Competitive Algorithms for Generalized k-Server in Uniform Metrics

NIKHIL BANSAL, University of Michigan, USA
MAREK ELIÁŠ, Bocconi University, Italy
GRIGORIOS KOUMOUTSOS, Lightcurve GmbH, Germany
JESPER NEDERLOF, Utrecht University, The Netherlands

The generalized k-server problem is a far-reaching extension of the k-server problem with several applications. Here, each server s_i lies in its own metric space M_i. A request is a k-tuple $r = (r_1, r_2, \ldots, r_k)$, which is served by moving some server s_i to the point $r_i \in M_i$, and the goal is to minimize the total distance traveled by the servers. Despite much work, no $f(k)$-competitive algorithm is known for the problem for $k > 2$ servers, even for special cases such as uniform metrics and lines.

Here, we consider the problem in uniform metrics and give the first $f(k)$-competitive algorithms for general k. In particular, we obtain deterministic and randomized algorithms with competitive ratio $k \cdot 2^k$ and $O(k^3 \log k)$ respectively. Our deterministic bound is based on a novel application of the polynomial method to online algorithms, and essentially matches the long-known lower bound of $2^k - 1$. We also give a $2^{2^{O(k)}}$-competitive deterministic algorithm for weighted uniform metrics, which also essentially matches the recent doubly exponential lower bound for the problem.

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

Additional Key Words and Phrases: k-server problem, online algorithms, competitive analysis

ACM Reference Format:
Nikhil Bansal, Marek Eliáš, Grigorios Koumoutsos, and Jesper Nederlof. 2023. Competitive Algorithms for Generalized k-Server in Uniform Metrics. 1, 1 (January 2023), 15 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The k-server problem was proposed by Manasse et al. [28] as a far-reaching generalization of many online problems, and its study has led to various remarkable developments in online computation. Here, we are given k-servers $s_1, \ldots, s_k$ located at points of a metric space M. At each time step a request arrives at some point of M and must be served by moving some server there. The goal is to minimize the total distance traveled by the servers.

A preliminary version of this article appeared in the Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA) 2018. This work was carried out while authors were affiliated with TU Eindhoven.
The first author was supported by a NWO Vici grant 639.023.812. The second author was supported by a NWO Vidi grant 639.022.211. The third author was supported by ERC consolidator grant 617951 and by FNRS Grant MISU F 6001 1. The last author was supported by NWO Veni project 639.021.438.
Authors' addresses: Nikhil Bansal, University of Michigan, USA, bansal@gmail.com; Marek Eliáš, Bocconi University, Italy, marek.elias@unibocconi.it; Grigorios Koumoutsos, Lightcurve GmbH, Berlin, Germany, gregkoumoutsos@gmail.com; Jesper Nederlof, Utrecht University, Utrecht, The Netherlands, j.nederlof@uu.nl.

XXXX-XXXX/2023/1-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Koutsoupias and Taylor [27] introduced a substantial generalization of the k-server problem, called the *generalized k-server problem.* Here, each server s_i lies in its own metric space M_i, with its own distance function d_i. A request is a k-tuple $r = (r_1, r_2, \ldots, r_k)$ and must be served by moving some server s_i to the point $r_i \in M_i$. Note that the standard k-server problem corresponds to the special case when all the metrics are identical, $M_1 = \ldots = M_k = M$, and the requests are of the form $(r, r, \ldots, r)$, i.e., the k-tuple is identical in each coordinate.

The generalized k-server problem can model a rich class of online problems, for which the techniques developed for the standard k-server problem do not apply, see e.g. [27]. For that reason, it is widely believed that a deeper understanding of this problem should lead to powerful new techniques for designing online algorithms [27, 31]. According to Koutsoupias and Taylor [27], this problem "may act as a stepping stone towards building a robust (and less ad hoc) theory of online computation".

1.1 Previous Work

The k-server problem. The k-server problem has been extensively studied (an excellent reference is [8]). The initial work focused on special metrics such as uniform metrics and lines, and optimum competitive ratios were obtained in many cases [14, 15, 26]. A particularly interesting case is that of uniform metrics, which corresponds to the very well-studied *paging* problem, where tight k-competitive deterministic [34] and $O(\log k)$-competitive randomized algorithms [1, 18, 30] are known.

For general metrics, Koutsoupias and Papadimitriou [25] showed in a breakthrough result that the Work Function Algorithm (WFA) is $(2k-1)$-competitive in any metric space. This essentially matches the lower bound of k for any deterministic algorithm [28]. For randomized algorithms the competitive ratio is $\Omega(\log k)$ and it is widely believed that there is a $O(\log k)$-competitive randomized algorithm. While this remains open, polylog(k, n) randomized competitive algorithms were obtained, where n in the number of points in M [4, 10].

The generalized k-server problem. This problem is much less understood. In their seminal paper, Koutsoupias and Taylor [27] studied the special case where $k = 2$ and both the metrics M_1 and M_2 are lines. This is called CNN problem and it has attracted a lot of attention [2, 13, 21, 22]. They showed that, even for this special case, many successful k-server algorithms or their natural generalizations are not competitive.

Lower Bounds. For uniform metrics, Koutsoupias and Taylor [27] showed that even when each M_i contains $n = 2$ points, the competitive ratio is at least $2^k - 1$. For general metrics, the best known lower bound is $2^{2^{\Omega(k)}}$ [5], and comes from the weighted k-server problem (the weighted variant of the standard k-server problem). This problem corresponds to generalized-k-server where the metric spaces are scaled copies of each other, i.e. $M_i = w_i M$ for some fixed M, and the requests have the form $(r, \ldots, r)$.

Upper Bounds. Despite considerable efforts, competitive algorithms[1] are known only for the case of $k = 2$ servers [31–33]. In a breakthrough result, Sitters and Stougie [33] obtained a $O(1)$-competitive algorithm for $k = 2$ in any metric space. Recently, Sitters [31] showed that the generalized WFA is also $O(1)$-competitive for $k = 2$ by a careful and subtle analysis of the structure of work functions. Despite this progress, no $f(k)$-competitive algorithms are known for $k > 2$, even for special cases such as uniform metrics and lines.

[1] We focus on algorithms with competitive ratio $f(k)$ that only depends on k. Note that an $n^k - 1$ competitive algorithm follows trivially, as the problem can be viewed as Metrical Service System (MSS) on n^k states, where $n = \max_{i=1}^{k} |M_i|$.

1.2 Our Results

We consider the generalized k-server problem on uniform metrics and obtain the first $f(k)$-competitive algorithms for general k, with competitive ratios close to the known lower bounds.

Perhaps surprisingly, there turn out to be two very different settings for uniform metrics:

(1) When all the metric spaces $M_1, \ldots, M_k$ are uniform (possibly with different number of points) with identical pairwise distance, say 1. We call this the *uniform metric* case.
(2) When the metric spaces M_i are all uniform, but have different scales, i.e. all pairwise distances in M_i are w_i. We call this the *weighted uniform metric* case.

Our first result is the following.

Theorem 1.1. *There is a $(k \cdot 2^k)$-competitive deterministic algorithm for the generalized k-server problem in the uniform metric case.*

This almost matches the $2^k - 1$ lower bound due to [27] (we describe this instructive and simple lower bound instance in the Appendix for completeness).

The proof of Theorem 1.1 is based on a general combinatorial argument about how the set of feasible states evolves as requests arrive. Specifically, we divide the execution of the algorithm in phases, and consider the beginning of a phase when all the MSS states are feasible (e.g. the cost is 0 and not ∞). As requests arrive, the set of states that remain valid for all requests during this phase can only reduce. In particular, for this problem we show that any sequence of requests that causes the feasible state space to strictly reduce at each step, can have length at most 2^k until all states becomes infeasible.

Interestingly, this argument is based on a novel application of the polynomial or the rank method from linear algebra [20, 24, 29]. While the rank method has led to some spectacular recent successes in combinatorics and computer science [16, 17], we are not aware of any previous applications to online algorithms. We feel our approach could be useful for other online problems that can be modeled as Metrical Service Systems by analyzing the combinatorial structure in a similar way.

Next, we consider randomized algorithms against oblivious adversaries.

Theorem 1.2. *There is a randomized algorithm for the generalized k-server problem on uniform metrics with competitive ratio $O(k^3 \log k)$.*

The rank method above does not seem to be useful in the randomized setting as it only bounds the number of requests until the set of feasible states becomes empty, and does not give any structural information about how the set of states evolves over time. As we observe in Section 3, a $o(2^k)$ guarantee cannot be obtained without using such structural information. So we explore the properties of this evolution more carefully and use it to design the randomized algorithm in Theorem 1.2.

Finally, we consider the weighted uniform metric case.

Theorem 1.3. *There is a $2^{2^{k+3}}$ competitive algorithm for generalized k-server on weighted uniform metrics.*

Theorem 1.3 follows by observing that a natural modification of an algorithm due to Fiat and Ricklin [19] for weighted k-server on uniform metrics also works for the more general generalized k-server setting. Our proof is essentially the same as that of [19], with some arguments streamlined and an improved competitive ratio[2]. Finally, note that the $2^{2^{\Omega(k)}}$ lower bound [5] for weighted k-server on uniform metrics implies that Theorem 1.3 is essentially optimal.

[2]It was first pointed out to us by Chiplunkar [11] that the competitive ratio $2^{2^{4k}}$ claimed in [19] can be improved to $2^{2^{k+O(1)}}$.

Recent Developments. Since the initial announcement of this work [6], Bienkowski et. al. [7] considered the generalized k-server on uniform metrics, focusing on randomized algorithms. They obtained a $O(k^2 \log k)$-competitive randomized algorithm by modifying our algorithm from Theorem 1.2, and also gave a lower bound of $\Omega(k)$. Chiplunkar and Vishwanathan [12] focused on randomized memoryless algorithms against oblivious adversaries for the weighted uniform case and showed tight doubly-exponential bounds on the competitive ratio.

2 DETERMINISTIC ALGORITHM FOR UNIFORM METRICS

In this section we prove Theorem 1.1. Recall that each M_i is the uniform metric with unit distance. We assume that all metrics have $n = \max_{i=1}^k |M_i|$ points (if for some metric $|M_i| < n$, we can add some extra points that are never requested). We use $[n]$ to denote $\{1, \dots, n\}$. As the requests are arbitrary k-tuples and each metric M_i is uniform, we can relabel the points arbitrarily and hence assume that the set of points in each M_i is $[n]$. At any time t, the state of an algorithm can be described by the k-tuple $q^t = (q_1^t, \dots, q_k^t)$ where for each $i \in [k]$, $q_i^t \in [n]$ denotes the location of server i. Let $r^t = (r_1^t, \dots, r_k^t)$ denote the request vector at time t. Given such a request, we need to move to some state q^t satisfying the following:

Definition 2.1. *A state q^t* satisfies (or is feasible for) *the request r^t if $q_i^t = r_i^t$ for some $i \in [k]$.*

Moreover, if the state changes from q^t to q^{t+1}, the algorithm pays the Hamming distance

$$d(q^{t+1}, q^t) = |\{i : q_i^{t+1} \neq q_i^t\}|,$$

between q^t and q^{t+1}.

Algorithm. Consider the following generic algorithm, that works in phases and upon each request, moves to some arbitrary location (if it exists) that satisfies *all* the requests in the current phase thus far. If no such location exists, it moves to some location satisfying the current request and the next phase begins. We call this algorithm *generic* as it can pick any arbitrary point q as long as it satisfies

Algorithm 1: A deterministic $k \cdot 2^k$ competitive algorithm.

If a phase begins, the algorithm starts in some arbitrary q^1.
At each time t when a request r^t arrives do the following.
if the current state q^t does not satisfy the current request r^t **then**
  if there exists a state q that satisfies all requests $r^1, \dots, r^t$ **then**
    Set $q^{t+1} = q$.
  else
    Set q^{t+1} to be an arbitrary location satisfying (only) r^t.
    End the current phase.
else
  Set $q^{t+1} = q^t$.

all the requests of the current phase $r^1, \dots, r^t$. Note that this captures a wide variety of natural algorithms; e.g. for the special case of the paging problem it behaves exactly as the well-known Marking algorithm.

Analysis. Clearly, during each phase the offline pays at least 1, as by the definition of a phase, no location can satisfies all the requests that arrive during the phase. We will show that the online algorithm can change its state at most 2^k times and hence pay at most $k2^k$ as the Hamming distance between any two states is at most k. implying the desired competitive ratio of $k2^k$.

Fix some phase that we wish to analyze, and let ℓ denote its length. Without loss of generality, we can assume that r^t always causes q^t to move (removing such requests does not reduce the online cost, and can only help the offline adversary). So the online algorithm moves exactly ℓ times. It suffices to show the following.

Theorem 2.2. *For any phase as defined above, its length satisfies $\ell \leq 2^k$.*

Proof. We use the rank method. Let $x = (x_1, \ldots, x_k), y = (y_1, \ldots, y_k)$, and consider the $2k$-variate degree k polynomial $p : \mathbb{R}^{2k} \to \mathbb{R}$,

$$p(x, y) := \prod_{i \in [k]} (x_i - y_i).$$

The key property of the polynomial p is that a state $q \in [n]^k$ satisfies a request $r \in [n]^k$ iff $p(q, r) = 0$.

We now construct a matrix M that captures the dynamics of the online algorithm during a phase. Let $M \in \mathbb{R}^{\ell \times \ell}$ be an $\ell \times \ell$ matrix, where columns correspond to the requests and rows to the states, with entries $M[t, t'] = p(q^t, r^{t'})$, i.e., the $[t, t']$ entry of M corresponds to the evaluation of the polynomial p on q^t and $r^{t'}$

Claim 2.3. *M is an upper triangular matrix with non-zero diagonal. In particular, M has rank ℓ.*

Proof. At any time $t = 1, \ldots, \ell$, as the current state q^t does not satisfy the request r^t, it must be that $p(q^t, r^t) \neq 0$. On the other hand, for $t = 2, \ldots, \ell$, the state q^t was chosen such that it satisfied all the previous requests t' for $t' < t$. This gives that $M[t, t'] = 0$ for $t' < t$ and hence all the entries below the diagonal are 0.

As the determinant of any upper-triangular matrix is the product of its diagonal entries, M has non-zero determinant and hence $\mathrm{rk}(M) = \ell$. □

So it suffices to show that the rank of any such matrix M is at most 2^k.[3].

Claim 2.4. *For any set of requests r^t and points q^t chosen by the algorithm, for $t = 1, 2, \ldots, \ell$, where ℓ is arbitrary, the matrix M has rank at most 2^k.*

Proof. We give an explicit factorization of M as $M = AB$, where A is $\ell \times 2^k$ matrix and M is a $2^k \times \ell$ matrix. As any $m \times n$ matrix has rank at most $\min(m, n)$, both A and B have rank at most 2^k. Moreover, as $\mathrm{rk}(AB) \leq \min(\mathrm{rk}(A), \mathrm{rk}(B))$, this implies $\mathrm{rk}(M) \leq 2^k$.

To obtain the factorization, we note that $p(x, y)$ can be expressed in terms of its 2^k monomials as

$$p(x, y) = \sum_{S \subseteq [k]} (-1)^{k-|S|} X_S Y_{[k] \setminus S},$$

where $X_S = \prod_{i \in S} x_i$ with $X_\emptyset = 1$, and Y_S is defined analogously.

Let A be the $\ell \times 2^k$ matrix with rows indexed by time t and columns by subsets $S \in 2^{[k]}$, with the entries

$$A[t, S] = q^t_S := \prod_{i \in S} q^t_i,$$

[3]Curiously, this particular rank upper bound was used in a previous work for answering a question in a completely different setting about the parameterized complexity of graph coloring parameterized by cutwidth [23].

and B be the $2^k \times \ell$ matrix with rows indexed by subsets $S \in 2^{[k]}$ and columns indexed by time t' with entries

$$B[S,t'] = (-1)^{k-|S|}\, r^{t'}_{[k]\setminus S} := (-1)^{k-|S|} \prod_{i\in[k]\setminus S} r^{t'}_i .$$

Then, for any $t, t' \in [\ell]$,

$$M[t,t'] = p(q^t, r^{t'}) = \sum_{S\subseteq[k]} (-1)^{k-|S|}\, q^t_S\, r^{t'}_{[k]\setminus S} = \sum_{S\subseteq[k]} A[t,S]B[S,t'] = (AB)[t,t'].$$

implying that $M = AB$, and giving the claimed factorization. □

The result now follows from Claims 2.3 and 2.4. □

3 RANDOMIZED ALGORITHM FOR UNIFORM METRICS

A natural way to randomize the algorithm above would be to pick a state uniformly at random among all the states that are feasible for all the requests thus far in the current phase. The standard randomized uniform MTS analysis [9] implies that this online algorithm would move $O(\log(n^k)) = O(k \log n)$ times. However, this guarantee is not useful if $n \gg \exp(\exp(k))$.

Perhaps surprisingly, even if we use the fact from Section 2 that the set of feasible states can shrink at most 2^k times, this does not suffice to give a randomized $o(2^k)$ guarantee. Indeed, consider the algorithm that picks a random state among the feasible ones in the current phase. If, at each step $t = 1, \dots, \ell$, half of the feasible states become infeasible (expect the last step when all states become infeasible), then the algorithm must move with probability at least $1/2$ at each step, and hence incur an expected $\Omega(\ell) = \Omega(2^k)$ cost during the phase.

So proving a better guarantee would require showing that the scenario above cannot happen. In particular, we need a more precise understanding of how the set of feasible states evolves over time, rather than simply a bound on the number of requests in a phase.

To this end, in Lemmas 3.1 and 3.3 below, we impose some stronger subspace-like structure over the set of feasible states. Then, we use this structure to design a variant of the natural randomized algorithm above, that directly works with these subspaces.

Spaces of configurations. Let U_i denote the set of points in M_i. We can think of $U_i = [n]$, but U_i makes the notation clear. We call state in $\prod_{i=1}^k U_i = [n]^k$ a configuration. Here we slightly abuse notation by letting $\prod$ denote the generalized Cartesian product. It will be useful to consider sets of configurations where some server locations are fixed at some particular location. For a vector $v \in \prod_{i=1}^k (U_i \cup \{*\})$, we define the space

$$S(v) := \left\{ c \in \prod_{i=1}^k U_i \;\middle|\; c_i = v_i \ \forall i \text{ s.t. } v_i \neq * \right\}.$$

A coordinate i with $v_i = *$ is called *free* and the corresponding server can be located at an arbitrary point of U_i. The number of free coordinates in the space $S(v)$ we call dimension and denote it with $\dim(S(v))$.

Let us consider a d-dimensional space S and a request r such that some configuration $c \in S$ is not feasible for r. Then, we claim that a vast majority of configurations from S are infeasible for r, as stated in the following lemma. We denote $F(r)$ the set of configuration satisfying r.

Lemma 3.1. *Let S be a d-dimensional space and let r be a request which makes some configuration $c \in S$ infeasible. Then, there exist d subspaces $S_1, \dots, S_d$, each of dimension $d-1$, such that we have $S \cap F(r) = S_1 \cup \dots \cup S_d$.*

Note that if all the metric spaces U_i contain n points, then $|S_i| = \frac{1}{n}|S|$ for each $i = 1, \dots, d$.

Proof. By reordering the coordinates, we can assume that the first d coordinates of S are free and S corresponds to the vector $(*, \dots, *, s_{d+1}, \dots, s_k)$, for some $s_{d+1}, \dots, s_k$. Let $r = (r_1, \dots, r_k)$.

Consider the subspaces $S(v_1), \dots, S(v_d)$, where

$$\begin{aligned} v_1 &= (r_1, *, \dots, *, s_{d+1}, \dots, s_k), \\ &\vdots \\ v_d &= (*, \dots, *, r_d, s_{d+1}, \dots, s_k). \end{aligned}$$

Clearly, any configuration contained in $S(v_1) \cup \ldots \cup S(v_d)$, is feasible for r. Conversely, as there exists $c \in S$ infeasible for r, we have $s_i = c_i \neq r_i$ for each $i = d+1, \dots, k$. This already implies that each configuration from S feasible for r must belong to $S(v_1) \cup \ldots \cup S(v_d)$: whenever $c' \in S$ is feasible for r, it needs to have $c'_i = r_i$ for some $i \in \{1, \dots, d\}$ and therefore $c' \in S(v_i)$. □

Spaces of feasible configurations. During each phase, we maintain a set $\mathcal{F}^t$ of spaces containing configurations which were feasible with respect to the requests $r^1, \dots, r^t$. In the beginning of the phase, we set $\mathcal{F}^1 = \{(r_1^1, *, \dots, *), \dots, (*, \dots, *, r_k^1)\}$, and, at time t, we update it in the following way. We remove all spaces of dimension 0 whose single configuration is infeasible w.r.t. r^t. In addition, we replace each $S \in \mathcal{F}^{t-1}$ of dimension $s > 0$ which contains some infeasible configuration by $S_1, \dots, S_d$ according to the Lemma 3.1. The following observation follows easily from Lemma 3.1.

Observation 3.2. *Let us consider a phase with requests $r^1, \dots, r^\ell$. A configuration c is feasible with respect to the requests $r^1, \dots, r^t$ if and only if c belongs to some space in $\mathcal{F}^t$.*

An alternative deterministic algorithm. Based on $\mathcal{F}^t$, we can design an alternative deterministic algorithm that has a competitive ratio of $3k!$. This is worse than Algorithm 1 but will be very useful to obtain our randomized algorithm. To serve a request at time t, it chooses some space $Q^t \in \mathcal{F}^t$ and moves to an arbitrary $q^t \in Q^t$. Whenever Q^{t-1} no more belongs to $\mathcal{F}^t$, it moves to another space Q^t regardless whether q^{t-1} stayed feasible or not, see Algorithm 2 for details. While, this is not an optimal behaviour, a simple exploitation of the structure of $\mathcal{F}^t$ already gives a reasonably good algorithm.

Algorithm 2: Alternative deterministic algorithm.

at time t:
foreach $S \in \mathcal{F}^{t-1}$ containing some infeasible configuration **do** // update $\mathcal{F}^t$ for r^t
  replace S by $S_1, \dots, S_d$ according to Lemma 3.1
if $\mathcal{F}^t = \emptyset$ **then** // start a new phase, // if needed
  $\mathcal{F}^t := \{S((r_1^t, *, \dots, *)), \dots, S((*, \dots, *, r_k^t))\}$
if $Q^{t-1} \in \mathcal{F}^t$ **then** // serve the request
  set $Q^t := Q^{t-1}$ and $q^t := q^{t-1}$
else
  choose arbitrary $Q^t \in \mathcal{F}^t$ and move to an arbitrary $q^t \in Q^t$

The following lemma bounds the maximum number of distinct spaces which can appear in $\mathcal{F}^t$ during one phase. In fact, it already implies that the competitive ratio of Algorithm 2 is at most $k \cdot k! \cdot \sum_{d=0}^{k-1} \frac{1}{d!} \leq 3kk! \leq 3(k+1)!$.

Lemma 3.3. *Let us consider a phase with requests $r^1, \ldots, r^\ell$. Then $\bigcup_{t=1}^{\ell} \mathcal{F}^t$ contains at most $k!/d!$ spaces of dimension d.*

Proof. We proceed by induction on d. In the beginning, we have $k = k!/(k-1)!$ spaces of dimension $k-1$ in $\mathcal{F}^1$ and, by Lemma 3.1, all spaces added later have strictly lower dimension.

By the way $\mathcal{F}^t$ is updated, each $(d-1)$-dimensional space is created from some d-dimensional space already present in $\bigcup_{t=1}^{\ell} \mathcal{F}^t$. By the inductive hypothesis, there could be at most $k!/d!$ distinct d-dimensional spaces and Lemma 3.1 implies that each of them creates at most d distinct $(d-1)$-dimensional spaces. Therefore, there can be at most $\frac{k!}{d!}d = \frac{k!}{(d-1)!}$ spaces of dimension $d-1$ in $\bigcup_{t=1}^{\ell} \mathcal{F}^t$. □

Randomized algorithm. Now we transform Algorithm 2 into a randomized one. Let m_t denote the largest dimension among all the spaces in $\mathcal{F}^t$ and let $\mathcal{M}^t$ denote the set of spaces of dimension m_t in $\mathcal{F}^t$.

The algorithm works as follows: Whenever moving, it picks a space Q^t from $\mathcal{M}^t$ uniformly at random, and moves to some arbitrary $q^t \in Q^t$. As the choice of q^t is arbitrary, whenever some configuration from Q^t becomes infeasible, the algorithm assumes that q^t is infeasible as well[4].

Algorithm 3: Randomized Algorithm for Uniform metrics.

at time t:
foreach $S \in \mathcal{F}^{t-1}$ containing some infeasible configuration **do** `// update` $\mathcal{F}^t$ `for` r^t
  replace S by $S_1, \ldots, S_d$ according to Lemma 3.1
if $\mathcal{F}^t = \emptyset$ **then** `// start a new phase, // if needed`
  $\mathcal{F}^t := \{S((r_1^t, *, \ldots, *)), \ldots, S((*, \ldots, *, r_k^t))\}$
if $Q^{t-1} \in \mathcal{M}^t$ **then** `// serve the request`
  set $Q^t := Q^{t-1}$ and $q^t := q^{t-1}$
else
  Choose a space Q^t from $\mathcal{M}^t$ uniformly at random
  Move to an arbitrary $q^t \in Q^t$

At each time t, ALG is located at some configuration q^t contained in some space in $\mathcal{F}^t$ which implies that its position is feasible with respect to the current request r^t, see Observation 3.2. Here is the key property about the state of ALG.

Lemma 3.4. *At each time t, the probability of Q^t being equal to some fixed $S \in \mathcal{M}^t$ is $1/|\mathcal{M}^t|$.*

Proof. If ALG moved at time t, the statement follows trivially, since Q^t was chosen from $\mathcal{M}^t$ uniformly at random. So, let us condition on the event that $Q^t = Q^{t-1}$.

Now, the algorithm does not change state if and only if $Q^{t-1} \in \mathcal{M}^t$. Moreover, in this case m_t does not change, and $\mathcal{M}^t \subset \mathcal{M}^{t-1}$. By induction, Q^{t-1} is distributed uniformly within $\mathcal{M}^{t-1}$, and hence conditioned on $Q^{t-1} \in \mathcal{M}^t$, Q^t is uniformly distributed within $\mathcal{M}^t$. □

[4]This is done to keep the calculations simple, as the chance of Q^t being removed from $\mathcal{F}$ and q^t staying feasible is negligible when $k \ll n$.

Proof of Theorem 1.2. At the end of each phase (except possibly for the last unfinished phase), the set of feasible states $\mathcal{F}^t = \emptyset$, and hence OPT must pay at least 1 during each of those phases. Denoting N the number of phases needed to serve the entire request sequence, we have $\mathrm{cost}(\mathrm{OPT}) \geq (N-1)$. On the other hand, the expected online cost is at most,

$$E[\mathrm{cost}(\mathrm{ALG})] \leq c(N-1) + c \leq c\,\mathrm{cost}(\mathrm{OPT}) + c,$$

where c denotes the expected cost of ALG in one phase. This implies that ALG is c-competitive, and strictly $2c$-competitive (as the offline must move at least once, if the online algorithm pays a non-zero cost).

Now we prove that c is at most $O(k^3 \log k)$. To show this, we use a potential function

$$\Phi(t) = H(|\mathcal{M}^t|) + \sum_{d=0}^{m_t - 1} H(k!/d!),$$

where $H(n)$ denotes the nth harmonic number. As the beginning of the phase, $\Phi(1) \leq kH(k!) \leq k(\log k! + 1) = O(k^2 \log k)$ as $|\mathcal{M}^1| \leq k!$ and $m_1 \leq k-1$. Moreover the phase ends whenever $\Phi(t)$ decreases to 0. Therefore, it is enough to show that, at each time t, the expected cost incurred by the algorithm is at most k times the decrease of the potential. We distinguish two cases.

If $m_t = m_{t-1}$, let us denote $b = |\mathcal{M}^{t-1}| - |\mathcal{M}^t|$. If $b > 0$, the potential decreases, and its change can be bounded as

$$\Delta\Phi \leq H(|\mathcal{M}^t|) - H(|\mathcal{M}^{t-1}|) = -\frac{1}{|\mathcal{M}^t|+1} - \frac{1}{|\mathcal{M}^t|+2} - \cdots - \frac{1}{|\mathcal{M}^t|+b} \leq -b \cdot \frac{1}{|\mathcal{M}^{t-1}|}.$$

On the other hand, the expected cost of ALG is at most k times the probability that it has to move, which is exactly $P[Q_{t-1} \in \mathcal{M}^{t-1} \setminus \mathcal{M}^t] = b/|\mathcal{M}^{t-1}|$ using Lemma 3.4. Thus the expected cost of the algorithm is at most $k \cdot b/|\mathcal{M}^{t-1}|$, which is at most $k \cdot (-\Delta\Phi)$.

In the second case, we have $m_t < m_{t-1}$. By Lemma 3.3, we know that $|\mathcal{M}^t| \leq k!/m_t!$ and hence

$$\Delta\Phi = H(|\mathcal{M}^t|) - H(|\mathcal{M}^{t-1}|) - H(k!/m_t!) \leq -H(|\mathcal{M}^{t-1}|) \leq -1,$$

since $|\mathcal{M}^{t-1}| \geq 1$ and therefore $H(|\mathcal{M}^{t-1}|) \geq 1$. As the expected cost incurred by the algorithm is at most k, this is at most $k \cdot (-\Delta\Phi)$.

4 ALGORITHM FOR WEIGHTED UNIFORM METRICS

In this section we prove Theorem 1.3. Our algorithm is a natural extension of the algorithm of Fiat and Ricklin [19] for the weighted k-server problem on uniform metrics.

High-level idea. The algorithm is defined by a recursive construction based on the following idea. First, we can assume that the weights of the metric spaces are highly separated, i.e., $w_1 \ll w_2 \ll \ldots \ll w_k$ (if they are not we can make them separated while losing some additional factors). So in any reasonable solution, the server s_k lying in metric M_k should move much less often than the other servers. For that reason, the algorithm moves s_k only when the accumulated cost of the other $k-1$ servers reaches w_k. Choosing where to move s_k turns out to be a crucial decision. For that reason, (in each "level k-phase") during the first part of the request sequence when the algorithm only uses $k-1$ servers, it counts how many times each point of M_k is requested. We call this "learning subphase". Intuitively, points of M_k which are requested a lot are "good candidates" to place s_k. Now, during the next $c(k)$ (to be defined later) subphases, s_k visits the $c(k)$ most requested points. This way, it visits all "important locations" of M_k. A similar strategy is repeated recursively using $k-1$ servers within each subphase.

Notation and Preliminaries. We denote by s_i^{ALG} and s_i^{ADV} the server of the algorithm (resp. adversary) that lies in metric space M_i. Sometimes we drop the superscript and simply use s_i when the context is clear. We set $R_k := 2^{2^{k+2}}$ and $c(k) := 2^{2^{k+1}-3}$. Note that $c(1) = 2$ and that for all i,

$$4(c(i)+1)\cdot c(i) \leq 8c(i)^2 = c(i+1). \tag{1}$$

Moreover, for all $i \geq 2$, we have

$$R_i = 8 \cdot c(i) \cdot R_{i-1}. \tag{2}$$

We assume (by rounding the weights if necessary) that $w_1 = 1$ and that for $2 \leq i \leq k$, w_i is an integral multiple of $2(1+c(i-1))\cdot w_{i-1}$. Let m_i denote the ratio $w_i/(2(1+c(i-1))\cdot w_{i-1})$.

The rounding can increase the weight of each server at most by a factor of $4^{k-1}c(k-1)\cdot \ldots \cdot c(1) \leq R_{k-1}$. So, proving a competitive ratio R_k for an instance with rounded weights will imply a competitive ratio $R_k \cdot R_{k-1} < (R_k)^2$ for arbitrary weights.

Finally, we assume that in every request ALG needs to move a server. This is without loss of generality: requests served by the algorithm without moving a server do not affect its cost and can only increase the optimal cost. This assumption will play an important role in the algorithm below.

4.1 Algorithm Description

The algorithm is defined recursively, where ALG_i denotes the algorithm using servers $s_1, \ldots, s_i$. An execution of ALG_i is divided into phases. The phases are independent of each other and the overall algorithm is completely determined by describing how each phase works. We now describe the phases.

ALG_1 is very simple; given any request, ALG_1 moves the server to the requested point. For purposes of analysis, we divide the execution of ALG_1 into phases, where each phase consists of $2(c(1)+1) = 6$ requests.

Phase of ALG_1**:**

```
for j = 1 to 2(c(1) + 1) do
    Request arrives to point p: Move s_1 to p.
Terminate Phase
```

We now define a phase of ALG_i for $i \geq 2$. Each phase of ALG_i consists of exactly $c(i)+1$ subphases. The first subphase within a phase is special and we call it the *learning subphase*. During each subphase we execute ALG_{i-1} until the cost incurred is exactly w_i.

Phase of ALG_i**,** $i \geq 2$**:**

```
Move s_i to an arbitrary point of M_i;
Run ALG_{i-1} until cost incurred equals w_i ;             // Learning subphase
For p ∈ M_i, m(p) ← # of requests such that r(i) = p;     // Assume m(p_1) ≥ ... ≥ m(p_n)
P ← {p_1, ..., p_{c(i)}};
for j = 1 to c(i) do
    Move s_i to an arbitrary point p ∈ P;
    P ← P − p;
    Run ALG_{i-1} until cost incurred equals w_i ;         // (j + 1)th subphase
Terminate Phase
```

During the learning subphase, for each point $p \in M_i$, ALG_i maintains a count $m(p)$ of the number of requests r where p is requested in M_i, i.e. $r(i) = p$. Let us order the points of M_i as $p_1, \ldots, p_n$ such that $m(p_1) \geq \ldots \geq m(p_n)$ (ties are broken arbitrarily). We assume that $|M_i| \geq c(i)$ (if M_i has fewer points, we add some dummy points that are never requested). Let P be the set of $c(i)$ most requested points during the learning subphase, i.e. $P = \{p_1, \ldots, p_{c(i)}\}$.

For the rest of the phase ALG_i repeats the following $c(i)$ times: it moves s_i to a point $p \in P$ that it has not visited during this phase, and starts the next subphase (i.e. it calls ALG_{i-1} until its cost reaches w_i).

4.2 Analysis

We first note some basic properties that follow directly by the construction of the algorithm. Call a phase of ALG_i, $i \geq 2$ *complete*, if all its subphases are finished. Similarly, a phase of ALG_1 is complete if it served exactly 6 requests.

Observation 4.1. *For $i \geq 2$, a complete phase of ALG_i consists of $(c(i) + 1)$ subphases.*

Observation 4.2. *For $i \geq 2$, the cost incurred to serve all the requests of a subphase of ALG_i is w_i.*

These observations give the following corollary.

Corollary 4.3. *For $i \geq 1$, the cost incurred by ALG_i to serve requests of a phase is $2(c(i) + 1)w_i$.*

Proof. For $i = 1$ this holds by definition of the phase. For $i \geq 2$, a phase consists of $(c(i) + 1)$ subphases. Before each subphase ALG_i moves server s_i, which costs w_i, and moreover ALG_{i-1} also incurs cost w_i. □

Using this, we get the following two simple properties.

Lemma 4.4. *By definition of ALG, the following properties hold:*

(1) A subphase of ALG_i, $i \geq 2$, consists of m_i complete phases of ALG_{i-1}.
(2) All complete phases of ALG_i, $i \geq 1$, consist of the same number of requests.

Proof. The first property uses the rounding of the weights. By Corollary 4.3, each phase of ALG_{i-1} costs $2(c(i-1) + 1)w_{i-1}$ and, in each subphase of ALG_i, the cost incurred by ALG_{i-1} is w_i. So there are exactly $w_i/(2(c(i-1) + 1)w_{i-1}) = m_i$ phases of ALG_{i-1}.

The property above, combined with Observation 4.1 implies that a complete phase of ALG_i contains $m_i \cdot (c(i) + 1)$ complete phases ALG_{i-1}. Now, the second property follows directly by induction: each phase of ALG_1 consists of $2(c(1) + 1) = 6$ requests, and each phase of ALG_i consists of $m_i(c(i) + 1)$ phases of ALG_{i-1}. □

Consider a phase of ALG_i. The next lemma shows that, for any point $p \in M_i$, there exists a subphase where it is not requested too many times. This crucially uses the assumption that ALG_i has to move a server in every request.

Lemma 4.5. *Consider a complete phase of ALG_i, $i \geq 2$. For any point $p \in M_i$, there exists a subphase such that at most $1/c(i)$ fraction of the requests have $r(i) = p$.*

Proof. Let P be the set of $c(i)$ most requested points of M_i during the learning subphase. We consider two cases: if $p \in P$, there exists a subphase where s_i^{ALG} is located at p. During this subphase there are no requests such that $r(i) = p$, by our assumption that the algorithm moves some server at every request. Otherwise, if $p \notin P$, then during the learning subphase, the fraction of requests such that $r(i) = p$ is no more than $1/c(i)$. □

To prove the competitiveness of ALG_k with respect to the optimal offline solution ADV_k, the proof uses a subtle induction on k. Clearly, one cannot compare ALG_i, for $i < k$ against ADV_k, since the latter has more servers and its cost could be arbitrarily lower. So the idea is to compare ALG_i against ADV_i, an adversary with servers $s_1, \ldots, s_i$, while ensuring that ADV_i is an accurate estimate of ADV_k during time intervals when ALG_i is called by ALG_k. To achieve this, the inductive hypothesis is required to satisfy certain properties described below. For a fixed phase, let $\mathrm{cost}(\mathrm{ALG}_i)$ and $\mathrm{cost}(\mathrm{ADV}_i)$ denote the cost of ALG_i and ADV_i respectively.

(i) **Initial Configuration of** ADV_i**.** Algorithm ALG_i (for $i < k$), is called several times during a phase of ALG_k. As we don't know the current configuration of ADV_i each time ALG_i is called, we require that for every complete phase, $\mathrm{cost}(\mathrm{ALG}_i) \le R_i \cdot \mathrm{cost}(\mathrm{ADV}_i)$, for any initial configuration of ADV_i.

(ii) **Adversary can ignore a fraction of requests.** During a phase of ALG_i, ADV_k may serve requests with servers $s_{i+1}, \ldots, s_k$, and hence the competitive ratio of ALG_i against ADV_i may not give any meaningful guarantee. To get around this, we will require that $\mathrm{cost}(\mathrm{ALG}_i) \le R_i \cdot \mathrm{cost}(\mathrm{ADV}_i)$, even if the ADV_i ignores an $f(i) := 4/c(i+1)$ fraction of requests. This will allow us to use the inductive hypothesis for the phases of ALG_i where ADV_k uses servers $s_{i+1}, \ldots, s_k$ to serve at most $f(i)$ fraction of requests.

For a fixed phase, we say that ALG_i is strictly R_i-competitive against ADV_i, if $\mathrm{cost}(\mathrm{ALG}_i) \le R_i \cdot \mathrm{cost}(\mathrm{ADV}_i)$. The key result is the following.

Theorem 4.6. *Consider a complete phase of* ALG_i. *Let* ADV_i *be an adversary with i servers that is allowed to choose any initial configuration and to ignore any* $4/c(i+1)$ *fraction of requests. Then,* ALG_i *is strictly* R_i*-competitive against* ADV_i.

Before proving this, let us note that this directly implies Theorem 1.3. Indeed, for any request sequence σ, all phases except possibly the last one, are complete, so $\mathrm{cost}(\mathrm{ALG}_k) \le R_k \cdot \mathrm{cost}(\mathrm{ADV}_k)$. The cost of ALG_k for the last phase, is at most $2(c(k)+1)w_k$, which is a fixed additive term independent of the length of σ. So, $\mathrm{ALG}_k(\sigma) \le R_k \cdot \mathrm{ADV}_k(\sigma) + 2(c(k)+1)w_k$, and ALG_k is R_k-competitive. Together with loss in rounding the weights, this gives a competitive ratio of at mot $(R_k)^2 \le 2^{2^{k+3}}$ for arbitrary weights.

Proof of Theorem 4.6. We prove the theorem by induction on k.

Base case ($i = 1$): As $R_1 > 6$ and $4/c(2) = 1/8 \le 1/3$, it suffices to show here that ALG_1 is strictly 6-competitive in a phase where ADV_1 can ignore at most $1/3$ fraction of requests, for any starting point of $s_1^{\mathrm{ADV}_1}$.

By Corollary 4.3, we have $\mathrm{cost}(\mathrm{ALG}_1) = 2(c(1)+1) = 6$. We show that $\mathrm{cost}(\mathrm{ADV}_1) \ge 1$. Consider two consecutive requests r_{t-1}, r_t. By our assumption that ALG_1 has to move its server in every request, it must be that $r_{t-1} \ne r_t$. So, for any t if ADV_1 does not ignore both r_{t-1} and r_t, then it must pay 1 to serve r_t. Moreover, as the adverary can chose the initial server location, it may (only) serve the first request at zero cost. As a phase consists of 6 requests, ADV_i can ignore at most $6/3 = 2$ of them, so there are at most 4 requests that are either ignored or appear immediately after an ignored request. So among requests $r_2, \ldots, r_6$, there is at least one request r_t, such that both r_{t-1} and r_t are not ignored.

Inductive step: Assume inductively that ALG_{i-1} is strictly R_{i-1}-competitive against any adversary with $i-1$ servers that can ignore up to $4/c(i)$ fraction of requests.

Let us consider some phase at level i, and let I denote the set of requests that ADV_i chooses to ignore during the phase. We will show that $\mathrm{cost}(\mathrm{ADV}_i) \ge w_i/(2R_{i-1})$. This implies the theorem, as $\text{cost}(\text{ALG}_i) = 2(c(i)+1)w_i$ by Corollary 4.3 and hence,

$$\frac{\text{cost}(\text{ALG}_i)}{\text{cost}(\text{ADV}_i)} \leq \frac{2(c(i)+1)w_i}{w_i/(2R_{i-1})} = 4(c(i)+1)R_{i-1} \leq 8 \cdot c(i) \cdot R_{i-1} = R_i.$$

First, if ADV_i moves server s_i during the phase, its cost is already at least w_i and hence more than $w_i/(2R_{i-1})$. So we can assume that s_i^{ADV} stays fixed at some point $p \in M_i$ during the entire phase. So, ADV_i is an adversary that uses $i-1$ servers and can ignore all requests with $r(i) = p$ and the requests of I. We will show that there is a subphase where $\text{cost}(\text{ADV}_i) \geq w_i/(2R_{i-1})$.

By Lemma 4.5, there exists a subphase, call it j, such that at most $1/c(i)$ fraction of the requests have $r(i) = p$. As all $c(i)+1$ subphases have the same number of requests (by Lemma 4.4), even if all the requests of I belong to subphase j, they make up at most $(4 \cdot (c(i)+1))/c(i+1) \leq 1/c(i)$ fraction of its requests, where the inequality follows from equation (1). So overall during subphase j, ADV_i uses servers $s_1, \dots, s_{i-1}$ and ignores at most $2/c(i)$ fraction of requests.

We now apply the inductive hypothesis together with an averaging argument. As subphase j consists of m_i phases of ALG_{i-1}, all of equal length, and ADV_i ignores at most $2/c(i)$ fraction of requests of the subphase, there are at most $m_i/2$ phases of ALG_{i-1} where it can ignore more than $4/c(i)$ fraction of requests. So, for at least $m_i/2$ phases of ALG_{i-1}, ADV_i uses $i-1$ servers and ignores no more than $4/c(i)$ fraction of requests. By the inductive hypothesis, ALG_{i-1} is strictly R_{i-1}-competitive against ADV_i in these phases. As the cost of ALG_{i-1} for each phase is the same (by Corollary 4.3), overall ALG_i is strictly $2R_{i-1}$ competitive during subphase j. As the cost of ALG_i during subphase j is w_i, we get that $\text{cost}(\text{ADV}_i) \geq w_i/2R_{i-1}$, as claimed. □

5 CONCLUDING REMARKS

We gave the first $f(k)$-competitive algorithms for uniform metrics, which attain (almost) optimal competitive ratios. The outstanding open problem is the following:

Open Problem 5.1. *Is there an $f(k)$-competitive algorithm for the generalized k-server problem in general metric spaces?*

Answering this question seems to require the development of powerful new techniques for online algorithms and could lead to a much deeper theory of online computation.

Even for the special case of the weighted k-server problem, no competitive algorithms are known for $k > 2$ beyond uniform metrics. As an intermediate step between uniform and arbitrary metric spaces, it would be interesting to focus on some fixed metric which is more complex than uniform metrics (e.g. weighted star, line). Understanding the easier weighted k-server could be a useful first step in understanding generalized k-server, as was the case in Section 4.

Randomized Algorithms. Can randomization help, even for the case of weighted uniform metrics? Recently, Ayyadevara and Chiplunkar [3] showed that the competitive ratio of randomized algorithms is at least exponential. However, even for easier the weighted k-server problem on uniform metrics, no $2^{2^{o(k)}}$ bound is known. This motivates the following question.

Open Problem 5.2. *Is there an* $\exp(k)$ *competitive randomized algorithm for weighted uniform metrics?*

ACKNOWLEDGMENTS

We would like to thank René Sitters for useful discussions on the generalized k-server problem.

REFERENCES

[1] Dimitris Achlioptas, Marek Chrobak, and John Noga. 2000. Competitive analysis of randomized paging algorithms. *Theor. Comput. Sci.* 234, 1-2 (2000), 203–218. https://doi.org/10.1016/S0304-3975(98)00116-9

[2] John Augustine and Nick Gravin. 2010. On the Continuous CNN Problem. In *ISAAC*. 254–265.

[3] Nikhil Ayyadevara and Ashish Chiplunkar. 2021. The Randomized Competitive Ratio of Weighted k-Server Is at Least Exponential. In *European Symposium on Algorithms, ESA (LIPIcs, Vol. 204)*, Petra Mutzel, Rasmus Pagh, and Grzegorz Herman (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 9:1–9:11. https://doi.org/10.4230/LIPIcs.ESA.2021.9

[4] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. 2015. A Polylogarithmic-Competitive Algorithm for the k-Server Problem. *J. ACM* 62, 5 (2015), 40.

[5] Nikhil Bansal, Marek Eliáš, and Grigorios Koumoutsos. 2017. Weighted k-Server Bounds via Combinatorial Dichotomies. In *Symposium on Foundations of Computer Science, FOCS*. 493–504. https://doi.org/10.1109/FOCS.2017.52

[6] Nikhil Bansal, Marek Eliáš, Grigorios Koumoutsos, and Jesper Nederlof. 2018. Competitive Algorithms for Generalized k-Server in Uniform Metrics. In *Symposium on Discrete Algorithms, SODA*. 992–1001. https://doi.org/10.1137/1.9781611975031.64

[7] Marcin Bienkowski, Łukasz Jeż, and Paweł Schmidt. 2019. Slaying Hydrae: Improved Bounds for Generalized k-Server in Uniform Metrics. In *Symposium on Algorithms and Computation, ISAAC 2019*. 14:1–14:14.

[8] Allan Borodin and Ran El-Yaniv. 1998. *Online computation and competitive analysis*. Cambridge University Press.

[9] Allan Borodin, Nathan Linial, and Michael E. Saks. 1992. An Optimal On-Line Algorithm for Metrical Task System. *J. ACM* 39, 4 (1992), 745–763. https://doi.org/10.1145/146585.146588

[10] Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. 2018. k-server via multiscale entropic regularization. In *Symposium on Theory of Computing, STOC*. 3–16. https://doi.org/10.1145/3188745.3188798

[11] Ashish Chiplunkar. Oct 2016. Personal Communication.

[12] Ashish Chiplunkar and Sundar Vishwanathan. 2020. Randomized Memoryless Algorithms for the Weighted and the Generalized k-server Problems. *ACM Trans. Algorithms* 16, 1 (2020), 14:1–14:28. https://doi.org/10.1145/3365002

[13] Marek Chrobak. 2003. SIGACT news online algorithms column 1. *SIGACT News* 34, 4 (2003), 68–77. https://doi.org/10.1145/954092.954104

[14] Marek Chrobak, Howard J. Karloff, Thomas H. Payne, and Sundar Vishwanathan. 1991. New Results on Server Problems. *SIAM J. Discrete Math.* 4, 2 (1991), 172–181.

[15] Marek Chrobak and Lawrence L. Larmore. 1991. An Optimal On-Line Algorithm for k-Servers on Trees. *SIAM J. Comput.* 20, 1 (1991), 144–148. https://doi.org/10.1137/0220008

[16] Z. Dvir. 2009. On the size of Kakeya sets in finite fields. *J. Amer. Math. Soc.* 22 (2009), 1093–1097.

[17] J. S. Ellenberg and D. Gijswijt. 2016. On large subsets of F_q^n with no three-term arithmetic progression. *ArXiv e-prints* arXiv:1605.09223 (2016). arXiv:1605.09223

[18] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. 1991. Competitive Paging Algorithms. *J. Algorithms* 12, 4 (1991), 685–699. https://doi.org/10.1016/0196-6774(91)90041-V

[19] Amos Fiat and Moty Ricklin. 1994. Competitive Algorithms for the Weighted Server Problem. *Theor. Comput. Sci.* 130, 1 (1994), 85–99. https://doi.org/10.1016/0304-3975(94)90154-6

[20] L. Guth. 2016. *Polynomial Methods in Combinatorics*. American Mathematical Society.

[21] Kazuo Iwama and Kouki Yonezawa. 2001. Axis-bound CNN problem. *IEICE TRANS* (2001), 1–8.

[22] Kazuo Iwama and Kouki Yonezawa. 2004. The orthogonal CNN problem. *Inf. Process. Lett.* 90, 3 (2004), 115–120.

[23] Bart M. P. Jansen and Jesper Nederlof. 2019. Computing the chromatic number using graph decompositions via matrix rank. *Theor. Comput. Sci.* 795 (2019), 520–539. https://doi.org/10.1016/j.tcs.2019.08.006

[24] Stasys Jukna. 2011. *Extremal Combinatorics - With Applications in Computer Science*. Springer.

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

[26] Elias Koutsoupias and Christos H. Papadimitriou. 1996. The 2-Evader Problem. *Inf. Process. Lett.* 57, 5 (1996), 249–252. https://doi.org/10.1016/0020-0190(96)00010-5

[27] Elias Koutsoupias and David Scot Taylor. 2004. The CNN problem and other k-server variants. *Theor. Comput. Sci.* 324, 2-3 (2004), 347–359. https://doi.org/10.1016/j.tcs.2004.06.002

[28] Mark S. Manasse, Lyle A. McGeoch, and Daniel D. Sleator. 1990. Competitive algorithms for server problems. *J. ACM* 11, 2 (1990), 208–230.

[29] Jiří Matoušek. 2010. *Thirty-three Miniatures: Mathematical and Algorithmic Applications of Linear Algebra*. American Mathematical Society.

[30] Lyle A. McGeoch and Daniel Dominic Sleator. 1991. A Strongly Competitive Randomized Paging Algorithm. *Algorithmica* 6, 6 (1991), 816–825. https://doi.org/10.1007/BF01759073

[31] René Sitters. 2014. The Generalized Work Function Algorithm Is Competitive for the Generalized 2-Server Problem. *SIAM J. Comput.* 43, 1 (2014), 96–125. https://doi.org/10.1137/120885309

[32] René Sitters, Leen Stougie, and Willem de Paepe. 2003. A Competitive Algorithm for the General 2-Server Problem. In *ICALP*. 624–636.

[33] René A. Sitters and Leen Stougie. 2006. The generalized two-server problem. *J. ACM* 53, 3 (2006), 437–458. https://doi.org/10.1145/1147954.1147960

[34] Daniel Dominic Sleator and Robert Endre Tarjan. 1985. Amortized Efficiency of List Update and Paging Rules. *Commun. ACM* 28, 2 (1985), 202–208. https://doi.org/10.1145/2786.2793

A LOWER BOUND

We present a simple lower bound on the competitive ratio of deterministic algorithms for the generalized k-server problem in uniform metrics. In particular, we show a simple construction due to [27] that directly implies a $(2^k - 1)/k$ lower bound on the competitive ratio of deterministic algorithms. Using a more careful argument, [27] also improve this to $2^k - 1$.

Assume that each metric space M_i has $n = 2$ points, labeled by 0,1. A configuration of servers is a vector $c \in \{0, 1\}^k$, so there are 2^k possible configurations. Now, a request $r = (r_1, \ldots, r_k)$ is unsatisfied if and only if the algorithm is in the antipodal configuration $\bar{r} = (1 - r_1, \ldots, 1 - r_k)$. Let ALG be any online algorithm and ADV be the adversary. Initially, ALG and ADV are in the same configuration. At each time step, if the current configuration of ALG is $a = (a_1, \ldots, a_k)$, the adversary requests $\bar{a}$ until ALG visits every configuration. If p is the configuration that ALG visits last, the adversary can simply move to p at the beginning, paying at most k, and satisfy all requests until ALG moves to p. On the other hand, ALG pays at least $2^k - 1$ until it reaches p. Once ALG and ADV are in the same configuration, the strategy repeats.