58th Annual IEEE Symposium on Foundations of Computer Science

Weighted k-Server Bounds via Combinatorial Dichotomies

Nikhil Bansal*, Marek Eliáš*, Grigorios Koumoutsos*
TU Eindhoven, The Netherlands
{*n.bansal,m.elias,g.koumoutsos*}*@tue.nl*

***Abstract*—The weighted k-server problem is a natural generalization of the k-server problem where each server has a different weight. We consider the problem on uniform metrics, which corresponds to a natural generalization of paging. Our main result is a doubly exponential lower bound on the competitive ratio of any deterministic online algorithm, that essentially matches the known upper bounds for the problem and closes a large and long-standing gap.**

The lower bound is based on relating the weighted k-server problem to a certain combinatorial problem and proving a Ramsey-theoretic lower bound for it. This combinatorial connection also reveals several structural properties of low cost feasible solutions to serve a sequence of requests. We use this to show that the generalized Work Function Algorithm achieves an almost optimum competitive ratio, and to obtain new refined upper bounds on the competitive ratio for the case of d different weight classes.

***Keywords*-weighted k-server; online algorithms; competitive analysis;**

I. Introduction

The k-server problem is one of the most natural and fundamental online problems and its study has been quite influential in the development of competitive analysis (see e.g. [1]–[5]). The problem is almost settled in the deterministic case: no algorithm can be better than k-competitive in any metric space of more than k points [6], and in their breakthrough result, Koutsoupias and Papadimitriou [2] showed that the Work Function Algorithm (WFA) is $(2k-1)$-competitive in any metric space. Tight k-competitive algorithms are also known for several special metrics [7]–[10].

Despite this progress, several natural variants and generalizations of the k-server problem are very poorly understood. In particular, they exhibit very different and intriguing behavior and the techniques for the standard k-server problem do not seem to apply to them (we describe some of these problems and results in Section I-B). Getting a better understanding of such problems is a natural step towards building a deeper theory of online computation.

Weighted k-server: Perhaps the simplest such problem is the weighted k-server problem on uniform metrics, that was first introduced and studied by Fiat and Ricklin [11]. Here, there are k servers located at points of a uniform metric space. In each step a request arrives at some point and must be served by moving some server there. Each server s_i has a positive weight w_i and it costs w_i to move s_i to another point. The goal is to minimize the total cost for serving the requests.

Note that in the unweighted case where each $w_i = 1$, this is the classic and extensively studied paging/caching problem [7], for which several tight k-competitive deterministic and $O(\log k)$-competitive randomized algorithms are known [1]. Indeed, one of the motivations of [11] for studying the weighted k-server problem was that it corresponds to paging where each memory slot has a different replacement cost.[1]

Throughout this paper, we only consider the uniform metric, and by weighted k-server we always mean the problem on the uniform metric, unless stated otherwise.

Previous Bounds: There is a surprisingly huge gap between the known upper and lower bounds on the competitive ratio for weighted k-server. In their seminal paper, Fiat and Ricklin [11] gave the first deterministic algorithm with a doubly exponential competitive ratio of about $2^{4^k} = 2^{2^{2k}}$. They also showed a (singly) exponential lower bound of $(k+1)!/2$ on the competitive ratio of deterministic algorithms, which can be improved to $(k+1)!-1$ by a more careful argument [14].

More recently, Chiplunkar and Vishwanathan [14] considered a simple memoryless randomized algorithm, where server s_i moves to the requested point with some fixed probability p_i. They showed that there is always a choice of p_i as function of the weights, for which this gives an $\alpha_k < 1.6^{2^k}$-competitive algorithm against adaptive online adversaries. Note that $\alpha_k \in [2^{2^{k-1}}, 2^{2^k}]$. They also showed that this ratio is tight for such randomized memoryless algorithms. By the simulation technique of Ben-David et al. [15] that relates different adversary models, this gives an implicit $\alpha_k^2 \leq 2^{2^{k+1}}$-competitive deterministic algorithm[2].

Conjectured upper bound: Prior to our work, it was widely believed that the right competitive ratio should be $(k+1)!-1$. In fact, [14] mention that WFA is a natural candidate to achieve this.

*Supported by NWO Vidi grant 639.022.211 and ERC consolidator grant 617951. The full version of this paper can be found at: https://arxiv.org/abs/1704.03318.

[1]We crucially note that this problem should not be confused by the related, but very different, weighted paging problem where the weights are on the pages instead of the servers. Weighted paging corresponds to (unweighted) k-server on weighted star metrics and is very well understood. In particular, tight k-competitive deterministic and $O(\log k)$-competitive randomized algorithms are known [8], [12], [13].

[2]A more careful analysis shows that the Fiat-Ricklin algorithm [11] is also $2^{2^{k+O(1)}}$ competitive [16].

0272-5428/17 $31.00 © 2017 IEEE
DOI 10.1109/FOCS.2017.52

There are several compelling reasons for believing this. First, for classical k-server, the lower bound of k is achieved in metric spaces with $n = k + 1$ points, where each request is at the (unique) point with no online server. The $(k + 1)! - 1$ lower bound for weighted k-server [11], [14] also uses $n = k + 1$ points. More importantly, this is in fact the right bound for $n = k + 1$. This follows as the weighted k-server problem on n points is a Metrical Service System (MSS)[3] with $N = \binom{n}{k}k!$ states, which correspond to the k-tuples describing the configuration of the servers. It is known that WFA is $(N - 1)$-competitive for any MSS with N states [18]. As $N = (k+1)!$ for $n = k+1$, this gives the $(k+1)!-1$ upper bound. Moreover, Chrobak and Sgall [19] showed that WFA is exactly $(k + 1)! - 1 = 3! - 1 = 5$-competitive for $k = 2$ (with arbitrary n), providing strong coincidental evidence for the $(k+1)!-1$ bound for general k.

A. Our Results

In this paper, we study the weighted k-server problem systematically and obtain several new results. A key idea is to relate online weighted k-server to a natural *offline* combinatorial question about the structure of all possible "feasible labelings" for a hierarchical collection of intervals of depth k. In particular, we show that the competitive ratio for weighted k-server is closely related to a certain Ramsey-theoretic parameter of this combinatorial problem. This parameter, let us call it $f(k)$ for the discussion here, reflects the amount of uncertainty that adversary can create about the truly good solutions in an instance. This connection is used for both upper and lower bound results in this paper.

Lower Bounds: Somewhat surprisingly, we show that the doubly exponential upper bounds [11], [14] for the problem are essentially the best possible (up to lower order terms in the exponent).

Theorem I.1. *Any deterministic algorithm for the weighted k-server problem on uniform metrics has a competitive ratio of at least $\Omega(2^{2^{k-4}})$.*

As usual, we prove Theorem I.1 by designing an adversarial strategy to produce an online request sequence dynamically (depending on the actions of the online algorithm), so that (i) the online algorithm incurs a high cost, while (ii) the adversary can always guarantee some low cost offline solution in hindsight. Our strategy is based on a recursive construction on $n \geq \exp(\exp(k))$ points (necessarily so, by the connection to MSS) and it is designed in a modular way using the combinatorial connection as follows: First, we construct a recursive lower bound instance for the combinatorial problem for which the Ramsey-theoretic parameter $f(k) \geq 2^{2^{k-4}}$. Second, to obtain the online lower bound, we embed this construction into a recursive strategy to dynamically generate an adversarial request sequence with the properties described above.

Moreover, we show (in the full version) that the lower bound from Theorem I.1, can be extended to general metric spaces. That means, in any metric space containing enough points, the competitive ratio of deterministic algorithms for weighted k-server is at least $\Omega(2^{2^{k-4}})$.

Upper Bounds: The combinatorial connection is also very useful for positive results. We first show that the generalized WFA, a very generic online algorithm that is applicable to a wide variety of problems, is essentially optimum.

Theorem I.2. *The generalized* WFA *is $2^{2^{k+O(\log k)}}$-competitive for weighted k-server on uniform metrics.*

While previous algorithms [11], [14] were also essentially optimum, this result is interesting as the generalized WFA is a generic algorithm and is not specifically designed for this problem at hand. In fact, as we discuss in Section I-B, for more general variants of k-server the generalized WFA is the only known candidate algorithm that can be competitive.

To show Theorem I.2, we first prove an almost matching upper bound of $f(k) \leq 2^{2^{k+3\log k}}$ for the combinatorial problem. As will be clear later, we call such results *dichotomy theorems*. Then, we relate the combinatorial problem to the dynamics of *work functions* and use the dichotomy theorem recursively to bound the cost of the WFA on any instance.

This approach also allows us to extend and refine these results to the setting of d different weight classes with $k_1, \ldots, k_d$ servers of each class. This corresponds to d-level caching where each cache has replacement cost w_i and capacity k_i. As practical applications usually have few weight classes, the case where d is a small constant independent of k is of interest. Previously, [11] gave an improved $k^{O(k)}$ bound for $d = 2$, but a major difficulty in extending their result is that their algorithm is phase-based and gets substantially more complicated for $d > 2$.

Theorem I.3. *The competitive ratio of the generalized* WFA *for the weighted k-server problem on uniform metrics with d different weights is at most $2^{O(d)\,k^3\prod_{i=1}^{d}(k_i+1)}$, where k_i is the number of servers of weight w_i, and $k = \sum_{i=1}^{d} k_i$.*

For k distinct weights, i.e $k_i = 1$ for each i, note that this matches the $2^{\text{poly}(k)\cdot 2^k}$ bound in Theorem I.2. For d weight classes, this gives $2^{O(dk^{d+3})}$, which is singly exponential in k for $d = O(1)$. To prove Theorem I.3, we proceed as before. We first prove a more refined dichotomy theorem (Theorem III.9) and use it recursively with the WFA.

B. Generalizations of k-server and Related Work

The weighted k-server problem on uniform metrics that we consider here is the simplest among the several generalizations of k-server that are very poorly understood. An

[3]This is a Metrical Task System [17] where the cost in each state is either 0 or infinite (called *forcing task systems* in [6]).

immediate generalization is the weighted k-server problem in general metrics. This seems very intriguing even for a line metric. Koutsoupias and Taylor [20] showed that natural generalizations of many successful k-server algorithms are not competitive. Chrobak and Sgall [19] showed that any memoryless randomized algorithm has unbounded competitive ratio. In fact, the only candidate competitive algorithm for the line seems to be the generalized WFA. There are also other qualitative differences. While the standard k-server problem is believed to have the same competitive ratio in every metric, this is not the case for weighted k-server. For $k = 2$ in a line, [20] showed that any deterministic algorithm is at least 10.12-competitive, while on uniform metrics the competitive ratio is 5 [19].

A far-reaching generalization of the weighted k-server problem is the *generalized* k-server problem [4], [20]–[22], with various applications. Here, there are k metric spaces $M_1, \ldots, M_k$, and each server s_i moves in its own space M_i. A request r_t at time t is specified by a k-tuple $r_t = (r_t(1), \ldots, r_t(k))$ and must be served by moving server s_i to $r_t(i)$ for some $i \in [k]$. Note that the usual k-server corresponds to very special case where the metrics M_i are identical and each request, $r_t = (\sigma_t, \sigma_t, \ldots, \sigma_t)$, has all coordinates identical. Weighted k-server (in a general metric M) is also a very special case where each $M_i = w_i \cdot M$ and $r_t = (\sigma_t, \sigma_t, \ldots, \sigma_t)$.

In a breakthrough result, Sitters and Stougie [21] gave an $O(1)$-competitive algorithm for the generalized 2-server problem. Recently, Sitters [4] showed that the generalized WFA is also $O(1)$-competitive for $k = 2$. Finding any competitive algorithm for $k > 2$ is a major open problem, even in very restricted cases. For example, the special case where each M_i is a line, also called the CNN problem, has received a lot of attention ([20], [23]–[26]), but even here no competitive algorithm is known for $k > 2$.

C. Notation and Preliminaries

We now give some necessary notation and basic concepts, that will be crucial for the technical overview of our results and techniques in Section II.

Problem definition: Let $M = (U, d)$ be a uniform metric space, where $U = \{1, \ldots, n\}$ is the set of points (we sometimes call them pages) and $d : U^2 \to \mathcal{R}$ is the distance function which satisfies $d(p, q) = 1$ for $p \neq q$, and $d(p, p) = 0$. There are k servers $s_1, \ldots, s_k$ with weights $w_1 \leq w_2 \leq \ldots \leq w_k$ located at points of M. The cost of moving server s_i from the point p to q is $w_i \cdot d(p, q) = w_i$. The input is a request sequence $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_T$, where $\sigma_t \in U$ is the point requested at time t. At each time t, an online algorithm needs to have a server at σ_t, without the knowledge of future requests. The goal is to minimize the total cost for serving σ.

We think of n and T as arbitrarily large compared to k. Note that if the weights are equal or similar, we can use the results for the (unweighted) k-server problem with no or small loss, so $w_{\max}/w_{\min}$ should be thought of as arbitrarily large. Also, if two weights are similar, we can treat them as same without much loss, and so in general it is useful to think of the weights as well-separated, i.e. $w_i \gg w_{i-1}$ for each i.

Work Functions and the Work Function Algorithm: We call a map $C \colon \{1, \ldots, k\} \to U$, specifying that server s_i is at point $C(i)$, a *configuration* C. Given a request sequence $\sigma = \sigma_1, \ldots, \sigma_t$, let $\mathrm{WF}_t(C)$ denote the optimal cost to serve requests $\sigma_1, \ldots, \sigma_t$ and end up in configuration C. The function WF_t is called *work function* at time t. Note that if the request sequence would terminate at time t, then $\min_C \mathrm{WF}_t(C)$ would be the offline optimum cost.

The Work Function Algorithm (WFA) works as follows: Let C_{t-1} denote its configuration at time $t - 1$. Then upon the request σ_t, WFA moves to the configuration C that minimizes $\mathrm{WF}_t(C) + d(C, C_{t-1})$. Roughly, WFA tries to mimic the offline optimum while also controlling its movement costs. For more background on WFA, see [1], [2], [18].

The generalized Work Function Algorithm (WFA_λ) is parameterized by a constant $\lambda \in (0, 1]$, and at time t moves to the configuration $C_t = \arg\min_C \mathrm{WF}_t(C) + \lambda d(C, C_{t-1})$. For more on WFA_λ, see [4].

Service Patterns and Feasible Labelings: We can view any solution to the weighted k-server problem as an interval covering in a natural way. For each server s_i we define a set of intervals $\mathcal{I}_i$ which captures the movements of s_i as follows: Let $t_1 < t_2 < t_3 < \cdots$ be the times when s_i moves. For each move at time t_j we have an interval $[t_{j-1}, t_j) \in \mathcal{I}_i$, which means that s_i stayed at the same location during this time period. We assume that $t_0 = 0$ and also add a final interval $[t_{\text{last}}, T + 1)$, where t_{last} is the last time when server s_i moved. So if s_i does not move at all, $\mathcal{I}_i$ contains the single interval $[0, T + 1)$. This gives a natural bijection between the moves of s_i and the intervals in $\mathcal{I}_i$, and the cost of the solution equals to $\sum_{i=1}^{k} w_i(|\mathcal{I}_i| - 1)$. We call $\mathcal{I}_i$ the ith level of intervals, and an interval in $I \in \mathcal{I}_i$ a level-i interval, or simply an ith level interval.

Definition I.4 (Service Pattern)**.** *We call the collection $\mathcal{I} = \mathcal{I}_1 \cup \cdots \cup \mathcal{I}_k$ a service pattern if each $\mathcal{I}_i$ is a partition of $[0, T + 1)$ into half-open intervals.*

Figure 1 contains an example of such service pattern.

To describe a solution for a weighted k-server instance completely, we label each interval $I \in \mathcal{I}_i$ with a point where s_i was located during time period I. We can also decide to give no label to some interval (which means that we don't care on which point the server is located). We call this a *labeled* service pattern.

Definition I.5 (Feasible Labeling)**.** *Given a service pattern $\mathcal{I}$ and a request sequence σ, we say that a (partial) labeling*

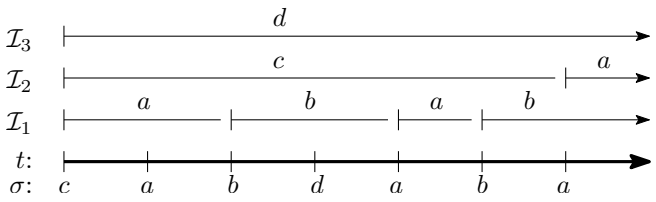

Figure 1. Illustration of a feasible service pattern for $k = 3$. Each interval in $\mathcal{I}_i$ defines a location for server s_i. At each time t, some interval covering t should be labeled by the requested point σ_t.

$\alpha\colon \mathcal{I} \to U$ *is* feasible *with respect to* σ*, if for each time* $t \geq 0$ *there exists an interval* $I \in \mathcal{I}$ *which contains* t *and* $\alpha(I) = \sigma_t$.

We call a service pattern $\mathcal{I}$ feasible with respect to σ if there is some labeling α of $\mathcal{I}$ that is feasible with respect to σ. Thus the offline weighted k-server problem for request sequence σ is equivalent to the problem of finding the cheapest feasible service pattern for σ.

Note that for a fixed service pattern $\mathcal{I}$, there may not exist any feasible labeling, or alternately there might exist many feasible labelings for it. Understanding the structure of the various possible feasible labelings for a given service pattern will play a major role in our results.

II. Overview

We now give an overview of the technical ideas and the organization of the paper.

Fix some request sequence σ. Suppose that the online algorithm knows the service pattern $\mathcal{I}$ of some optimal offline solution, but not the actual labels for the intervals in $\mathcal{I}$. Then intuitively, the online algorithm may still need to try out all possible candidate labels for an interval before figuring out the right one used by the offline solution. So, a key natural question turns out to be: How does the structure of all possible feasible labelings for $\mathcal{I}$ look like?

Let us consider this more closely. First, we can assume for simplicity that $\mathcal{I}$ has a tree structure (i.e whenever an interval at level i ends, all intervals at levels $1, \ldots, i-1$ end as well). Now, we can view $\mathcal{I}$ as a collection of disjoint trees on different parts of σ, that do not interact with each other. Focusing on some tree T with a root interval I (corresponding to the heaviest server s_k), it now suffices to understand what is the number of labels for I in all feasible labelings with respect to σ. This is because whenever we fix some label a for I, we get a similar question about the depth-$(k-1)$ subtrees of T on σ with a removed, and we can proceed recursively. This leads to the following problem.

The Combinatorial Problem: Given an interval tree T in a service pattern $\mathcal{I}$ on some request sequence σ. How many labels can the root interval I get over all possible feasible assignments to $\mathcal{I}$?

We will show the following dichotomy result for this problem: (i) Either any label in U works (i.e. the location of s_k does not matter), or (ii) there can be at most $f(k)$ feasible labels for I. This might be somewhat surprising as the tree T can be arbitrarily large and the number of its subtrees of depth $k-1$ may not be even a function of k.

We give two such dichotomy theorems in Section III-C. In Theorem III.8, we show $f(k) = O(\exp(\exp(k))$ for arbitrary weights, and in Theorem III.9, we give a more refined bound for the case with $k_1, \ldots, k_d$ servers of weights $w_1, \ldots, w_d$. These results are proved (in the full version) by induction but require some subtle technical details. In particular, we need a stronger inductive hypothesis, where we track all the feasible labels for the intervals on the path from the particular node towards the root. To this end, in Section III-A we describe some properties of these path labelings and their interactions at different nodes.

Upper Bounds: These dichotomy theorems are useful to upper-bound the competitive ratio as follows. Suppose that the online algorithm knows the optimum service pattern $\mathcal{I}$, but not the actual labels. Fix some tree $T \subseteq \mathcal{I}$ with root interval I. We know that the offline solution pays w_k to move the server s_k at the end of I, and let $\mathrm{cost}(k-1)$ denote the offline cost incurred during I due to the movement of the $k-1$ lighter servers. Then, intuitively, the online algorithm has only $f(k)$ reasonable locations[4] to try during I. Assuming recursively that its competitive ratio with $k-1$ servers is c_{k-1}, its cost on I should be at most

$$f(k)\cdot(w_k + c_{k-1}\cdot\mathrm{cost}(k-1)) \leq f(k)\cdot c_{k-1}(w_k + \mathrm{cost}(k-1)) = f(k)\cdot c_{k-1}\cdot \mathrm{OPT}(I),$$

which gives $c_k \leq f(k)c_{k-1}$, and hence $c_k \leq f(k)\cdots f(1)$.

Of course, the online algorithm does not know the offline service pattern $\mathcal{I}$, but we can remove this assumption by losing another factor $f(k)$. The idea is roughly the following. Consider some time period $[t_1, t_2]$, during which online incurs cost about $c_{k-1}\, w_k$ and decides to move its heaviest server at time t_2. We claim that there can be at most $f(k)$ locations for the heavy server where the offline solution would pay less than $w_k/(4f(k))$ during $[t_1, t_2]$. Indeed, suppose there were $m = f(k)+1$ such locations $p_1, \ldots, p_m$. Then, for each $j = 1, \ldots, m$, take the corresponding optimum service pattern $\mathcal{I}^j$ with s_k located at p_j throughout $[t_1, t_2]$, and consider a new pattern $\mathcal{I}'$ by taking the common refinement of $\mathcal{I}^1, \ldots, \mathcal{I}^m$ (where any interval in $\mathcal{I}^j$ is a union of consecutive intervals in $\mathcal{I}'$). The pattern $\mathcal{I}'$ is quite cheap, its cost is at most $m \cdot w_k/(4f(k)) \leq w_k/2$, and we know that its root interval I can have $m = f(k)+1$ different labels. However, the dichotomy theorem implies that any point is a feasible label for I, including the location of

[4]The situation in case (i) of the dichotomy theorem, where the location of s_k does not matter, is much easier as the online algorithm can keep s_k any location.

the algorithm's heaviest server. But in such case, algorithm would not pay more than $c_{k-1}\,\mathrm{cost}(\mathcal{I}')$, what leads to a contradiction.

We make this intuition precise in Section V using work functions. In particular, we use the idea above to show that during any request sequence when WFA_λ moves s_k about $f(k)$ times, any offline algorithm must pay $\Omega(w_k)$.

Lower bound: In a more surprising direction, we can also use the combinatorial problem to create a lower bound. In Section III, we give a recursive combinatorial construction of a request sequence σ and a service pattern $\mathcal{I}$ consisting of a single interval tree, such that the number of feasible labelings for its root can actually be about $r_k = 2^{2^k}$.

Then in Section IV, we use the underlying combinatorial structure of this construction to design an adversarial strategy that forces any online algorithm to have a doubly-exponential competitive ratio.

Our adversarial strategy reacts adaptively to the movements of the online algorithm ALG, enforcing the two key properties. First, the adversary never moves a server s_i, where $i < k$, unless ALG also moves some heavier server of weight at least w_{i+1}. Second, the adversary never moves the heaviest server s_k unless ALG already moved s_k to all r_k possible feasible locations. By choosing the weights of the servers well-separated, e.g. $w_{i+1} \geq r_k \cdot \sum_{j=1}^{i} w_j$ for each i, it is easy to see that the above two properties imply an $\Omega(r_k)$ lower bound on the competitive ratio.

III. Service Patterns

In this section, we study the structure of feasible labelings. A crucial notion for this will be *request lists* of intervals. We also define two Ramsey-theoretic parameters to describe the size of the request lists. In Subsection III-B, we present a combinatorial lower bound for these two parameters.

Hierarchical service patterns: We call a service pattern $\mathcal{I}$ hierarchical, if each interval I at level $i < k$ has a unique parent J at level $i+1$ such that $I \subseteq J$. An arbitrary service pattern $\mathcal{I}$ can be made hierarchical easily and at relatively small cost: whenever an interval at level $i > 1$ ends at time t, we also end all intervals at levels $j = 1, \ldots, i-1$. This operation adds at most $w_1 + \ldots w_{i-1} \leq k w_i$, for each interval of weight w_i, so the overall cost can increase by a factor at most k. In fact, if the weights are well-separated, the loss is even smaller.

Henceforth, by service pattern we will always mean hierarchical service patterns, which we view as a disjoint collection of trees. We adopt the usual terminology for trees. The ancestors of I are all intervals at higher levels containing I, and descendants of I are all intervals at lower levels which are contained in I. We denote $A(I)$ the set of the ancestors of I and T_I the subtree of intervals rooted at I (note that T_I includes I).

Composition of feasible labelings: In hierarchical service patterns, the labelings can be composed easily in modular way. Let σ_I be the request sequence during the time interval I, and σ_J during some sibling J of I. If α_I and α_J are two feasible labelings with respect to σ_I and σ_J respectively, and if they assign the same labels to the ancestors of I and J (i.e. $\alpha_I(A(I)) = \alpha_J(A(J))$), we can easily construct a single α which is feasible with respect to both σ_I and σ_J: Label the intervals in T_I according to α_I, intervals in T_J according to α_J and their ancestors according to either α_I or α_J.

A. Structure of the feasible labelings

Consider a fixed service pattern $\mathcal{I}$ and some request sequence σ. There is a natural inductive approach for understanding the structure of feasible labelings of $\mathcal{I}$. Consider an interval $I \in \mathcal{I}$ at level $\ell < k$, and the associated request sequence $\sigma(I)$. In any feasible labeling of $\mathcal{I}$, some requests in $\sigma(I)$ will be covered (served) by the labels for the intervals in T_I, while others (possibly none) will be covered by labels assigned to ancestors $A(I)$ of I. So, it is useful to understand how many different "label sets" can arise for $A(I)$ in all possible feasible labelings. This leads to the notion of request lists.

Request lists: Let I be an interval at level $\ell < k$. We call a set of pages $S \subseteq U$ with $|S| \leq k - \ell$, a *valid tuple* for I, if upon assigning S to ancestors of I (in any order) there is some labeling of T_I that is feasible for σ_I. Let $R(I)$ denote the collection of all valid tuples for I.

Note that if S is a valid tuple for I, then all its supersets of size up to $k - \ell$ are also valid. This makes the set of all valid tuples hard to work with, and so we only consider the inclusion-wise minimal tuples.

Definition III.1 (Request list of an interval)**.** *Let I be an interval at level $\ell < k$. The request list of I, denoted by $L(I)$, is the set of inclusion-wise minimal valid tuples.*

Remark. We call this a request list as we view I as requesting a tuple in $L(I)$ as "help" from its ancestors in $A(I)$, to feasibly cover $\sigma(I)$. It is possible that there is a labeling α of T_I that can already cover $\sigma(I)$ and hence I does not need any "help" from its ancestors $A(I)$. In this case $L(I) = \{\emptyset\}$ (or equivalently every subset $S \subset U$ of size $\leq k - \ell$ is a valid tuple).

Tuples of size 1 in a request list will play an important role, and we will call them singletons.

Example. Let $I \in \mathcal{I}$ be an interval at level 1, and $I_2, \ldots, I_k$ its ancestors at levels $2, \ldots, k$. If $P = \{p_1, \ldots, p_j\}$, where $j < k$, is the set of all pages requested in σ_I, then one feasible labeling α with respect to σ_I is to assign $\alpha(I_{i+1}) = p_i$ for $i = 1, \ldots, j$, and no label for any other $J \in \mathcal{I}$. So P is a feasible tuple. However, P is not inclusion-wise minimal, as $\{p_2, \ldots, p_j\}$ is also valid tuple: We can set $\alpha(I) = p_1$, $\alpha(I_i) = p_i$ for $i = 2, \ldots, j$ and no

label for other intervals. Similarly, $P \setminus \{p_i\}$ for $i = 2, \ldots, j$, are also valid and inclusion-wise minimal. So, we have

$$L(I) = \big\{P \setminus \{p_1\}, P \setminus \{p_2\}, \ldots, P \setminus \{p_j\}\big\}.$$

Computation of Request Lists: Given a service pattern $\mathcal{I}$ and request sequence σ, the request lists for each interval I can be computed inductively. For the base case of a leaf interval I we already saw that $L(I) = \big\{P \setminus \{p_1\}, P \setminus \{p_2\}, \ldots, P \setminus \{p_j\}\big\}$.

For a higher level interval I, we will take the request lists of the children of I and combine them suitably. To describe this, we introduce the notion of *joint request lists*. Let I be a level ℓ interval for $\ell > 1$, and let $C = \{J_1, \ldots, J_m\} \subseteq \mathcal{I}_{\ell-1}$ be the set of its child intervals. Note that m can be arbitrarily large (and need not be a function of k). We define the joint request list of the intervals in C as follows.

Definition III.2 (Joint request list)**.** *Let I be an interval at level $\ell > 1$ and C be the set of its children at level $\ell - 1$. The joint request list of C, denoted by $L(C)$, is the set of inclusion-wise minimal tuples S with $|S| \leq k - (\ell - 1)$ for which there is a labeling α that is feasible with respect to σ_I and $\alpha(\{I\} \cup A(I)) = S$.*

Let $R(C)$ denote collection of all valid tuples (not necessarily minimal) in the joint request list for C. We note the following simple observation.

Observation III.3. *A tuple S belongs to $R(C)$ if and only if S belongs to $R(J_i)$ for each $i = 1, \ldots, m$. This implies that $S \in L(C)$ whenever it is an inclusion-wise minimal tuple such that each $L(J_i)$ for $i \in [m]$ contains some tuple $S_i \subseteq S$.*

Proof: Consider the feasible labeling α with respect to σ_I, which certifies that $S \in R(C)$. This is also feasible with respect to each σ_{J_i} and certifies that $S \in R(J_i)$. Conversely, let α_i, for $i = 1, \ldots, m$, denote the labeling feasible with respect to σ_{J_i} which certifies that $S \in R(C)$. As $J_1, \ldots, J_m$ are siblings, the composed labeling α defined by $\alpha(J) = \alpha_i(J)$ if $J \in T_{J_i}$ and, say, $\alpha(J) = \alpha_1(J)$ if $J \in I \cup A(I)$ is feasible for σ_I. ∎

Creation of the joint request list can be also seen as a kind of a product operation. For example, if there are two siblings J_1 and J_2 whose request lists are disjoint and contain only singletons, then their joint request list $L(J_1, J_2)$ contains all pairs $\{p, q\}$ such that $\{p\} \in L(J_1)$ and $\{q\} \in L(J_2)$. By Observation III.3, all such pairs belong to $R(J_1, J_2)$ and they are inclusion-wise minimal. The number of pairs in $L(J_1, J_2)$ equals to $|L(J_1)| \cdot |L(J_2)|$. In general, if $L(J_1)$ and $L(J_2)$ are not disjoint or contain tuples of different sizes, the product operation becomes more complicated, and therefore we use the view from Observation III.3 throughout this paper.

Finally, having obtained $L(C)$, the request list $L(I)$ is obtained using the following observation.

Observation III.4. *A tuple S belongs to $R(I)$ if and only if $S \cup \{p\}$ belongs to $R(C)$ for some $p \in U$.*

Proof: If $S \cup \{p\} \in R(C)$, then we find a feasible labeling α for σ_I with $\alpha(I) = p$ and $\alpha(A(I)) = S$. Conversely, if $S \in R(I)$, then there must some feasible labeling α for σ_I with $\alpha(A(I)) = S$, and we simply take $p = \alpha(I)$. ∎

So $L(I)$ can be generated by taking $S \setminus \{p\}$ for each $p \in S$ and $S \in L(C)$, and eliminating all resulting tuples that are not inclusion-wise minimal.

Example. Consider the interval I having two children J_1 and J_2. We saw that if $L(J_1)$ and $L(J_2)$ are disjoint and both contain only singletons, their joint list $L(J_1, J_2)$ contains $|L(J_1)| \cdot |L(J_2)|$ pairs. Then, according to Observation III.4, $L(I)$ contains $|L(J_1)| + |L(J_2)|$ singletons.

This observation that composition of request lists with singletons give request lists with singletons will be useful in the lower bound below.

Sizes of request lists: Now we define the Ramsey-theoretic parameters of the service patterns. Let us denote $f(\ell, t)$ the maximum possible numbers of t-tuples in the request list $L(I)$ of any interval I at level ℓ from any service pattern $\mathcal{I}$. Similarly, we denote $n(\ell, t)$ the maximum possible number of t-tuples contained in a joint request list $L(C)$, where C are the children of some ℓth level interval I. The examples above show that $n(\ell, 2)$ can be of order $f^2(\ell, 1)$, and $f(\ell + 1, 1) \geq 2f(\ell, 1)$. In the following subsection we show that the $f(\ell, 1)$ and $n(\ell, 1)$ can grow doubly exponentially with ℓ.

B. Doubly-exponential growth of the size of request lists

In the following theorem we show a doubly exponential lower bound on $n(k, 1)$. In particular, we construct a request sequence σ, and a service pattern such that each level-ℓ interval has a request list of $\Omega(2^{2^{\ell-3}})$ singletons.

Theorem III.5. *The numbers $n(\ell, 1)$ and $f(\ell - 1, 1)$ grow doubly-exponentially with ℓ. More specifically, for level k intervals we have*

$$n(k, 1) \geq f(k - 1, 1) \geq 2^{2^{k-4}}.$$

Proof: We construct a request sequence and a hierarchical service pattern with a single kth level interval, such that any interval I at level $1 \leq \ell < k$ has a request list $L(I)$ consisting of $n_{\ell+1}$ singletons, where $n_2 = 2$ and

$$n_{i+1} = (\lfloor n_i/2 \rfloor + 1) + (\lfloor n_i/2 \rfloor + 1)\lceil n_i/2 \rceil \geq (n_i/2)^2.$$

Note that $n_2 = 2, n_3 = 4, n_4 = 9, \ldots$ and in general as $n_{i+1} \geq (n_i/2)^2$ it follows that for $\ell \geq 4$, we have $n_\ell \geq 2^{2^{\ell-4}+2} \geq 2^{2^{\ell-4}}$.

We describe our construction inductively, starting at the first level. Let I be an interval at level 1 and σ_I be a

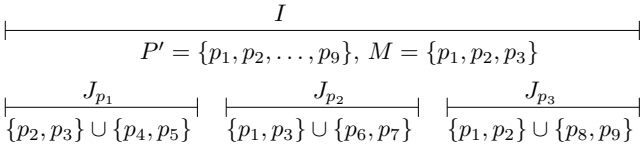

Figure 2. Construction of an interval at level $i = 3$ with request list of $n_4 = 9$ singletons, using intervals of level $i - 1 = 2$ with request lists of $n_3 = 4$ singletons. The set P' is decomposed into a mask of size $\lfloor n_3/2 \rfloor + 1 = 3$, $M = \{p_1, p_2, p_3\}$ and sets $Q_{p_1} = \{p_4, p_5\}$, $Q_{p_2} = \{p_6, p_7\}$ and $Q_{p_3} = \{p_8, p_9\}$. For each $q \in M$, the requested set of interval J_q is $P_q = (M \setminus q) \cup Q_q$.

request sequence consisting of $n_2 = 2$ distinct pages p and q. Clearly, $L(I) = \big\{\{p\}, \{q\}\big\}$. We denote the subtree at I together with the request sequence σ_I as $T_1(\{p, q\})$.

Now, for $i \geq 2$, let us assume that we already know how to construct a tree $T_{i-1}(P)$ which has a single $(i-1)$th level interval J, its request sequence σ_J consists only of pages in P, and for each $p \in P$, $\{p\}$ is contained as a singleton in $L(J)$. Let n_i denote the size of P.

We show how to construct $T_i(P')$, for an arbitrary set P' of n_{i+1} pages, such that T_i has a single ith level interval I, and all pages $p \in P'$ are contained in the request list $L(I)$ as singletons.

First, we create a set of pages $M \subset P'$ called *mask*, such that $|M| = \lfloor n_i/2 \rfloor + 1$. Pages that belong to M are arbitrarily chosen from P'. Then, we partition $P' \setminus M$ into $|M|$ disjoint sets of size $\lceil n_i/2 \rceil$ and we associate each of these sets with a page $q \in M$. We denote by Q_q the set associated to page $q \in M$. For each $q \in M$, let $P_q = (M \setminus \{q\}) \cup Q_q$. Note that $|P_q| = n_i$. The interval tree $T_i(P')$ is created as follows. It consists of a single interval I at level i with $\lfloor n_i/2 \rfloor + 1$ children J_q. For each J_q we inductively create a level $i - 1$ subtree $T_{i-1}(P_q)$. See Figure 2 for an example.

This construction has two important properties:

Lemma III.6. *First, for each $p \in P'$ there exists a subtree $T_{i-1}(P_q)$ such that $p \notin P_q$. Second, for each $p \in P'$ there exists a page $\bar{p} \in P'$ such that each P_q contains either p or $\bar{p}$.*

Proof: If page $p \in M$, then it belongs to all sets P_q except for P_p. If $p \notin M$, then $p \in Q_q$ for some q and hence p only lies in P_q. This proves the first property.

For the second property, if $p \in M$, we can choose an arbitrary $\bar{p} \in P_p$, and note that p lies in every P_q for $q \in M \setminus \{p\}$. On the other hand, if $p \notin M$, let $q \in M$ be such that $p \in Q_q$ (and hence $p \in P_q$) and we define $\bar{p} = q$. Then by construction, q is contained in all other sets $P_{q'}$ for $q' \neq M \setminus \{q\}$. ∎

Using the above lemma, we can now understand the structure of request lists.

Lemma III.7. *The request list $L(I)$ consists of all singletons in P', i.e. $L(I) = \{\{p\} \mid p \in P'\}$.*

Proof: Let us assume by inductive hypothesis that for each child J_q of I we have $L(J_q) = \{\{p\} \mid p \in P_q\}$. As discussed above, this is true for the base case of intervals at level 1.

By Observation III.3 and by the first property in Lemma III.6, no singleton belongs to $R(C)$. By the second property we also know that each $p \in P'$ appears in some pair $\{p, \bar{p}\}$ in $R(C)$. Therefore, by Observation III.4, we know that $R(I)$ contains a singleton $\{p\}$ for each $p \in P'$, and also that $R(I)$ does not contain empty set, since $R(C)$ contains no singletons. So, we have $L(I) = \big\{\{p\} \mid p \in P'\big\}$. ∎

This completes the construction of the tree $T_{k-1}(P)$ with P of size n_k and with a single interval J at level $k - 1$. To finish the service pattern, we create a single kth level interval I having J as its only child. By the discussion above, $L(J)$ contains a singleton $\{p\}$ for each $p \in P$ and trivially the joint request list of the children of I is simply $L(J)$. Therefore we have $n(k, 1) \geq f(k - 1, 1) \geq n_k \geq 2^{2^{k-4}}$. ∎

C. Dichotomy theorems for service patterns

The theorems of this section are matching counterparts to Theorem III.5 — they provide an upper bound for the size of the request lists in a fixed service pattern. Proofs of those theorems can be found in the full version. The first one shows that the parameter $n(k, 1)$ is at most doubly exponential in k.

Theorem III.8 (Dichotomy theorem for k different weights)**.** *Let $\mathcal{I}$ be a k-level service pattern and $I \in \mathcal{I}$ be an arbitrary interval at level k. Let $Q \subseteq U$ be the set of feasible labels for I. Then either $Q = U$, or $|Q| \leq n(k, 1)$ and $n(k, 1)$ can be at most $2^{2^{k+3 \log k}}$.*

This bound is used in Section V to prove an upper bound on the competitive ratio of WFA_λ.

Moreover, we consider the special case when there are only $d < k$ different weights $w_1, \ldots, w_d$. Then, for each $i = 1, \ldots, d$, we have k_i servers of weight w_i. This situation can be modeled using a service pattern with d levels, where each interval at level i can be labeled by at most k_i pages. For such service patterns, we can get a stronger upper bound, which is singly exponential in d, k^2 and the product $\prod k_i$.

Theorem III.9 (Dichotomy theorem for d different weights)**.** *Let $\mathcal{I}$ be a service pattern, $I \in \mathcal{I}$ be an arbitrary interval at level d and let us denote $\mathcal{Q} = \mathcal{Q}_1 \cup \cdots \cup \mathcal{Q}_{k_d}$ a set of labels for I satisfying the following:*

- *Each $\mathcal{Q}_t$ contains feasible labels T for I, such that $|T| = t$.*
- *Whenever T is in $\mathcal{Q}_t$, no T' containing T as a subset belongs to any $\mathcal{Q}_j$ for $j > t$.*

Then, either $Q_1 = U$, or $|Q_t| \leq n(d, t)$ for each t, where $n(d, t) \leq 2^{4dk^2 t \prod_{j=1}^{d-1}(k_j+1)}$.

This theorem is used (in the full version) to prove a performance guarantee for WFA_λ in this special setting.

IV. Online Lower bound

In this section we transform the combinatorial construction of Theorem III.5 into a lower bound for any deterministic algorithm, proving Theorem I.1. Throughout this section we denote $s_1^{\mathrm{ALG}}, \ldots, s_k^{\mathrm{ALG}}$ the servers of the online algorithm and $s_1^{\mathrm{ADV}}, \ldots, s_k^{\mathrm{ADV}}$ the servers of the adversary.

Here is the main idea. Let ALG be a fixed online algorithm. We create a request sequence adaptively, based on the decisions of ALG, which consists of arbitrary number of *phases*. During each phase, the heaviest server of the adversary s_k^{ADV} stays at some fixed location. Whenever a phase ends, the adversary might move all its servers (including s_k^{ADV}) and a new phase may start. During a phase, requests are determined by a recursive construction, using *strategies* which we define later on. At a high-level, the goal of the strategies is to make sure that the following two properties are satisfied:

(i) For $i = 1, \ldots, k-1$, ADV never moves server s_i^{ADV}, unless ALG moves some heavier server s_j^{ALG} for $j > i$ at the same time.

(ii) During each phase, ALG moves its heaviest server s_k^{ALG} at least n_k times.

These two properties already imply a lower bound on the competitive ratio of ALG of order n_k, whenever the weights of the servers are well separated, i.e. $w_{i+1} \geq n_k \cdot \sum_{j=1}^{i} w_j$ for each $1 \leq i < k$. Here $n_k \geq 2^{2^{k-4}}$ is the number of candidate points for s_k^{ADV}.

In the following section we show how each phase is defined using strategies. We conclude the proof of Theorem I.1 in Section IV-B.

A. Definition of Strategies

Each phase is created using k adaptive *strategies* $S_1, \ldots, S_k$, where S_1 is the simplest one and S_{i+1} consists of several executions of S_i. An execution of strategy S_i for $1 \leq i < k$ corresponds to a subsequence of requests where ALG moves only servers $s_1, \ldots, s_i$. Whenever ALG moves some server s_j, for $j > i$, the execution of S_i ends. An execution of S_k ends only when ALG moves its heaviest server to the location of s_k^{ADV}.

We denote by $S_i(P)$ an execution of strategy S_i, with a set of requested points P, where $|P| = n_{i+1}$. We start by defining the strategy of the highest level S_k. An execution $S_k(P)$ defines a *phase* of the request sequence. We make sure that if p is the location of s_k^{ALG} when the execution starts, then $p \notin P$.

Strategy $S_k(P)$**:**

partition P into T and B of size n_k each arbitrarily;
$B' := \emptyset$;
while $T \neq \emptyset$ **do**
  Run $S_{k-1}(T \cup B')$ until ALG moves s_k;
  $p :=$ new position of s_k^{ALG};
  $T := T \setminus \{p\}$;
  $B' :=$ arbitrary subset of $B \setminus \{p\}$ of size $n_k - |T|$;
end while
Terminate Phase

Intuitively, we can think of T as the set of candidate locations for s_k^{ADV}. The set B' is just a padding of new pages to construct a set $T \cup B'$ of size n_k as an argument for S_{k-1}. Whenever s_k^{ALG} is placed at some point $p \in T$, we remove it from T, otherwise T does not change. We then update B' such that $|T \cup B'| = n_k$ and $p \notin B'$. This way, we make sure that p is never requested as long as s_k^{ALG} stays there.

We now define the strategies S_i for $1 < i < k$. An execution of $S_i(P)$ executes several consecutive instances of S_{i-1}. We first describe how do we choose the set P' for each execution of $S_{i-1}(P')$, such that $P' \subset P$ and $|P'| = n_i$.

We use the construction described in Section III-B. In particular, we choose an arbitrary set of points $M \subset P$ called *mask*, and we partition $P \setminus M$ into $|M|$ disjoint sets of equal size, each one associated with a point $q \in M$. We denote by Q_q the set associated to point $q \in M$. All executions of strategy S_{i-1}, have as an argument a set $P_q = (M \setminus \{q\}) \cup Q_q$, for some $q \in M$. We discuss about the size of M and Q_q later on. Before describing the strategies, we observe that Lemma III.6 implies the following two things regarding those sets:

Observation IV.1. *For each point $p \in P$ there is a set P_q such that $p \notin P_q$.*

Observation IV.2. *For any $p \in P$ there is a point $\bar{p}$ such that each P_q contains either p or $\bar{p}$.*

Strategy $S_i(P)$**, where** $1 < i < k$**:**

Decompose P into mask M and sets Q_q;
For each $q \in M$, denote $P_q := (M \setminus \{q\}) \cup Q_q$;
repeat
  $p :=$ position of s_i^{ALG};
  Choose any P_q, s.t. $p \notin P_q$, and run $S_{i-1}(P_q)$ until ALG moves s_i;
until ALG moves s_{i+1} or some heavier server

Last, strategy S_1 takes as an argument a set of $n_2 = 2$ points and requests them in an alternating way.

Strategy $S_1(\{p, q\})$**:**

repeat
  If s_1^{ALG} is at q: request p; Otherwise: request q;
until ALG moves s_2 or some heavier server

Observe that, an execution of a strategy S_i, for $1 \leq i < k$, ends only if ALG moves some heavier server. This means that if ALG decides not to move any heavier server, then the execution continues until the end of the request sequence. Moreover, it is crucial to mention that, by construction of the strategies $S_1, \ldots, S_k$, we have the following:

Observation IV.3. *For $1 \leq i \leq k$, if server s_i^{ALG} is located*

at some point p, then p is never requested until s_i^{ALG} moves elsewhere.

Cardinality of sets: We now determine the size of the sets used by S_i for $1 < i \leq k$. For $2 \leq i \leq k$, recall that all arguments of S_{i-1} should have size n_i. In order to satisfy this, we choose the sizes as in Section III-B, i.e. $|M| = \lceil n_i/2 \rceil + 1$ and $|Q_q| = \lfloor n_i/2 \rfloor$. It is clear that $|P_q| = |(M \setminus \{q\})| + |Q_q| = n_i$. Recall that $P = M \cup (\bigcup_{q \in M} Q_q)$, and therefore we have

$$n_{i+1} = |P| = (\lceil n_i/2 \rceil + 1) + (\lceil n_i/2 \rceil + 1)\lfloor n_i/2 \rfloor =$$
$$= (\lceil n_i/2 \rceil + 1)(\lfloor n_i/2 \rfloor + 1) \geq n_i^2/4.$$

Therefore, by choosing $n_2 = 2$ we have $n_3 = 4$, $n_4 = 9$ and in general for $k \geq 4$

$$n_k \geq 2^{2^{k-4}+2} \geq 2^{2^{k-4}}. \tag{1}$$

Last, for strategy S_k we have that $n_{k+1} = 2n_k$.

Service pattern associated with the request sequence: We associate the request sequence with a service pattern $\mathcal{I}$, which is constructed as follows: For each execution of strategy S_i, we create one interval $I \in \mathcal{I}_i$. We define $\mathcal{I} = \mathcal{I}_1 \cup \cdots \cup \mathcal{I}_k$. Clearly, $\mathcal{I}$ is hierarchical. Next lemma gives a characterization of request lists of intervals $I \in \mathcal{I}$.

Lemma IV.4. *Let I be the interval corresponding to a particular instance $S_i(P)$. Given that there is an interval J at level $j > i$ such that J is labeled by some point $p \in P$, then there is a feasible labeling for I and its descendants covering all requests issued during the lifetime of I.*

In other words, all points $p \in P$ are contained in the request list $L(I)$ as singletons.

Proof: We prove the lemma by induction. The lemma holds trivially for S_1: The requests are issued only in two different points p and q. Whenever J has assigned p, we assign q to I and vice versa. In both cases all the requests during the lifetime of I are covered either by I or by J.

Now, assuming that lemma holds for level $i-1$, let us prove it also for i. Let $p \in P$ be the point assigned to J the ancestor of I. By Observation IV.2, we know that there is a $\bar{p} \in P$ such that each P_q contains either p or $\bar{p}$. We assign $\bar{p}$ to I and this satisfies the condition of the inductive hypothesis for all children of I, as all those instances have one of P_q as an input. ∎

Moreover, by construction of strategy S_k, we get the following lemma.

Lemma IV.5. *The service pattern $\mathcal{I}$ associated with the request sequence is feasible.*

Proof: Let $\mathcal{I}_P$ the service pattern associated with an execution of $S_k(P)$. We show that $\mathcal{I}_P$ is feasible. Since the request sequence consists of executions of S_k, the lemma then follows. Let p be the last point which remained in T during the execution of $S_k(P)$. Then, all the former children of S_k were of form $S_{k-1}(P')$ where $p \in P'$. By Lemma IV.4, by assigning p to the kth level interval, there is a feasible labeling for each children interval and all their descendants. We get that the service pattern $\mathcal{I}_{\mathcal{P}}$ associated with strategy $S_k(P)$ is feasible. ∎

B. Proof of Theorem I.1

We now prove Theorem I.1. We first define the moves of ADV and then we proceed to the final calculation of the lower bound on the competitive ratio of ALG. Recall that the request sequence consists of arbitrary many phases, where each phase is an execution of strategy S_k.

Moves of the adversary: Initially, we allow the adversary to move all its servers, to prepare for the first phase. The cost of this move is at most $\sum_{i=1}^{k} w_i$. Since the request sequence can be arbitrarily long, this additive term does not affect the competitive ratio and we ignore it for the rest of the proof. It remains to describe the moves of the adversary during the request sequence.

Consider the service pattern $\mathcal{I}$ associated with the request sequence. By lemma IV.5, $\mathcal{I}$ is feasible. We associate to the adversary a feasible assignment of $\mathcal{I}$. This way, moves of servers s_i^{ADV} for all i are completely determined by $\mathcal{I}$. We get the following lemma.

Lemma IV.6. *At each time t,* ADV *does not move server s_i^{ADV}, for $i = 1, \ldots, k-1$, unless* ALG *moves some heavier server s_j^{ALG} for $j > i$.*

Proof: Consider the service pattern $\mathcal{I}$ associated with the request sequence. Each execution of strategy S_i is associated with an interval $I_i \in \mathcal{I}$ at level i. The adversary moves s_i^{ADV} if and only if interval I_i ends. By construction, interval I_i ends if and only if its corresponding execution of S_i ends. An execution of S_i ends if and only if ALG moves some heavier server s_j for $j > i$. We get that at any time t, server s_i^{ADV} moves if and only if ALG moves some server s_j for $j > i$. ∎

Calculation of the lower bound: Let $\mathrm{cost}(s_i^{\mathrm{ALG}})$ and $\mathrm{cost}(s_i^{\mathrm{ADV}})$ denote the cost due to moves of the ith server of ALG and ADV respectively. Without loss of generality[5], we assume that $\sum_{i=2}^{k} \mathrm{cost}(s_i^{\mathrm{ALG}}) > 0$. Recall that we assume a strong separation between weights of the servers. Namely, we have $w_1 = 1$ and $w_{i+1} = n_k \cdot \sum_{j=1}^{i} w_i$. This, combined with lemma (IV.6) implies that

$$\sum_{i=1}^{k-1} \mathrm{cost}(s_i^{\mathrm{ADV}}) \leq \big(\sum_{i=2}^{k} \mathrm{cost}(s_i^{\mathrm{ALG}})\big)/n_k. \tag{2}$$

[5]It is easy to see that, if ALG uses only its lightest server s_1^{ALG}, it is not competitive: the whole request sequence is an execution of $S_2(p, q)$, so the adversary can serve all requests at cost $w_2 + w_1$ by moving at the beginning s_2^{ADV} at q and s_1^{ADV} at p. ALG pays 1 for each request, thus its cost equals the length of the request sequence, which implies an unbounded competitive ratio for ALG.

Moreover, by construction of strategy S_k, a phase of the request sequence ends if and only if ALG has moved its heaviest server s_k^{ALG} at least n_k times. For each phase ADV moves s_k^{ADV} only at the end of the phase. Thus we get that

$$\mathrm{cost}(s_k^{\mathrm{ADV}}) \leq \mathrm{cost}(s_k^{\mathrm{ALG}})/n_k. \tag{3}$$

Overall, using (2) and (3) we get

$$\begin{aligned}\mathrm{cost}(\mathrm{ADV}) =& \sum_{i=1}^{k-1} \mathrm{cost}(s_i^{\mathrm{ADV}}) + \mathrm{cost}(s_k^{\mathrm{ADV}}) \\ \leq& \big(\sum_{i=2}^{k} \mathrm{cost}(s_i^{\mathrm{ALG}})\big)/n_k + \mathrm{cost}(s_k^{\mathrm{ALG}})/n_k \\ \leq& 2 \cdot \mathrm{cost}(\mathrm{ALG})/n_k.\end{aligned}$$

Therefore, the competitive ratio of ALG is at least $n_k/2$, which by (1) is $\Omega(2^{2^{k-4}})$. □

V. Upper bounds for generalized WFA

We now show that the generalized Work Function Algorithm with $\lambda = 0.5$ achieves the bounds claimed in Theorems I.2 and I.3. We present the proof of Theorem I.2, as it is simpler and highlights the main ideas directly. For the proof of Theorem I.3, see the full version of this paper.

We prove Theorem I.2 by induction on the number of servers. Let r_k denote the bound on the competitive ratio with k servers. We will show that $r_k = O(n_k^3 r_{k-1})$, where n_k is the constant from the Dichotomy Theorem III.8. As $r_1 = 1$ trivially, this will imply the result. We begin with some definitions and the basic properties of WFA.

Definitions and Notation: Recall the definition of work functions and the WFA_λ from Section I-C. A basic property of work functions is that for any two configurations C and C' and any time t, the work function values $\mathrm{WF}_t(C)$ and $\mathrm{WF}_t(C')$ can differ by at most $d(C, C')$. Moreover, at any time t, the generalized WFA will always be in some configuration that contains the current request σ_t.

Let M_t denote the minimum work function value at time t over all configurations, and let $\mathrm{WF}_t(p) = \min\{\mathrm{WF}_t(C) \mid C(k) = p\}$ denote the minimum work function value over all configurations with the heaviest server s_k at p. We denote $W_i = \sum_{j=1}^{i} w_i$. We will assume (by rounding if necessary) that the weights w_i are well-separated and satisfy $W_{i-1} \leq w_i/(20in_i)$ for each $i = 2, \ldots, k$. This can increase the competitive ratio by at most $O(k^k \Pi_{i=1}^k n_i) \ll O(n_k^2)$. This will ensure that for any two configurations C and C' that both have s_k at p, their work function values differ by at most W_{k-1} which is negligibly small compared to w_k.

For a point $p \in U$, we define the *"static" work function* $\mathrm{SW}_t(p)$ as the optimal cost to serve requests $\sigma_1, \ldots, \sigma_t$ while keeping server s_k fixed at point p. Note that this function will in general take very different values than the (usual) work function. However, the local changes of $\mathrm{SW}(p)$ will be useful in our inductive argument. Intuitively, if the online algorithm (WFA) keeps s_k at p during some interval $[t_1, t_2]$ and $\mathrm{SW}(p)$ rises by x during this period, then the cost incurred by online should be at most $r_{k-1}x$.

For any quantity X, we use $\Delta_{t_1}^{t_2} X := X_{t_2} - X_{t_1}$ to denote the change in X during the time interval $[t_1, t_2]$. If the time interval is clear from the context, we use ΔX.

We partition the request sequence into *phases*, where a phase ends whenever ALG moves its heaviest server s_k^{ALG}.

Basic Properties of WFA: We describe some simple facts that follow from basic properties of WFA_λ and work functions. The proofs of the following lemmas are in the full version.

Lemma V.1. *Consider a phase that starts at time t_1 and end at t_2, and let p be the location of s_k^{ALG} during this phase. Then,*

(i) $M_{t_1} \leq \mathrm{WF}_{t_1}(p) \leq M_{t_1} + W_{k-1}$, *and*
(ii) $w_k/2 - 2W_{k-1} \leq \Delta\,\mathrm{WF}(p) \leq \Delta M + w_k/2 + 2W_{k-1}$.

The next lemma shows that $\mathrm{WF}(p)$ and $\mathrm{SW}(p)$ increase by similar amount while s_k^{ALG} remains at point p.

Lemma V.2. *For a phase where s_k^{ALG} is at point p, we have that $|\Delta\,\mathrm{WF}(p) - \Delta\,\mathrm{SW}(p)| \leq W_{k-1}$.*

We remark that the preceding lemma does not hold for some q where s_k^{ALG} is not present. The next lemma upper bounds the cost of ALG during a phase.

Lemma V.3. *Consider a phase when s_k^{ALG} is at point p. The cost incurred by* ALG *during this phase is at most $w_k + r_{k-1} \cdot (\Delta\,\mathrm{WF}(p) + W_{k-1})$.*

This follows as w_k is the cost of moving the heaviest server, and the second term is the cost incurred by the $k-1$ lighter servers, using the inductive hypothesis.

The following lemma is more general and holds for any point $p \in U$ and for any time interval, even if there are many phases in between.

Lemma V.4. *For any $t' > t$, $p \in U$,*

$$\mathrm{WF}_{t'}(p) \geq \min\{\mathrm{WF}_t(p) + \Delta_t^{t'}\,\mathrm{SW}(p) - W_{k-1}, M_t + w_k\}.$$

Bounding the Performance: We are now ready to prove Theorem I.2. The key lemma will be the following.

Lemma V.5. *Consider any sequence of $m = n_k + 1$ consecutive phases. Then, $\Delta M \geq w_k/(8k \cdot n_k)$ and $\mathrm{cost}(\mathrm{ALG}) \leq 4n_k \cdot r_{k-1} \cdot w_k + r_{k-1} \cdot \Delta M$.*

Before proving Lemma V.5, let us see why gives a competitive ratio $r_k = O(n_k^3) \cdot r_{k-1}$, and hence proves Theorem I.2.

Proof of Theorem I.2: Assume inductively that ALG with $k-1$ servers is r_{k-1}-competitive. For $k = 1$, ALG is obviously 1-competitive. We now bound r_k.

Let $\mathrm{cost}(\mathrm{OPT}) = M_T$, where M_T is the minimum work function value at the end of the request sequence and m the total number of phases. We partition the sequence into $h = \lceil \frac{m}{n_k+1} \rceil$ groups where each group (except possibly the last one) consists of $n_k + 1$ phases. By Lemma V.5, during ith group, $i \leq h-2$, the ratio between $\mathrm{cost}(\mathrm{ALG})$ and ΔM is at most

$$\frac{4n_k \cdot w_k \cdot r_{k-1} + r_{k-1}\Delta M}{\Delta M} \leq \frac{4n_k \cdot w_k \cdot r_{k-1}}{w_k/(8k \cdot n_k)} + \frac{r_{k-1}\Delta M}{\Delta M} \leq 33k \cdot n_k^2 \cdot r_{k-1}. \tag{4}$$

Since the last group might contain less than n_k+1 phases, we treat the last two groups together[6]. By a similar calculation as in (4) we get that the ratio between $\mathrm{cost}(\mathrm{ALG})$ and ΔM during groups $h-1$ and h is at most $65kn_k^2 \cdot r_{k-1}$. We conclude that for any request sequence,

$$r_k \leq \frac{\mathrm{cost}(\mathrm{ALG})}{M_T} \leq 65kn_k^2 \cdot r_{k-1} = O(n_k^3 r_{k-1}).$$

Assuming that $r_{k-1} \leq 2^{2^{k+5\log k}}$, and as $n_k = 2^{2^{k+3\log k}}$ and $\log 65k < 2^{k+3\log k}$, it follows that

$$\log r_k \leq 2^{k+1+5\log(k+1)}.$$

■

We now focus on proving Lemma V.5. The crucial part is to lower bound the increase in ΔM during the m phases. Let t_1 and t_2 denote the start and end times of the m phase sequence. We will show that for all points p, $\mathrm{WF}_{t_2}(p) \geq M_{t_1} + w_k/(8k \cdot n_k)$. To do this, we upper bound the number of points p where the increase in $\mathrm{WF}(p)$ could be very small in the first phase (Lemma V.6). Then, using Lemma V.1 we show that, during the subsequent m phases, s_k^{ALG} will visits all such points p which would increase $\mathrm{WF}(p)$ significantly for each of them. We now give the details.

Call a point q *lucky* during a phase, if its static work function increases by at most $\Delta\,\mathrm{SW}(q) < w_k/(4kn_k)$ during that phase. The next lemma shows that there cannot be too many lucky points during a phase.

Lemma V.6. *Let L be the set of lucky points during some phase. Then, $|L| \leq n_k$.*

Proof: For the sake of contradiction, suppose that $|L| > n_k$. Let Q be an arbitrary subset of L such that $|Q| = n_k + 1$. For each $q \in Q$, let $\mathcal{I}^q$ be the optimal service pattern for the phase where s_k remained at q throughout. Clearly, $\mathrm{cost}(\mathcal{I}^q) \leq \Delta\,\mathrm{SW}(q)$.

We create a new service pattern $\mathcal{I}$ that is a *refinement* of all $\mathcal{I}^q$, for $q \in Q$ as follows. For each $\ell = 1, \ldots, k$, we set $\mathcal{I}_\ell = \{[t_i, t_{i+1}) \mid \text{for } i = 1, \ldots, s-1\}$, where $t_1 < \cdots < t_s$ are the times when at least one interval from $\mathcal{I}_\ell^1, \ldots, \mathcal{I}_\ell^{|Q|}$ ends. This way, each interval $I \in \mathcal{I}_\ell^q$ is a union of some intervals from $\mathcal{I}_\ell$. Let $\mathcal{I} = \mathcal{I}_1 \cup \cdots \cup \mathcal{I}_k$. Note that any feasible labeling α for any $\mathcal{I}^q$, extends naturally to a feasible labeling for $\mathcal{I}$: If an interval $I \in \mathcal{I}^q$ is partitioned into smaller intervals, we label all of them with $\alpha(I)$.

We modify $\mathcal{I}$ to be hierarchical, which increases its cost at most by a factor of k. By construction, we have

$$\mathrm{cost}(\mathcal{I}) \leq k \cdot \sum_{q \in Q} \mathrm{cost}(\mathcal{I}^q) \leq k \cdot \sum_{q \in Q} \Delta\,\mathrm{SW}(q) \leq k(n_k+1) \cdot \frac{w_k}{4kn_k} \leq \frac{w_k}{3}. \tag{5}$$

Now the key point is that $\mathcal{I}$ has only one interval I at level k, and all $q \in Q$ can be feasibly assigned to it. But by the Dichotomy theorem III.8, either the number of points which can be feasibly assigned to I is at most $n(k,1)$, or else any point can be feasibly assigned there. As $|Q| > n_k \geq n(k,1)$, this implies that any point can be feasibly assigned to I. Let p be the location of s_k^{ALG} during the phase. One possible way to serve all requests of this phase having s_k at p is to use $\mathcal{I}$ (with possibly some initial cost of at most W_{k-1} to bring the lighter servers in the right configuration). This gives that,

$$\Delta\,\mathrm{SW}(p) \leq \mathrm{cost}(\mathcal{I}) + W_{k-1} \leq w_k/3 + W_{k-1}. \tag{6}$$

On the other hand, by Lemma V.1, during the phase $\Delta\,\mathrm{WF}(p) \geq w_k/2 - 2W_{k-1}$, and by Lemma V.2, $\Delta\,\mathrm{SW}(p) \geq \Delta\,\mathrm{WF}(p) - W_{k-1}$. Together, this gives $\Delta\,\mathrm{SW}(p) \geq w_k/2 - 3W_{k-1}$ which contradicts (6), as $W_{k-1} \ll w_k/40k$. ■

The next simple observation shows that if a point is not lucky during a phase, its work function value must be non-trivially high at the end of the phase.

Observation V.7. *Consider a phase that starts at time t and ends at t'. Let p be a point which is not lucky during that phase. Then, $\mathrm{WF}_{t'}(p) \geq M_t + w_k/(5k \cdot n_k)$.*

Proof: By Lemma V.4 we have either $\mathrm{WF}_{t'}(p) \geq M_t + w_k$, in which case the result is trivially true. Otherwise, we have that

$$\mathrm{WF}_{t'}(p) \geq \mathrm{WF}_t(p) + \Delta_t^{t'}\,\mathrm{SW}(p) - W_{k-1}.$$

But as p is not lucky, $\Delta\,\mathrm{SW}(p) \geq w_k/(4kn_k)$, and as $W_{k-1} \leq w_k/(20k \cdot n_k)$, together this gives have that $\mathrm{WF}_{t'}(p) \geq \mathrm{WF}_t(p) + w_k/(5k \cdot n_k)$. ■

Proof of Lemma V.5: We first bound the cost of the algorithm. We denote by $\mathrm{cost}_i(\mathrm{ALG})$ the cost of ALG during ith phase. Let p_i be the location of s_k^{ALG}, and $\Delta_i M$ the increase of M during the ith phase. By Lemma V.3, $\mathrm{cost}_i(\mathrm{ALG})$ is at most $w_k + r_{k-1} \cdot \Delta\,\mathrm{WF}(p_i)$. Also, by Lemma V.1, we have that $\Delta\,\mathrm{WF}(p_i) \leq \Delta_i M + w_k/2 + 2W_{k-1}$. Combining those two bounds, and using that $W_{k-1} \leq w_k/(20k \cdot n_k)$, we get that $\mathrm{cost}_i(\mathrm{ALG}) \leq$

[6] A minor technical detail: We should make sure that the bound of r_k holds also in case the request sequence consists only of one group. This case can be easily handled, as we explain in the full version.

$2 \cdot r_{k-1} w_k + r_{k-1} \cdot \Delta_i M$. By summing over all phases, we get the desired upper bound.

Now, we show the lower bound on ΔM. Let t_1 and t_2 be the start and the end time of the $m = n_k + 1$ phases.

For the sake of contradiction, suppose that $M_t < M_{t_1} + w_k/(8k \cdot n_k)$ for all $t \in [t_1, t_2]$. By Lemma V.6, during first phase there are at most n_k lucky points. We claim that s_k^{ALG} must necessarily visit some lucky point in each subsequent phase. For $1 \leq i \leq m$, let Q_i denote the set of points that have been lucky during all the phases $1, \ldots, i$. Let t denote the starting time of ith phase and p the location of s_k^{ALG} during this phase, for any $i \geq 2$. By Lemma V.1, we have

$$\mathrm{WF}_t(p) < M_t + W_{k-1} \leq M_{t_1} + w_k/(5k \cdot n_k).$$

By Observation V.7, this condition can only be satisfied by points $p \in Q_{i-1}$ and hence we get that p was lucky in all previous phases. Now, by Lemma V.1, ith phase $\mathrm{WF}(p)$ rises by at least $w_k/2 - 2W_{k-1}$ during the ith phase, and hence p is not lucky. Therefore, $p \notin Q_i$ and $p \in Q_{i-1}$ and $|Q_i| \leq |Q_{i-1}| - 1$. Since $|Q_1| \leq n_k = m - 1$, we get that $Q_m = \emptyset$, which gives the desired contradiction. ■

ACKNOWLEDGMENTS

We are grateful to René Sitters for first bringing the problem to our attention. We would like to thank Niv Buchbinder, Ashish Chiplunkar and Janardhan Kulkarni for several useful discussions. We would also like to thank the anonymous referees for some useful comments. Part of the work was done when NB and ME were visiting the Simons Institute at Berkeley and we thank them for their hospitality.

REFERENCES

[1] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*. Cambridge University Press, 1998.

[2] E. Koutsoupias and C. H. Papadimitriou, "On the k-server conjecture," *J. ACM*, vol. 42, no. 5, pp. 971–983, 1995.

[3] E. Koutsoupias, "The k-server problem," *Computer Science Review*, vol. 3, no. 2, pp. 105–118, 2009.

[4] R. Sitters, "The generalized work function algorithm is competitive for the generalized 2-server problem," *SIAM J. Comput.*, vol. 43, no. 1, pp. 96–125, 2014.

[5] N. Bansal, N. Buchbinder, A. Madry, and J. Naor, "A polylogarithmic-competitive algorithm for the *k*-server problem," *J. ACM*, vol. 62, no. 5, p. 40, 2015.

[6] M. S. Manasse, L. A. McGeoch, and D. D. Sleator, "Competitive algorithms for server problems," *J. ACM*, vol. 11, no. 2, pp. 208–230, 1990.

[7] D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules," *Commun. ACM*, vol. 28, no. 2, pp. 202–208, 1985.

[8] M. Chrobak, H. J. Karloff, T. H. Payne, and S. Vishwanathan, "New results on server problems," *SIAM J. Discrete Math.*, vol. 4, no. 2, pp. 172–181, 1991.

[9] M. Chrobak and L. L. Larmore, "An optimal on-line algorithm for k-servers on trees," *SIAM J. Comput.*, vol. 20, no. 1, pp. 144–148, 1991.

[10] E. Koutsoupias and C. H. Papadimitriou, "The 2-evader problem," *Inf. Process. Lett.*, vol. 57, no. 5, pp. 249–252, 1996.

[11] A. Fiat and M. Ricklin, "Competitive algorithms for the weighted server problem," *Theor. Comput. Sci.*, vol. 130, no. 1, pp. 85–99, 1994.

[12] N. E. Young, "The k-server dual and loose competitiveness for paging," *Algorithmica*, vol. 11, no. 6, pp. 525–541, 1994.

[13] N. Bansal, N. Buchbinder, and J. Naor, "A primal-dual randomized algorithm for weighted paging," *J. ACM*, vol. 59, no. 4, pp. 19:1–19:24, 2012.

[14] A. Chiplunkar and S. Vishwanathan, "On randomized memoryless algorithms for the weighted k-server problem," in *Foundations of Computer Science, FOCS*, 2013, pp. 11–19.

[15] S. Ben-David, A. Borodin, R. M. Karp, G. Tardos, and A. Wigderson, "On the power of randomization in on-line algorithms," *Algorithmica*, vol. 11, no. 1, pp. 2–14, 1994.

[16] A. Chiplunkar, personal Communication. Oct 2016.

[17] A. Borodin, N. Linial, and M. E. Saks, "An optimal on-line algorithm for metrical task system," *J. ACM*, vol. 39, no. 4, pp. 745–763, 1992.

[18] M. Chrobak and L. L. Larmore, "Metrical task systems, the server problem and the work function algorithm," in *Online Algorithms, The State of the Art*, 1996, pp. 74–96.

[19] M. Chrobak and J. Sgall, "The weighted 2-server problem," *Theor. Comput. Sci.*, vol. 324, no. 2-3, pp. 289–312, 2004.

[20] E. Koutsoupias and D. S. Taylor, "The CNN problem and other k-server variants," *Theor. Comput. Sci.*, vol. 324, no. 2-3, pp. 347–359, 2004.

[21] R. A. Sitters and L. Stougie, "The generalized two-server problem," *J. ACM*, vol. 53, no. 3, pp. 437–458, 2006.

[22] R. Sitters, L. Stougie, and W. de Paepe, "A competitive algorithm for the general 2-server problem," in *ICALP*, 2003, pp. 624–636.

[23] M. Chrobak, "SIGACT news online algorithms column 1," *SIGACT News*, vol. 34, no. 4, pp. 68–77, 2003.

[24] J. Augustine and N. Gravin, "On the continuous CNN problem," in *ISAAC*, 2010, pp. 254–265.

[25] K. Iwama and K. Yonezawa, "Axis-bound cnn problem," *IEICE TRANS*, pp. 1–8, 2001.

[26] ——, "The orthogonal CNN problem," *Inf. Process. Lett.*, vol. 90, no. 3, pp. 115–120, 2004.