{1}{1 - \eta(u)} - 1 \right) \cdot \eta(w) = \frac{\eta(u)}{1 - \eta(u)} \cdot \eta(w)$$

$$= \frac{1}{\mathsf{rank}(r_T) - 1} \cdot \eta(w) \leq \frac{2}{\mathsf{rank}(r_T)} \cdot \eta(w), \tag{2}$$

where the last inequality follows as $\mathsf{rank}(r_T) \geq 2$.

Using (2) and the relation $\mathsf{rank}'(r_T) = \mathsf{rank}(r_T) - 1$ (the number of non-dead leaves decreases by 1), we compute the change of the potential:

$$\Delta\Phi = 4 \cdot h_T \cdot \left(H(\mathsf{rank}'(r_T)) - H(\mathsf{rank}(r_T)) \right) + \sum_{w \in T} (\eta'(w) - \eta(w)) \cdot \mathsf{level}(w)$$

$$= -\frac{4 \cdot h_T}{\mathsf{rank}(r_T)} + (\eta'(u) - \eta(u)) \cdot \mathsf{level}(u) + \sum_{w \neq u} (\eta'(w) - \eta(w)) \cdot \mathsf{level}(w)$$

$$\leq -\frac{4 \cdot h_T}{\mathsf{rank}(r_T)} + \sum_{w \neq u} \frac{2}{\mathsf{rank}(r_T)} \cdot \eta(w) \cdot h_T \leq -\frac{2 \cdot h_T}{\mathsf{rank}(r_T)} \leq -\Delta\text{HERC}.$$

In the first inequality, we used that $\mathsf{level}(u) = 0$ and $\mathsf{level}(w) \leq h_T$ for any w.
Summing up, we showed that $\Delta\Phi \leq -\Delta$HERC in both cases.   ◀

▶ **Theorem 4.** *For the Hydra game played on any tree T of height h_T and L_T leaves, the total cost of* HERC *is at most $O(h_T \cdot (1 + \log L_T))$.*

Proof. Let Φ_B denote the initial value of Φ. By non-negativity of Φ and Lemma 3, it holds that the total cost of HERC is at most Φ_B. The latter amount is at most $O(h_T \cdot (1 + \log L_T))$ by Lemma 2. ◀

Although HERC and Theorem 4 may seem simple, when applied to appropriate trees, they yield improved bounds for the generalized k-server problem in uniform metrics, as shown in the next section.

3   Improved Algorithm for Generalized k-Server Problem

In this part, we show how any solution for the Hydra game on a specific tree (defined later) implies a solution to the generalized k-server problem in uniform metrics. This will yield an $O(k^2 \log k)$-competitive randomized algorithm for the generalized k-server problem, improving the previous bound of $O(k^3 \cdot \log k)$ [2]. We note that this reduction is implicit in the paper of Bansal et al. [2], so our contribution is in formalizing the Hydra game and solving it more efficiently.

3.1   Preliminaries

The generalized k-server problem in uniform metrics is formally defined as follows. The offline part of the input comprises k uniform metric spaces $M_1, \ldots, M_k$. The metric M_i has $n_i \geq 2$ points, the distance between each pair of its points is 1. There are k servers denoted $s_1, \ldots, s_k$, the server s_i starts at some fixed point in M_i and always remains at some point of M_i.

The online part of the input is a sequence of requests, each request being a k-tuple $(r_1, \ldots, r_k) \in \prod_{i=1}^{k} M_i$. To service a request, an algorithm needs to move its servers, so that at least one server s_i ends at the request position r_i. Only after the current request is serviced, an online algorithm is given the next one.

The cost of an algorithm ALG on input I, denoted ALG(I), is the total distance traveled by all its k servers. We say that a randomized online algorithm ALG is β-competitive if there exists a constant γ, such that for any input I, it holds that $\mathbf{E}[\mathrm{ALG}(I)] \leq \beta \cdot \mathrm{OPT}(I) + \gamma$, where the expected value is taken over all random choices of ALG, and where OPT(I) denotes the cost of an optimal *offline* solution for input I. The constant γ may be a function of k, but it cannot depend on an online part of the input.

3.2   Phase-Based Approach

We start by showing how to split the sequence of requests into phases. To this end, we need a few more definitions. A (server) *configuration* is a k-tuple $c = (c_1, \ldots, c_k) \in \prod_{i=1}^{k} M_i$, denoting positions of respective servers. For a request $r = (r_1, \ldots, r_k) \in \prod_{i=1}^{k} M_i$, we define the set of *compatible* configurations $\mathsf{comp}(r) = \{(c_1, \ldots, c_k) : \exists_i \, c_i = r_i\}$, i.e., the set of all configurations that can service the request r without moving a server. Other configurations we call *incompatible* with r.

An input is split into phases, with the first phase starting with the beginning of an input. The phase division process described below is constructed to ensure that OPT pays at least 1 in any phase, perhaps except the last one. At the beginning of a phase, all configurations are *phase-feasible*. Within a phase, upon a request r, all configurations incompatible with r become *phase-infeasible*. The phase ends once all configurations are phase-infeasible; if this is not the end of the input, the next phase starts immediately, i.e., all configurations are

restored to the phase-feasible state before the next request. Note that the description above is merely a way of splitting an input into phases and marking configurations as phase-feasible and phase-infeasible. The actual description of an online algorithm will be given later.

Fix any finished phase and any configuration c and consider an algorithm that starts the phase with its servers at configuration c. When configuration c becomes phase-infeasible, such algorithm is forced to move and pay at least 1. As each configuration eventually becomes phase-infeasible in a finished phase, any algorithm (even OPT) must pay at least 1 in any finished phase. Hence, if the cost of a phase-based algorithm for servicing requests of a single phase can be bounded by $f(k)$, the competitive ratio of this algorithm is then at most $f(k)$.

3.3 Configuration Spaces

Phase-based algorithms that we construct will not only track the set of phase-feasible configurations, but they will also group these configurations in certain sets, called *configuration spaces*.

To this end, we introduce a special *wildcard* character $\star$. Following [2], for any k-tuple $q = (q_1, \ldots, q_k) \in \prod_{i=1}^{k}(M_i \cup \{\star\})$, we define a *(configuration) space* $S[q] = \{(c_1, \ldots, c_k) \in \prod_{i=1}^{k} M_i : \forall_i \, c_i = q_i \vee q_i = \star\}$. A coordinate with $q_i = \star$ is called *free* for the configuration space $S[q]$. That is, $S[q]$ contains all configurations that agree with q on all non-free coordinates.

The number of free coordinates in q defines the *dimension* of $S[q]$ denoted $\dim(S[q])$. Observe that the k-dimensional space $S[(\star, \ldots, \star)]$ contains all configurations. If tuple q has no $\star$ at any position, then $S[q]$ is 0-dimensional and contains only (configuration) q. The following lemma, proven by Bansal et al. [2], follows immediately from the definition of configuration spaces.

▶ **Lemma 5** (Lemma 3.1 of [2]). *Let $S[q]$ be a d-dimensional configuration space (for some $d \geq 0$) whose all configurations are phase-feasible. Fix a request r. If there exists a configuration in $S[q]$ that is not compatible with r, then there exist d (not necessarily disjoint) subspaces $S[q_1], \ldots, S[q_d]$, each of dimension $d - 1$, such that $\bigcup_i S[q_i] = S[q] \cap \mathsf{comp}(r)$. Furthermore, for all i, the k-tuples q_i and q differ exactly at one position.*

Using the lemma above, we may describe a way for an online algorithm to keep track of all phase-feasible configurations. To this end, it maintains a set $\mathcal{A}$ of (not necessarily disjoint) configuration spaces, such that their union is exactly the set of all phase-feasible configurations. We call spaces from $\mathcal{A}$ *alive*.

At the beginning, $\mathcal{A} = \{S[(\star, \ldots, \star)]\}$. Assume now that a request r makes some configurations from a d-dimensional space $S[q] \in \mathcal{A}$ phase-infeasible. (A request may affect many spaces from $\mathcal{A}$; we apply the described operations to each of them sequentially in an arbitrary order.) In such case, $S[q]$ stops to be alive, it is removed from $\mathcal{A}$ and till the end of the phase it will be called *dead*. Next, we apply Lemma 5 to $S[q]$, obtaining d configuration spaces $S[q_1], \ldots, S[q_d]$, such that their union is $S[q] \cap \mathsf{comp}(r)$, i.e., contains all those configurations from $S[q]$ that remain phase-feasible. We make all spaces $S[q_1], \ldots, S[q_d]$ alive and we insert them into $\mathcal{A}$. (Note that when $d = 0$, set $S[q]$ is removed from $\mathcal{A}$, but no space is added to it.) This way we ensure that the union of spaces from $\mathcal{A}$ remains equal to the set of all phase-feasible configurations. Note that when a phase ends, $\mathcal{A}$ becomes empty. We emphasize that the evolution of set $\mathcal{A}$ within a phase depends only on the sequence of requests and not on the particular behavior of an online algorithm.

3.4   Factorial Trees: From Hydra Game to Generalized k-Server

Given the framework above, an online algorithm may keep track of the set of alive spaces $\mathcal{A}$, and at all times try to be in a configuration from some alive space. If this space becomes dead, an algorithm changes its configuration to any configuration from some other alive space from $\mathcal{A}$.

The crux is to choose an appropriate next alive space. To this end, our algorithm for the generalized k-server problem will internally run an instance of the Hydra game (a new instance for each phase) on a special tree, and maintain a mapping from alive and dead spaces to alive and dead nodes in the tree. Moreover, spaces that are created during the algorithm runtime, as described in Section 3.3, have to be dynamically mapped to tree nodes that were so far asleep.

In our reduction, we use a k-*factorial* tree. It has height k (the root is on level k and leaves on level 0). Any node on level d has exactly d children, i.e., the subtree rooted at a d-level node has $d!$ leaves, hence the tree name. On the k-factorial tree, the total cost of HERC is $O(k \cdot (1 + \log k!)) = O(k^2 \cdot \log k)$. We now show that this implies an improved algorithm for the generalized k-server problem.

▶ **Theorem 6.** *If there exists a (randomized) online algorithm* H *for the Hydra game on the* k-*factorial tree of total (expected) cost* R, *then there exists a (randomized)* $(R+1)$-*competitive online algorithm* G *for the generalized* k-*server problem in uniform metrics.*

Proof. Let I be an input for the generalized k-server problem in uniform metric spaces. G splits I into phases as described in Section 3.2 and, in each phase, it tracks the phase-feasible nodes using set $\mathcal{A}$ of alive spaces as described in Section 3.3. For each phase, G runs a new instance I_H of the Hydra game on a k-factorial tree T, translates requests from I to adversarial actions in I_H, and reads the answers of H executed on I_H. At all times, G maintains a (bijective) mapping from alive (respectively, dead) d-dimensional configuration spaces to alive (respectively, dead) nodes on the d-th level of the tree T. In particular, at the beginning, the only alive space is the k-dimensional space $S[(\star, \ldots, \star)]$, which corresponds to the tree root (on level k). The configuration of G will always be an element of the space corresponding to the tree node containing H. More precisely, within each phase, a request r is processed in the following way by G.

- Suppose that request r does not make any configuration phase-infeasible. In this case, G services r from its current configuration and no changes are made to $\mathcal{A}$. Also no adversarial actions are executed in the Hydra game.
- Suppose that request r makes some (but not all) configurations phase-infeasible. We assume that this kills only one d-dimensional configuration space $S[q]$. (If r causes multiple configuration spaces to become dead, G processes each such killing event separately, in an arbitrary order.)
  By the description given in Section 3.3, $S[q]$ is then removed from $\mathcal{A}$ and d new $(d-1)$-dimensional spaces $S[q_1], \ldots, S[q_d]$ are added to $\mathcal{A}$. G executes appropriate adversarial actions in the Hydra game: a node v corresponding to $S[q]$ is killed and its d children on level $d-1$ change state from asleep to alive. G modifies the mapping to track the change of $\mathcal{A}$: (new and now alive) spaces $S[q_1], \ldots, S[q_d]$ become mapped to (formerly asleep and now alive) d children of v. Afterwards, G observes the answer of algorithm H on the factorial tree and replays it. Suppose H moves from (now dead) node v to an alive node v', whose corresponding space is $S[q'] \in \mathcal{A}$. In this case, G changes its configuration to the closest configuration (requiring minimal number of server moves) from $S[q']$. It remains to relate its cost to the cost of H. By Lemma 5 (applied to spaces corresponding to all nodes

on the tree path from v to v'), the corresponding k-tuples q, q' differ on at most $\mathsf{dist}(v, v')$ positions. Therefore, adjusting the configuration of G, so that it becomes an element of $S[q']$, requires at most $\mathsf{dist}(v, v')$ server moves, which is exactly the cost of H.

Finally, note that when G processes all killing events, it ends in a configuration of an alive space, and hence it can service the request r from its new configuration.

- Suppose that request r makes all remaining configurations phase-infeasible. In such case, G moves an arbitrary server to service this request, which incurs a cost of 1. In this case, the current phase ends, a new one begins, and G initializes a new instance of the Hydra game.

Let $f \geq 1$ be the number of all phases for input I (the last one may be not finished). The cost of OPT in a single finished phase is at least 1. By the reasoning above, the (expected) cost of G in a single phase is at most $R + 1$. Therefore, $\mathbf{E}[\mathrm{G}(I)] \leq (R+1) \cdot f \leq (R+1) \cdot \mathrm{OPT}(I) + (R+1)$, which completes the proof. ◄

Using our algorithm HERC for the Hydra game along with the reduction given by Theorem 6 immediately implies the following result.

▶ **Corollary 7.** *There exists a randomized $O(k^2 \cdot \log k)$-competitive online algorithm for the generalized k-server problem in uniform metrics.*

4 Lower bound

Next, we show that that competitive ratio of any (even randomized) online algorithm for the generalized k-server problem in uniform metrics is at least $\Omega(k)$, as long as each metric space M_i contains at least two points. For each M_i, we choose two distinct points, the initial position of the i-th server, which we denote 0 and any other point, which we denote 1. The adversary is going to issue only requests satisfying $r_i \in \{0, 1\}$ for all i, hence without loss of generality any algorithm will restrict its server's position in each M_i to 0 and 1. (To see this, assume without loss of generality that the algorithm is lazy, i.e., it is only allowed to move when a request is not covered by any of its server, and is then allowed only to move a single server to cover that request.) For this reason, from now on we assume that $M_i = \{0, 1\}$ for all i, ignoring superfluous points of the metrics.

The configuration of any algorithm can be then encoded using a binary word of length k. It is convenient to view all these 2^k words (configurations) as nodes of the k-dimensional hypercube: two words are connected by a hypercube edge if they differ at exactly one position. Observe that a cost of changing configuration c to c', denoted $\mathsf{dist}(c, c')$ is exactly the distance between c and c' in the hypercube, equal to the number of positions on which the corresponding binary strings differ.

In our construction, we compare the cost of an online algorithm to the cost of an algorithm provided by the adversary. Since OPT's cost can be only lower than the latter, such approach yields a lower bound on the performance of the online algorithm.

For each word w, there is exactly one word at distance k, which we call its *antipode* and denote $\bar{w}$. Clearly, $\bar{w}_i = 1 - w_i$ for all i. Whenever we say that an adversary *penalizes* configuration c, it issues a request at $\bar{c}$. An algorithm that has servers at configuration c needs to move at least one of them. On the other hand, any algorithm with servers at configuration $c' \neq c$ need not move its servers; this property will be heavily used by an adversary's algorithm.

4.1 A Warm-Up: Deterministic Algorithms

To illustrate our general framework, we start with a description of an $\Omega(2^k/k)$ lower bound that holds for any deterministic algorithm DET [2]. (A more refined analysis yields a better lower bound of $2^k - 1$ [16].) The adversarial strategy consists of a sequence of independent identical phases. Whenever DET is in some configuration, the adversary penalizes this configuration. The phase ends when $2^k - 1$ *different* configurations have been penalized. This means that DET was forced to move at least $2^k - 1$ times, at a total cost of at least $2^k - 1$. In the same phase, the adversary's algorithm makes only a single move (of cost at most k) at the very beginning of the phase: it moves to the only configuration that is not going to be penalized in the current phase. This shows that the DET-to-OPT ratio in each phase is at least $(2^k - 1)/k$.

4.2 Extension to Randomized Algorithms

Adopting the idea above to a randomized algorithm RAND is not straightforward. Again, we focus on a single phase and the adversary wants to leave (at least) one configuration non-penalized in this phase. However, now the adversary only knows RAND's probability distribution μ over configurations and not its actual configuration. (At any time, for any configuration c, $\mu(c)$ is the probability that RAND's configuration is equal to c.) We focus on a greedy adversarial strategy that always penalizes the configuration with maximum probability. However, arguing that RAND incurs a significant cost is not as easy as for DET.

First, the support of μ can also include configurations that have been already penalized by the adversary in the current phase. This is but a nuisance, easily overcome by penalizing such configurations repeatedly if RAND keeps using them, until their probability becomes negligible. Therefore, in this informal discussion, we assume that once a configuration c is penalized in a given phase, $\mu(c)$ remains equal to zero.

Second, a straightforward analysis of the greedy adversarial strategy fails to give a non-trivial lower bound. Assume that $i \in \{0, \ldots, 2^k - 2\}$ configurations have already been penalized in a given phase, and the support of μ contains the remaining $2^k - i$ configurations. The maximum probability assigned to one of these configurations is at least $1/(2^k - i)$. When such configuration is penalized, RAND needs to move at least one server with probability at least $1/(2^k - i)$. With such bounds, we would then prove that the algorithm's expected cost is at least $\sum_{i=0}^{2^k-2} 1/(2^k - i) = \Omega(\log 2^k) = \Omega(k)$. Since we bounded the adversary's cost per phase by k, this gives only a constant lower bound.

What we failed to account is that the actual distance traveled by RAND in a single step is either larger than 1 or RAND would not be able to maintain a uniform distribution over non-penalized configurations. However, actually exploiting this property seems quite complex, and therefore we modify the adversarial strategy instead.

The crux of our actual construction is choosing a subset Q of the configurations, such that Q is sufficiently large (we still have $\log(|Q|) = \Omega(k)$), but the minimum distance between any two points of Q is $\Omega(k)$. Initially, the adversary forces the support of μ to be contained in Q. Afterwards, the adversarial strategy is almost as described above, but reduced to set Q only. This way, in each step the support of μ is a set $S \subseteq Q$, and the adversary forces RAND to move with probability at least $1/|S|$ *over a distance at least* $\Omega(k)$, which is the extra $\Theta(k)$ factor. We begin by proving the existence of such a set Q for sufficiently large k. The proof is standard (see, e.g., Chapter 17 of [13]); we give it below for completeness.

▶ **Lemma 8.** *For any $k \geq 16$, there exists a set $Q \subseteq \{0,1\}^k$ of binary words of length k, satisfying the following two properties:*
size property: $|Q| \geq 2^{k/2}/k$,
distance property: $\mathsf{dist}(v,w) \geq k/16$ for any $v,w \in Q$.

Proof. Let $\ell = \lfloor k/16 \rfloor \geq k/32$. For any word q, we define its ℓ-neighborhood $B_\ell(q) = \{w : \mathsf{dist}(q,w) \leq \ell\}$.

We construct set Q greedily. We maintain set Q and set $\Gamma(Q) = \bigcup_{q \in Q} B_\ell(q)$. We start with $Q = \emptyset$ (and thus with $\Gamma(Q) = \emptyset$). In each step, we extend Q with an arbitrary word $w \in \{0,1\}^k \setminus \Gamma(Q)$ and update $\Gamma(Q)$ accordingly. We proceed until set $\Gamma(Q)$ contains all possible length-k words. Clearly, the resulting set Q satisfies the distance property.

It remains to show that $|Q| \geq 2^{k/2}/k$. For a word q, the size of $B_\ell(q)$ is

$$|B_\ell(q)| = \sum_{i=0}^{\lfloor k/16 \rfloor} \binom{k}{i} < k \cdot \binom{k}{\lfloor k/16 \rfloor} \leq k \cdot \left(\frac{k \cdot \mathrm{e}}{\lfloor k/16 \rfloor} \right)^{\lfloor k/16 \rfloor}$$
$$\leq k \cdot \left(\frac{k \cdot \mathrm{e}}{k/32} \right)^{k/16} = k \cdot \left((32 \cdot \mathrm{e})^{1/8} \right)^{k/2} < k \cdot 2^{k/2} .$$

That is, in a single step, $\Gamma(Q)$ increases by at most $k \cdot 2^{k/2}$ elements. Therefore, the process continues for at least $2^k/(k \cdot 2^{k/2}) = 2^{k/2}/k$ steps, and thus the size of Q is at least $2^{k/2}/k$. ◀

▶ **Theorem 9.** *The competitive ratio of every (randomized) online algorithm solving the generalized k-server problem in uniform metrics is at least $\Omega(k)$.*

Proof. In the following we assume that $k \geq 16$, otherwise the theorem follows trivially. We fix any randomized online algorithm RAND. The lower bound strategy consists of a sequence of independent phases. Requests of each phase can be (optimally) serviced with cost at most k and we show that RAND's expected cost for a single phase is $\Omega(k^2)$, i.e., the ratio between these costs is $\Omega(k)$. As the adversary may present an arbitrary number of phases to the algorithm, this shows that the competitive ratio of RAND is $\Omega(k)$, i.e., by making the cost of RAND arbitrarily high, the additive constant in the definition of the competitive ratio (cf. Section 3.1) becomes negligible.

As in our informal introduction, $\mu(c)$ denotes the probability that RAND has its servers in configuration c (at time specified in the context). We extend the notion μ to sets, i.e., $\mu(X) = \sum_{c \in X} \mu(x)$ where X is a set of configurations. We denote the complement of X (to $\prod_{i=1}^{k} M_i$) by X^C. We use $\varepsilon = 2^{-(2k+2)}$ throughout the proof.

To make the description concise, we define an auxiliary routine CONFINE(X) for the adversary (for some configuration set X). In this routine, the adversary repeatedly checks whether there exists a configuration $c \notin X$, such that $\mu(x) > \varepsilon$. In such case, it penalizes c; if no such configuration exists, the routine terminates. We may assume that the procedure always terminates after finite number of steps, as otherwise RAND's competitive ratio would be unbounded. (RAND pays at least ε in each step of the routine while an adversary's algorithm may move its servers to any configuration from set X, and from that time service all requests of CONFINE(X) with no cost.)

The adversarial strategy for a single phase is as follows. First, it constructs Q_1 as the configuration set fulfilling the properties of Lemma 8; let m denote its cardinality. The phase consists then of m executions of CONFINE routine: CONFINE(Q_1), CONFINE(Q_2), ..., CONFINE(Q_m). For $i \in \{2, \ldots, m\}$, set Q_i is defined in the following way. The adversary observes RAND's distribution μ right after routine CONFINE(Q_{i-1}) terminates; at this point this distribution is denoted μ_{i-1}. Then, the adversary picks configuration c_{i-1} to be the element of Q_{i-1} that maximizes the probability μ_{i-1}, and sets $Q_i = Q_{i-1} \setminus \{c_{i-1}\}$.

We begin by describing the way that the adversary services the requests. Observe that set Q_m contains a single configuration, henceforth denoted c^*. The configuration c^* is contained in all sets $Q_1, \ldots, Q_m$, and thus c^* is never penalized in the current phase. Hence, by moving to c^* at the beginning of the phase, which costs at most k, and remaining there till the phase ends, the adversary's algorithm services all phase requests at no further cost.

It remains to lower-bound the cost of RAND. CONFINE(Q_1) may incur no cost; its sole goal is to confine the support of μ to Q_1. Now, we fix any $i \in \{2, \ldots, m\}$ and estimate the cost incurred by CONFINE(Q_i). Recall that the probability distribution right before CONFINE(Q_i) starts (and right after CONFINE(Q_{i-1}) terminates) is denoted μ_{i-1} and the distribution right after CONFINE(Q_i) terminates is denoted μ_i.

During CONFINE(Q_i) a probability mass $\mu_{i-1}(c_{i-1})$, is moved from c_{i-1} to nodes of set Q_i (recall that $Q_i \uplus \{c_{i-1}\} = Q_{i-1}$). Some negligible amounts (at most $\mu_i(Q_i^C)$) of this probability may however remain outside of Q_i after CONFINE(Q_i) terminates. That is, RAND moves at least the probability mass of $\mu_{i-1}(c_{i-1}) - \mu_i(Q_i^C)$ from configuration c_{i-1} to configurations from Q_i (i.e., along a distance of at least $\mathsf{dist}(c_{i-1}, Q_i)$), Therefore, its expected cost due to CONFINE(Q_i) is at least $(\mu_{i-1}(c_{i-1}) - \mu_i(Q_i^C)) \cdot \mathsf{dist}(c_{i-1}, Q_i)$.

First, using the properties of CONFINE(Q_{i-1}) and the definition of c_{i-1}, we obtain

$$\mu_{i-1}(c_{i-1}) \geq \frac{\mu_{i-1}(Q_{i-1})}{|Q_{i-1}|} = \frac{1 - \mu_{i-1}(Q_{i-1}^C)}{|Q_{i-1}|} \geq \frac{1 - |Q_{i-1}^C| \cdot \varepsilon}{|Q_{i-1}|} > \frac{1 - 2^{-(k+2)}}{|Q_{i-1}|} \, . \tag{3}$$

Second, using the properties of CONFINE(Q_i) yields

$$\mu_i(Q_i^C) \leq |Q_i^C| \cdot \varepsilon < 2^{-(k+2)} = \frac{2^{-2}}{2^k} < \frac{2^{-2}}{|Q_{i-1}|} \, . \tag{4}$$

Using (3) and (4), we bound the expected cost of RAND due to routine CONFINE(Q_i) as

$$\begin{aligned}
\mathbf{E}[\text{RAND}(\text{CONFINE}(Q_i))] &\geq \left(\mu_{i-1}(c_{i-1}) - \mu_i(Q_i^C) \right) \cdot \mathsf{dist}(c_{i-1}, Q_i) \\
&\geq \left(\frac{1 - 2^{-(k+2)}}{|Q_i|} - \frac{2^{-2}}{|Q_i|} \right) \cdot \frac{k}{16} \geq \frac{1}{2 \cdot |Q_i|} \cdot \frac{k}{16} \\
&= k/(32 \cdot (m - i + 1))
\end{aligned} \tag{5}$$

The second inequality above follows as all configurations from $\{c_{i-1}\} \uplus Q_i$ are distinct elements of Q_1, and hence their mutual distance is at least $k/16$ by the distance property of Q_1 (cf. Lemma 8). By summing (5) over $i \in \{2, \ldots, m\}$, we obtain that the total cost of RAND in a single phase is $\mathbf{E}[\text{RAND}] \geq \sum_{i=2}^m \mathbf{E}[\text{RAND}(\text{CONFINE}(Q_i))] \geq \frac{k}{32} \cdot \sum_{i=2}^m \frac{1}{m-i+1} = \Omega(k \cdot \log m) = \Omega(k^2)$. The last equality holds as $m \geq 2^{k/2}/k$ by the size property of Q_1. (cf. Lemma 8). ◀

5   Final remarks

In this paper, we presented an abstract Hydra game whose solution we applied to create an algorithm for the generalized k-server problem. Any improvement of our HERC strategy for the Hydra game would yield an improvement for the generalized k-server problem. However, we may show that on a wide class of trees (that includes factorial trees used in our reduction), HERC is optimal up to a constant factor. Thus, further improving our upper bound of $O(k^2 \log k)$ for the generalized k-server problem will require another approach.

A lower bound for the cost of any randomized strategy for the Hydra game is essentially the same as our single-phase construction from Section 4.2 for the generalized k-server problem. That is, the adversary fixes a subset Q of tree leaves, makes only nodes of Q alive

(this forces the algorithm to be inside set Q), and then iteratively kills nodes of Q where the algorithm is most likely to be. As in the proof from Section 4.2, such adversarial strategy incurs the cost of $\Omega(\mathsf{mindist}(Q) \cdot \log |Q|)$, where $\mathsf{mindist}(Q) = \min_{u \neq v \in Q} \mathsf{dist}(u, v)$.

The construction of appropriate Q for a tree T of depth $k = h_T$ (be either the k-factorial tree or the complete k-ary tree) is as follows. Let Z be the set of all nodes of T at level $\lfloor k/2 \rfloor$; for such trees, $\log |Z| = \Omega(\log L_T)$. Let Q consist of $|Z|$ leaves of the tree, one per node of Z chosen arbitrarily from its subtree. Then, $\mathsf{mindist}(Q) = \Omega(h_T)$ and $\log |Q| = \Omega(\log L_T)$, and thus the resulting lower bound $\Omega(h_T \cdot \log L_T)$ on the cost asymptotically matches the performance of HERC from Theorem 4.

References

1 Nikhil Bansal, Marek Eliás, and Grigorios Koumoutsos. Weighted k-Server Bounds via Combinatorial Dichotomies. In *Proc. 58th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 493–504. IEEE Computer Society, 2017.

2 Nikhil Bansal, Marek Eliás, Grigorios Koumoutsos, and Jesper Nederlof. Competitive Algorithms for Generalized *k*-Server in Uniform Metrics. In *Proc. 29th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 992–1001, 2018.

3 Yair Bartal, Béla Bollobás, and Manor Mendel. Ramsey-type theorems for metric spaces with applications to online problems. *J. Comput. Syst. Sci.*, 72(5):890–921, 2006.

4 Yair Bartal, Nathan Linial, Manor Mendel, and Assaf Naor. On metric Ramsey-type phenomena. In *Proc. 35th ACM Symp. on Theory of Computing (STOC)*, pages 463–472, 2003.

5 Alan Borodin, Nati Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *Journal of the ACM*, 39(4):745–763, 1992.

6 Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. k-server via multiscale entropic regularization. In *Proc. 50th ACM Symp. on Theory of Computing (STOC)*, pages 3–16. ACM, 2018.

7 Ashish Chiplunkar and Sundar Vishwanathan. On Randomized Memoryless Algorithms for the Weighted K-Server Problem. In *Proc. 54th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 11–19, 2013.

8 Marek Chrobak. SIGACT news online algorithms col. 1. *SIGACT News*, 34(4):68–77, 2003.

9 Marek Chrobak, Howard J. Karloff, Thomas H. Payne, and Sundar Vishwanathan. New Results on Server Problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, 1991.

10 Marek Chrobak and Lawrence L. Larmore. An Optimal On-Line Algorithm for k-Servers on Trees. *SIAM Journal on Computing*, 20(1):144–148, 1991.

11 Marek Chrobak and Jirí Sgall. The weighted 2-server problem. *Theoretical Computer Science*, 324(2-3):289–312, 2004.

12 Amos Fiat and Moty Ricklin. Competitive Algorithms for the Weighted Server Problem. *Theoretical Computer Science*, 130(1):85–99, 1994.

13 Stasys Jukna. *Extremal Combinatorics*. Springer, 2011.

14 Elias Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009.

15 Elias Koutsoupias and Christos H. Papadimitriou. On the k-Server Conjecture. *Journal of the ACM*, 42(5):971–983, 1995.

16 Elias Koutsoupias and David Scot Taylor. The CNN problem and other k-server variants. *Theoretical Computer Science*, 324(2-3):347–359, 2004.

17 James R. Lee. Fusible HSTs and the Randomized k-Server Conjecture. In *Proc. 59th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 438–449, 2018.

18 Mark S. Manasse, Lyle A. McGeoch, and Daniel D. Sleator. Competitive algorithms for server problems. *Journal of the ACM*, 11(2):208–230, 1990.

19 René Sitters. The Generalized Work Function Algorithm Is Competitive for the Generalized 2-Server Problem. *SIAM Journal on Computing*, 43(1):96–125, 2014.

20 René A. Sitters and Leen Stougie. The generalized two-server problem. *Journal of the ACM*, 53(3):437–458, 2006.

21 Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

A   Probability Distribution and Algorithms

When we described our algorithm HERC for the Hydra game, we assumed that its current position in the tree is a random node with probability distribution given by η. In a single step, HERC decreases probability at some node u from $\eta(u)$ to zero and increases the probabilities of some other nodes $w_1, \ldots, w_\ell$ by a total amount of $\eta(u)$. Such change can be split into ℓ elementary changes, each decreasing the probability at node u by p_i and increasing it at node w_i by the same amount. Each elementary change can be executed and analyzed as shown in the following lemma.

▶ **Lemma 10.** *Let η be a probability distribution describing the position of* ALG *in the tree. Fix two tree nodes, u and w. Suppose η' is a probability distribution obtained from η by decreasing $\eta(u)$ by p and increasing $\eta(w)$ by p. Then,* ALG *can change its random position, so that it will be described by η', and the expected cost of such change is $p \cdot \mathsf{dist}(u, w)$.*

Proof. We define ALG's action as follows: if ALG is at node u, then with probability $p/\eta(u)$ it moves to node w. If ALG is at some other node it does not change its position.

We observe that the new distribution of ALG is exactly η'. Indeed, the probability of being at node u decreases by $\eta(u) \cdot p/\eta(u) = p$, while the probability of being at node w increases by the same amount. The probabilities for all nodes different than u or w remain unchanged.

Furthermore, the probability that ALG moves is $\eta(u) \cdot (p/\eta(u)) = p$ and the traveled distance is $\mathsf{dist}(u, w)$. The expected cost of the move is then $p \cdot \mathsf{dist}(u, w)$, as desired.   ◀