Algorithms for k-server problems

Algorithms for k-Server Problems

Grigorios Koumoutsos

This research was supported by the European Research Council (ERC) grant 617951 ("Algorithms for coping with uncertainty and intractability").

A catalogue record is available from the Eindhoven University of Technology Library.

ISBN: 978-94-9301-405-3

Printed by Gildeprint, Enschede.

Algorithms for k-Server Problems

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op donderdag 6 september 2018 om 16:00 uur

door

Grigorios Koumoutsos

geboren te Amarousion, Griekenland

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

Voorzitter:	prof. dr. ir. B. Koren
Promotor:	prof. dr. N. Bansal
Copromotoren:	dr. J. Nederlof
	dr. A. Rosén (CNRS and University Paris Diderot)
Leden:	prof. dr. E. Koutsoupias (University of Oxford)
	prof. dr. J. Naor (Technion)
	prof. dr. F.C.R. Spieksma
	prof. dr. M.T. de Berg
	prof. dr. L. Stougie (Vrije Universiteit Amsterdam)

Het onderzoek dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

Acknowledgments

I would like to express my deep gratitude to my advisor Nikhil Bansal for his guidance and support. His passion about research was a great source of motivation and it was a pleasure to work with him. I would also like to thank him for teaching me how to think in a simple way and how to present my work to a broad audience. Thanks also for the financial support for attending many conferences, workshops and summer schools around the world. I could not have asked for better conditions for pursuing a PhD.

I would like to thank the members of my thesis committee. Jesper Nederlof, Leen Stougie, Seffi Naor, Frits Spieksma and Mark de Berg, thank you all for reviewing my thesis and for being part of my graduation. Special thanks to Elias Koutsoupias, my undergraduate supervisor, for being a member of the committee and for his inspiring lectures back in fall 2010 at the University of Athens, which served as a springboard for my journey in the area of algorithms. I am grateful to Adi Rosén for his supervision, support and advice since 2013, and for his hospitality and financial support for my research visits in Paris.

I would like to thank Marek Eliáš for being a colleague, a collaborator, a coauthor, a roommate in many trips and a friend. I would also like to thank all my coauthors. Łukasz Jeż, Kirk Pruhs, Jesper Nederlof, Seeun William Umboh and Martin Böhm, I really enjoyed working with you all. Thanks to László Kozma, Janardhan Kulkarni, Spyros Angelopoulos, Marc Renault and René Sitters for many stimulating research discussions.

Thanks to Bouke Cloostermans, Tjark Vredeveld and Shashwat Garg for the nice environment at the office. Thanks to Jorn van der Pol and Aleksandar Markovic for organizing great dinners in Eindhoven.

During the PhD studies it was sometimes hard to keep a balance between work and personal life. I would like to thank my friends for the nice moments I had with them. Jerry, Stelios, Christos, Tilemachos, Danae and Kyveli, thank you for discovering Maastricht together. Special thanks to all my friends from Greece for their presence and support at all times and for the great trips during those years. I am not mentioning your names, you know who you are!

I am grateful to my parents for their unconditional support and love. Thanks to my father for making me love Mathematics and Computer Science

since childhood. Thanks to my mother for all her patience and understanding. Thanks to my brother for his presence and love at all times.

Last and most importantly, I would like to thank Mina for standing by me during the PhD life. Mina, in the last 8 years you have been a girlfriend, a partner, the best friend, a flatmate, a travel buddy and a "FightFun" classmate. Thank you for your unlimited support throughout the highs and lows of those years. Thank you for sharing your life with me. I look forward to everything that is to come.

Contents

1 Introduction **1**
- 1.1 Online Algorithms . . . 1
  - 1.1.1 Competitive Analysis . . . 2
  - 1.1.2 Metrical Task Systems . . . 4
  - 1.1.3 The k-Server Problem . . . 5
- 1.2 The (h, k)-Server Problem . . . 7
- 1.3 The Generalized k-Server Problem . . . 8

2 Tight Bounds for Double Coverage Against Weak Adversaries **11**
- 2.1 Introduction . . . 11
- 2.2 Lower Bound . . . 14
- 2.3 Upper Bound . . . 20
- 2.4 Extension to Trees . . . 23
- 2.5 Competitive Ratio of DC for the Paging Problem . . . 26

3 The (h, k)-Server Problem on Bounded Depth Trees **29**
- 3.1 Introduction . . . 29
- 3.2 Lower Bound for Double Coverage on Depth-2 HSTs . . . 31
- 3.3 Algorithm for Depth-d Trees . . . 33
  - 3.3.1 Algorithm Description . . . 34
  - 3.3.2 Analysis . . . 35
- 3.4 Algorithm for Bounded-Diameter Trees . . . 42
- 3.5 Lower Bounds . . . 42
  - 3.5.1 General Lower Bound for Depth-2 HSTs . . . 42
  - 3.5.2 Lower Bound for WFA on Depth-3 HSTs . . . 45
- 3.6 Tight Bounds for WFA on the Line . . . 59
- 3.7 Concluding Remarks . . . 61

4 Competitive Algorithms for Generalized k-Server in Uniform Metrics **63**
- 4.1 Introduction . . . 63

4.2 Deterministic Algorithm for Uniform Metrics 66
4.3 Randomized Algorithm for Uniform Metrics 69
4.4 Algorithm for Weighted Uniform Metrics 74
4.4.1 Algorithm Description 75
4.4.2 Analysis 76
4.5 Lower Bounds 79
4.6 Conclusion and Open Problems 80

Bibliography **83**

Summary **89**

Curriculum Vitae **91**

Chapter 1

Introduction

1.1 Online Algorithms

In classical optimization, we are given a problem and a specific input, and the goal is to find the optimal solution for the given input. However, in many real life applications, the assumption that the whole input is available, is not realistic. Most of the times, we need to solve optimization problems, while taking decisions with incomplete information about the input. Below we list some examples of such problems.

- **Paging (Caching):** This is a classical problem in operating systems design. We are given a two-level memory system, composed by the *cache* and the main memory. The cache is divided into k parts of equal size, called *pages*, while the main memory has larger capacity, but it is much slower. Whenever a page needs to be accessed by the CPU, it should be in the cache; if it is not, a *page fault* occurs. The page needs to be fetched in the cache, possibly by evicting some other page. Thus, any operating system needs a *page eviction policy*. Here, the input can be seen as a sequence of requests to pages, and the goal is to minimize the number of page faults. In case all the requests are known in advance, it is easy to find the optimal solution: Whenever a page eviction is needed, we evict the page that will be requested the latest in the future. However, in reality the future requests are not known, and the operating system has to decide which page to evict taking into account only the requests seen so far.

- **Online Advertising:** Consider a search engine like Google or Yahoo!. Whenever a query is received, apart from the search results, some ads are also displayed. Every day, for each ad there is a fixed number of times it should appear, depending on the contract with the advertised business.

For each query, the displayed ads should be targeted, so that there is a good chance that the user will click on them. Clearly, if all the queries of the day were known in advance, the search engine could choose which ads to display to each query in order to maximize the expected revenue. However, this information is not available, and whenever a user submits a query, the search engine should decide immediately which ads to show.

- **Vehicle Routing:** Imagine a service provider with a certain number of vehicles (e.g. taxis, firetrucks, ambulances, etc.). Whenever a client needs a service, it submits a request, specifying a starting point and a destination. The provider should send a vehicle to serve the request. At the time a request is issued, there is no information about future requests, and the provider should decide immediately which vehicle will serve the request. The goal is to minimize the total distance traveled by the vehicles.

All the examples above show problems where the input is not available from the beginning, but it arrives over time and our goal is to optimize some objective function. We call such problems *online optimization problems*, or simply online problems. To solve problems in online optimization, we need to devise algorithms which make decisions without knowledge of the future. We call such algorithms *online algorithms*.

Model. Formally, in an online optimization problem the input σ is divided into *requests* $\sigma = \sigma_1, \ldots, \sigma_m$. Whenever each request σ_t is received, some actions must be performed, without knowledge of the future requests $\sigma_{t+1}, \ldots, \sigma_m$. An algorithm that solves an online optimization problem is called an *online algorithm*. An algorithm which reads the whole input σ and then produces a solution is called an *offline algorithm*.

1.1.1 Competitive Analysis

The standard framework used to evaluate the performance of online algorithms is *competitive analysis*, which was introduced by Sleator and Tarjan [66]. Here, the performance of an online algorithm is compared to the optimal *offline* solution which knows the whole input in advance.

From now on, we focus on minimization problems with computable optimal solution, since this is the case for all problems considered in this thesis. Let P be a minimization problem and let I denote the set of all valid inputs for P. For an instance $\sigma \in I$, let $\mathrm{OPT}(\sigma)$ denote the optimal cost on σ. For an online algorithm ALG, let $\mathrm{ALG}(\sigma)$ denote the cost of ALG on σ.

Definition 1.1. *An online algorithm* ALG *for an online minimization problem* P *is* c*-competitive if there exists a constant* α *such that for any input* $\sigma \in I$*, we have that*

$$\mathrm{ALG}(\sigma) \leq c \cdot \mathrm{OPT}(\sigma) + \alpha.$$

If $\alpha = 0$ we say that the algorithm ALG is *strictly* c*-competitive* [38]. The competitive ratio of an algorithm ALG is the infimum value c such that ALG is c-competitive. The competitive ratio of an online minimization problem P is the infimum value c for which a c-competitive algorithm for P exists.

Note that there is no restriction on the computational resources used by the online algorithm; the primary goal in competitive analysis is to to understand the importance of knowing the future. The competitive ratio of an online problem is the loss factor due to lack of information, assuming unlimited computational power. In practice, we seek efficient algorithms that achieve the optimal or a nearly optimal competitive ratio.

An alternative view of competitive analysis is to think of each online problem as a game between an algorithm and an all-powerful adversary. The adversary knows the description of the algorithm and constructs an input in order to maximize the ratio between the cost of the algorithm and the optimal cost.

Randomized Algorithms: A usual approach in the design of online algorithms is to use randomization. In competitive analysis, there are various adversary models proposed to evaluate randomized algorithms.

Oblivious Adversaries: In this model, the adversary knows the description of the algorithm, but it does not know its random choices and it has to construct the whole input before the algorithm starts serving the requests. A randomized online algorithm ALG for a minimization problem P is c-competitive against oblivious adversaries if there exists a constant α, such that for any request sequence σ generated by an oblivious adversary, $E[\mathrm{ALG}(\sigma)] \leq c \cdot \mathrm{OPT}(\sigma) + \alpha$.

Adaptive Online Adversaries: Here, the adversary knows all actions of the algorithm, including its random choices. At each step, the adversary generates a request in order to maximize the cost incurred by the algorithm. However, the adversary must also serve the request sequence online. This way, the costs of both the algorithm and the adversary depend on the random choices of the algorithm. A randomized online algorithm ALG for a minimization problem P is c-competitive against adaptive online adversaries, if there exists a constant α, such that for any request sequence σ generated by an adaptive online adversary ADV, $E[\mathrm{ALG}(\sigma)] \leq c \cdot E[\mathrm{ADV}(\sigma)] + \alpha$, where $\mathrm{ADV}(\sigma)$ is the cost of ADV to serve σ.

Typically the oblivious adversary model allows improved competitive ratios compared to deterministic algorithms. For the rest of this thesis we focus on the

oblivious adversary model, unless stated otherwise. We refer the reader to [18] for a more detailed discussion on different adversary models for randomized algorithms and connections between them.

An excellent reference on competitive analysis is [19].

1.1.2 Metrical Task Systems

Competitive analysis has been used to analyze online algorithms for many important online problems. Initially, each problem was solved in a rather ad-hoc way, using problem-specific techniques. In order to create a unifying framework that enables the study of online algorithms in a systematic way, Borodin et al. [20] defined the problem of *metrical task systems* (MTS), which is a great generalization of various well-studied online problems.

In the MTS problem we are given a server which can be in one of N different states and a metric distance function d specifying the cost of switching between the states. At each time step, a task r arrives, represented by a vector $r = (r_1, \ldots, r_N)$, where r_i is the cost of processing r at state i. The server has to decide in which state it will process the task. If it switches from state i to state j and processes the task there, it incurs a cost $d(i, j) + r_j$. Given an initial state and a sequence of tasks, the goal is to process all tasks at minimum cost.

Metrical Service Systems (MSS): In [32, 31, 57] a slight restriction of the MTS model was introduced, called metrical service systems (MSS)[1]. Here, each component of each task vector is either 0 or ∞. Therefore, each task can be processed only in a subset of the states, whose coefficient is 0 (we call them *feasible* states for this task).

It is easy to see that the paging problem is a special case of both MTS and MSS: states correspond to all possible sets of pages in the cache and the cost of switching between states equals the number of different pages between the corresponding sets. Whenever a page p is requested, all the states that contain p in the cache are feasible and the rest of the states are infeasible. Other notable special cases of MTS include fundamental data structure problems such as the list update problem [66, 62, 2] and the binary search tree problem [67, 3, 46].

In the general case, i.e. when the task vectors are arbitrary, the deterministic competitive ratio is $2N-1$ for MTS [20] and $N-1$ for MSS [57]. For randomized algorithms the competitive ratio of both problems is $O(\log^2 N \log\log N)$ [40] and $\Omega(\log N)$ [20].

All problems considered in the rest of this thesis are special cases of MTS and MSS with special structure, where usually a competitive ratio independent of the number of states is possible.

[1] In [57] it was called *forcing task systems*.

1.1.3 The k-Server Problem

The k-server problem is one of the most important and well-studied special cases of the MTS and MSS problems. It was introduced by Manasse et al. [57] as a far-reaching generalization of various online problems, the most notable of which is the paging problem. The study of the k-server problem has led to various remarkable developments in competitive analysis. One of the reasons is its intriguing level of generality: on the one hand it is a special case of MSS where a competitive ratio independent of the number of states can be achieved, on the other hand it is a generalization of many online problems and requires the development of generic and powerful techniques for online algorithms.

Formally, the k-server problem is defined in a metric space $M = (U, d)$, where U is a set of n points and $d : U^2 \to \mathcal{R}$ is a (non-negative, symmetric) distance function which satisfies the triangle inequality. There are k mobile servers located at some points of the metric space. The input is a request sequence $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_m$, where $\sigma_t \in U$ is the point requested at time t. To serve the request, some server must move at σ_t. The goal is to minimize the total distance traveled by the servers for serving σ.

Connection with MSS and paging. Note that the k-server problem can be viewed as a MSS on $N = \binom{n}{k}$ states corresponding to all possible configurations of the k servers. The distance between two states equals the minimum cost perfect matching between the corresponding configurations. For each request, the set of feasible MSS states corresponds to all configurations which have a server at the requested point.

The paging problem is the special case of the k-server problem in a uniform metric, i.e. when all distances between distinct points are 1. Here, the k-servers correspond to the k slots in the cache, and the pages correspond to the points. Evicting a page from the cache and bringing a new one maps to moving a server between the corresponding points at a cost of 1. Equivalently, we can think of a uniform metric as a star graph on n leaves corresponding to the n pages, where all edges have length 1/2.

Background. In their seminal paper, Manasse et al. [57] showed that the competitive ratio of deterministic algorithms is at least k, even if the metric space contains $n = k+1$ points. For the paging problem, Sleator and Tarjan [66] showed that many natural algorithms are k-competitive. Manasse et al. [57] conjectured that this competitive ratio can be achieved in any metric space.

Conjecture 1.2 (The k-Server Conjecture)**.** *For any metric space, there exists a k-competitive deterministic online algorithm.*

Qualitatively, this means that general metrics are believed to be no harder than the simplest possible case of uniform metrics. The k-server conjecture attracted a lot of attention and it has influenced the research on online algorithms over the last three decades.

At the time the k-server conjecture was posed, it was not even known whether a competitive ratio $f(k)$ depending only on k is possible for general metric spaces. The initial research focused on special metrics like weighted stars, lines and trees, and for many cases tight k-competitive algorithms were obtained [29, 30, 42, 54] (we discuss some of those results in detail in Chapter 2).

For general metric spaces, Fiat et al. [41] obtained the first $f(k)$-competitive algorithm, with competitive ratio $O((k!)^3)$. Several improvements followed [5, 44, 18], but the ratio was still exponential in k. Then, in a breakthrough result, Koutsoupias and Papadimitriou [53] showed that the Work Function Algorithm (WFA) is $(2k-1)$-competitive for every metric space, almost resolving the conjecture. This remains up to date the best known upper bound on the competitive ratio of the k-server problem.

We refer the reader to [19, 52] for an extensive treatment of the large body of work on the k-server conjecture.

Randomized Algorithms: Typically, using randomization, an exponential improvement on the competitive ratio is possible. However, randomized online algorithms are not well understood compared to the deterministic ones.

For the k-server problem, it is believed that, similarly to the deterministic case, the distance function does not affect the competitive ratio and an $O(\log k)$-competitive randomized algorithm against oblivious adversaries is possible in any metric space (the so-called *randomized k-server conjecture*). However, this is known to be true only for very special cases. In particular, for the paging problem several $O(\log k)$-competitive randomized algorithms are known [1, 39, 59]. Nevertheless, even the simple generalization of the weighted paging problem (which corresponds to the k-server problem on weighted star graphs) remained open for almost two decades, until Bansal et al. [8] gave an $O(\log k)$-competitive algorithm using the primal-dual method.

More recently, polylog(k, n) competitive ratios for general metric spaces were obtained [6, 21]. Those bounds are better than the deterministic competitive ratio $2k-1$ in case n is sub-exponential in k. The techniques developed in those works imply an $O(\log k)$-competitive randomized algorithm for hierarchically separated trees (HSTs – defined formally in Chapter 3) of constant depth. Very recently, an $O(\log^6 k)$-competitive algorithm for any metric space was claimed [56].

1.2 The (h, k)-Server Problem

While the work on the k-server problem has been quite influential in the development of competitive analysis, it also reveals some of its drawbacks. The assumption that the input is created by a spiteful adversary is too pessimistic and might enable strong lower bounds on the competitive ratio. For the k-server problem, the competitive ratio grows linearly with k. That means, when the number of servers k increases, the competitive ratio worsens. Intuitively, when the resources of the online algorithm increase, we expect its performance to improve. This can not be captured by competitive analysis, since the cost of the algorithm is compared to the optimal solution with the same number of servers.

Resource Augmentation. A well-known approach to overcome the pessimistic lower bounds for online algorithms is called *resource augmentation*. Here, we assume that the online algorithm is provided with some extra resources, which are not available for the adversary. This approach has led to spectacular success in online scheduling problems, see e.g. [50, 61, 14, 25].

In the context of the k-server problem, the resource augmentation is defined by providing more servers to the online algorithm than the adversary. In particular, the online algorithm has k servers, but its performance is compared to an offline optimal algorithm with $h \leq k$ servers. This is called the (h, k)-server problem, also known as the weak adversaries model for the k-server problem [51].

Previous Work. In their seminal paper [66], Sleator and Tarjan gave several $\frac{k}{k-h+1}$-competitive algorithms for uniform metrics (the (h, k)-paging problem) and also showed that this is the best possible ratio. This bound was later extended to the weighted star metric (weighted paging) [68]. Note that this guarantee equals k for $k = h$ (the usual k-server setting), and tends to 1 as k/h approaches infinity. In particular, for $k = 2h$, this is smaller than 2.

This shows that the resource augmentation model gives a more accurate interpretation on the performance of online algorithms: The competitive ratio improves substantially when the number of servers grows.

Interestingly, in contrast to the classic k-server problem, the (h, k)-server problem is not well understood beyond uniform metrics. Prior to our work, no $o(h)$-competitive algorithm was known, even for very simple extensions of the weighted star, like trees of depth 2. In particular, for $k \gg h$ it was not even known whether using k servers any better performance can be achieved compared to simply disabling the extra $k - h$ servers and run the standard k-server algorithms using only h servers.

Our Contribution. In this thesis we initiate a systematic study of the (h, k)-server problem. In Chapter 2, we focus on the (h, k)-server problem in the euclidean line and in trees. We consider the Double Coverage (DC) algorithm, which is known to be k-competitive for the standard k-server problem in those metric spaces. We show that, surprisingly, its performance does not improve as k increases, and its competitive ratio is exactly $\frac{k(h+1)}{k+1}$. Note that this ratio equals h for $k = h$ and tends to $h + 1$ as k grows.

In Chapter 3, we consider trees of bounded depth. We show that the previously known k-server algorithms (DC and WFA) do not improve as k increases and they have competitive ratio $\Omega(h)$. Furthermore, we design a new algorithm which is $O(1)$-competitive for trees of constant depth, whenever $k = (1 + \epsilon)h$ for any $\epsilon > 0$. This gives the first $o(h)$-competitive algorithm for any metric space other than the weighted star. Finally, we give a general lower bound that any deterministic online algorithm has competitive ratio at least 2.4, even for trees of depth 2 and when k/h is arbitrarily large. This gives a surprising qualitative separation between trees of depth 1 (weighted star) and trees of depth 2 for the (h, k)-server problem.

Note on joint work. Chapter 2 is based on joint work with Nikhil Bansal, Marek Eliáš, Łukasz Jeż and Kirk Pruhs, which is published in *Theory of Computing Systems* [11]. A preliminary version of this work appeared in the *13th International Workshop on Approximation and Online Algorithms* (WAOA), 2015 [10]. Chapter 3 is based on joint work with Nikhil Bansal, Marek Eliáš and Łukasz Jeż. A preliminary version of this work appeared in the *28th ACM-SIAM Symposium on Discrete Algorithms* (SODA), 2017 [9].

1.3 The Generalized k-Server Problem

The study of the k-server problem has been essential in the development of powerful techniques for online algorithms. For example, the landmark result of Koutsoupias and Papadimitriou [53] on the k-server conjecture enabled the belief that the WFA (or the generalized WFA [24]) performs optimally for any metrical task system. Furthermore, the work on randomized k-server algorithms enabled the development of powerful techniques using the primal-dual method [23, 22, 7, 6] and more recently the mirror descent method [21].

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. Getting a better understanding of such problems is a natural step towards building a deeper theory of online computation. Below we list some examples of server problems that are not

captured by the standard k-server model:

- **The weighted k-server problem** [60]. Here servers have different weights $w_1, \ldots, w_k$ and the cost of moving the ith server by distance d is $w_i \cdot d$. This problem is substantially different from the (unweighted) k-server problem. To get a feel for the problem, for uniform metrics the competitive ratio is $2^{2^{\Theta(k)}}$ [43, 27, 12], and no competitive algorithms are known for general metrics.

- **The CNN problem** [55]. In this problem we are given two servers in the euclidean plane, the one moving in the horizontal axis and the other in the vertical axis. At each time step a point (r_1, r_2) is requested, and in order to serve the request we should either move the horizontal server to point $x = r_1$ or the vertical server to $y = r_2$. This problem models the movement of the crew of a news network in Manhattan: whenever an event occurs, a camera should be either in the same street or in the same avenue.

Motivated by all those variants of the k-server problem, Koutsoupias and Taylor [55] introduced a substantial generalization of the k-server problem, called the *generalized k-server problem*[2]. 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. Similarly, the weighted k-server problem corresponds to the case when the metric spaces are scaled copies of each other, i.e. $M_i = w_i \cdot M$ for some fixed M, and the requests have the form $(r, \ldots, r)$. Finally, the CNN problem corresponds to the case where $k = 2$ and both M_1, M_2 are lines.

Previous Work. Despite the intense interest, this problem is poorly understood. Prior to our work, competitive algorithms were known only for $k = 2$ [65, 63]. Sitters [63] highlights that the existence of an $f(k)$-competitive algorithm is among the most intriguing and important open problems in online computation.

Our Contribution. In Chapter 4, we consider the generalized k-server problem on uniform metrics and obtain the first $f(k)$-competitive algorithms for general k, whose competitive ratios almost match the known lower bounds.

[2]In fact, Koutsoupias and Taylor called the problem "sum of k 1-server problems". The name generalized k-server was proposed by Sitters and Stougie [65].

Note on joint work. The results in Chapter 4 are based on joint work with Nikhil Bansal, Marek Eliáš and Jesper Nederlof, which appeared in the *29th ACM-SIAM Symposium on Discrete Algorithms* (SODA), 2018 [13].

Chapter 2

Tight Bounds for Double Coverage Against Weak Adversaries

2.1 Introduction

In this chapter we study the Double Coverage (DC) algorithm for the (h, k)-server problem. It is well-known that DC is k-competitive for $h = k$ in any tree metric. We prove that even if $k > h$ the competitive ratio of DC does not improve; in fact, it increases up to $h + 1$ as k grows. In particular, we show matching upper and lower bounds of $\frac{k(h+1)}{k+1}$ on the competitive ratio of DC in tree metrics.

Related Work. The k-server problem is a far reaching generalization of various online problems. The most well-studied of those is the paging (caching) problem, which corresponds to k-server problem on a uniform metric space. Sleator and Tarjan [66] gave several k-competitive algorithms for paging and showed that this is the best possible ratio for any deterministic algorithm.

Interestingly, the k-server problem does not seem to get harder in more general metrics. The celebrated *k-server conjecture* states that a k-competitive deterministic algorithm exists for every metric space. Koutsoupias and Papadimitriou [53] showed that the Work Function Algorithm (WFA) is $(2k - 1)$-competitive for every metric space, almost resolving the conjecture. The conjecture has been settled for several special metrics (an excellent reference is [19]). In particular for the line metric, Chrobak et al. [29] gave an elegant k-competitive algorithm called Double Coverage (DC). This algorithm was later extended and shown to be k-competitive for all tree metrics [30]. Additionally, in [17] it was shown that the WFA is k-competitive for some special metrics,

including the line.

The (h,k)-server problem: In the (h,k)-setting, the online algorithm has k servers, but its performance is compared to an offline optimal algorithm with $h \leq k$ servers. The (h,k)-server setting turns out to be much more intriguing and is much less understood than the standard k-server. For uniform metrics (the (h,k)-paging problem), $k/(k-h+1)$-competitive algorithms are known [66] and no deterministic algorithm can achieve a better ratio. Note that this guarantee equals k for $h=k$, and tends to 1 as the ratio k/h becomes arbitrarily large. In particular, for $k=2h$, this is smaller than 2. The same competitive ratio can also be achieved for the weighted caching problem [68], which is equivalent to the (h,k)-server problem on weighted stars (and even for the more general file caching problem [69], which is not a special case of the (h,k)-server problem).

It might seem natural to conjecture that, analogously to the k-server case, general metrics are no harder than the uniform metrics, and hence that $k/(k-h+1)$ is the right bound for the (h,k)-server problem in all metrics. However, surprisingly, Bar-Noy and Schieber (cf. [19, p. 175]) showed this to be false: In the line metric, for $h=2$, no deterministic algorithm can be better than 2-competitive, regardless of the value of k. This is the best known lower bound for the general (h,k)-server problem.

On the other hand, the best known upper bound is $2h$, even when $k/h \to \infty$. In particular, Koutsoupias [51] showed that the WFA with k servers is $2h$-competitive[1] against an offline optimum with h servers. Note that one way to achieve a guarantee of $2h-1$ is simply to disable the $k-h$ extra online servers and use WFA with h servers only. The interesting thing about the result of [51] is that the online algorithm does not know h and is $2h$-competitive simultaneously for every $h \leq k$. But, even if we ignore this issue of whether the online algorithm knows h or not, no guarantee better than h is known, even for very special metrics such as depth-2 HSTs or the line, and even when $k/h \to \infty$.

The DC algorithm: This situation motivates us to consider the (h,k)-server problem on the line and more generally on trees. In particular, we consider the Double Coverage (DC) algorithm [29] originally defined for a line, and its generalization to trees [30]. We call an algorithm's server s adjacent to the request r if there are no algorithm's servers on the unique path between the locations of r and s. Note that there may be multiple servers in one location, satisfying this requirement — in such case, one of them is chosen arbitrarily

[1] Actually, [51] gives a stronger bound: $\mathrm{WFA}_k \leq 2h \cdot \mathrm{OPT}_h - \mathrm{OPT}_k + O(1)$, where the algorithm's subscripts specify how many servers they use.

as the adjacent server for this location, and the others are considered non-adjacent.

DC-Line: If the current request r lies outside the convex hull of current locations of the servers, serve it with the nearest server. Otherwise, we move the two servers adjacent to r towards it at equal speed until some server reaches r.

DC-Tree: We move all the servers adjacent to r towards it at equal speed until some server reaches r. Note that the set of servers adjacent to r can change during the move, in particular when some server reaches a vertex of the tree. We call the parts of the move where the set of adjacent servers stays the same *elementary moves*.

There are several natural reasons to consider DC for the line and trees. For paging (and weighted paging), all known k-competitive algorithms also attain the optimal ratio for the (h, k) version. This suggests that a k-competitive algorithm for the k-server problem might attain the "right" ratio in the (h, k)-setting. DC is the only deterministic k-server algorithm known to be k-competitive for the line and trees. Moreover, DC obtains the optimum $k/(k-h+1)$-competitive ratio for the (h, k)-paging problem, as we show[2] in Section 2.5.

Our Results: We show that the exact competitive ratio of DC on lines and trees in the (h, k)-setting is $\frac{k(h+1)}{(k+1)}$.

Theorem 2.1. *The competitive ratio of DC is at least* $\frac{k(h+1)}{(k+1)}$*, even for a line.*

Note that for a fixed h, the competitive ratio worsens slightly as the number of online servers k increases. In particular, it equals h for $k = h$ and it approaches $h + 1$ as $k \to \infty$.

Consider the seemingly trivial case of $h = 1$. If $k = 1$, clearly DC is 1-competitive. However, for $k = 2$ it becomes $4/3$ competitive, as we now sketch. Consider the instance where all servers are at $x = 0$ initially. A request arrives at $x = 2$, upon which both DC and offline move a server there and pay 2. Then a request arrives at $x = 1$. DC moves both servers there and pays 2 while offline pays 1. All servers are now at $x = 1$, and the instance repeats.

Generalizing this example to $(1, k)$ already becomes quite involved. Our lower bound in Theorem 2.1 for general h and k is based on an adversarial strategy obtained by a careful recursive construction.

We also give a matching upper bound.

Theorem 2.2. *For any tree, the competitive ratio of DC is at most* $\frac{k(h+1)}{(k+1)}$.

[2]This was known implicitly since the work of Chrobak and Larmore [30], where it was pointed out that DC in star metrics is equivalent to Flush-When-Full (FWF) algorithm, and it is well-known that FWF attains the optimal competitive ratio. In Section 2.5 we give an explicit proof.

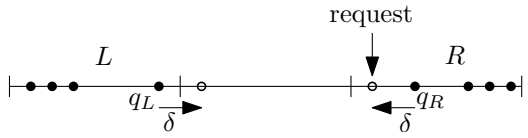


Figure 2.1: DC server is pulled to the right by δ

This generalizes the previous results for $h = k$ [29, 30]. Our proof also follows similar ideas, but our potential function is more involved (it has three terms instead of two), and the analysis is more subtle. To keep the main ideas clear, we first prove Theorem 2.2 for the simpler case of a line in Section 2.3. The proof for trees is analogous but more involved, and is described in Section 2.4.

2.2 Lower Bound

We now prove Theorem 2.1. We will describe an adversarial strategy S_k for the setting where DC has k servers and the offline optimum (adversary) has h servers, whose analysis establishes that the competitive ratio of DC is at least $k(h+1)/(k+1)$.

Roughly speaking (and ignoring some details), the strategy S_k works as follows. Let $I = [0, b_k]$ be the *working interval* associated with S_k. Let $L = [0, \epsilon b_k]$ and $R = [(1-\epsilon)b_k, b_k]$ denote the (tiny) *left front* and *right front* of I. Initially, all offline and online servers are located in L. The adversary moves all its h servers to R and starts requesting points in R, until DC eventually moves all its servers to R. The strategy inside R is defined recursively depending on the number of DC servers currently in R: if DC has i servers in R, the adversary executes the strategy S_i repeatedly inside R, until another DC server arrives there, at which point it switches to the strategy S_{i+1}. When all DC servers reach R, the adversary moves all its h servers back to L and repeats the symmetric version of the above instance until all servers move from R to L. This defines a *phase*. To show the desired lower bound, we recursively bound the online and offline costs during a phase of S_k in terms of costs incurred by strategies $S_1, S_2, \ldots, S_{k-1}$.

A crucial parameter of a strategy will be the *pull*. Recall that DC moves some server q_L closer to R if and only if q_L is the rightmost DC server outside R and a request is placed to the left of q_R, the leftmost DC server in R, as shown in Figure 2.1. In this situation q_R moves by δ to the left and q_L moves to the right by the same distance, and we say that the strategy in R exerts a *pull* of δ on q_L. We will be interested in the amount of pull exerted by a strategy during one phase.

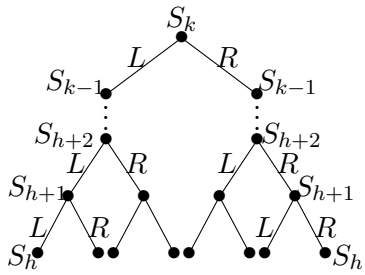


Figure 2.2: Representation of strategies and the areas that they define using a binary tree.

Formal description: We now give a formal definition of the instance. We begin by introducing the quantities (that we bound later) associated with each strategy S_i during a single phase:

- d_i, lower bound for the cost of DC inside the working interval.
- A_i, upper bound for the cost of the adversary.
- p_i, P_i, lower resp. upper bound for the "pull" exerted on any external DC servers located to the left of the working interval of S_i. Note that, as will be clear later, by symmetry the same pull is exerted to the right.

For $i \geq h$, the ratio $r_i = \frac{d_i}{A_i}$ is a lower bound for the competitive ratio of DC with i servers against an adversary with h servers.

We now define the right and left front precisely. Let $\varepsilon > 0$ be a sufficiently small constant. For $i \geq h$, we define the size of working intervals for strategy S_i as $s_h := h$ and $s_{i+1} := s_i/\varepsilon$. Note that $s_k = h/\varepsilon^{k-h}$. The working interval for strategy S_k is $[0, s_k]$, and inside it we have two working intervals for strategies S_{k-1}: $[0, s_{k-1}]$ and $[s_k - s_{k-1}, s_k]$. We continue this construction recursively and the nesting of these intervals creates a tree-like structure as shown in Figure 2.2. For $i \geq h$, the working intervals for strategy S_i are called type-i intervals. Strategies S_i, for $i \leq h$, are special and are executed in type-h intervals.

Strategies S_i for $i \leq h$: For $i \leq h$, strategies S_i are performed in a type-h interval (recall this has length h). Let Q be $h+1$ points in such an interval, with distance 1 between consecutive points.

There are two variants of S_i that we call $\overrightarrow{S_i}$ and $\overleftarrow{S_i}$. We describe $\overrightarrow{S_i}$ in detail, and the construction of $\overleftarrow{S_i}$ will be exactly symmetric. At the beginning of $\overrightarrow{S_i}$, we ensure that DC servers occupy the rightmost i points of Q and adversary servers occupy the rightmost h points of Q as shown in Figure 2.3.

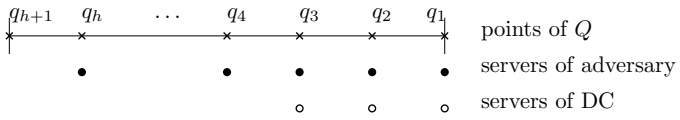


Figure 2.3: The initial position for Strategy $\overrightarrow{S_3}$ (for $h \geq 3$), in which the adversary requests q_4, q_3, q_2, q_1. DC's servers move for a total of 6, exerting a pull of 1 in the process, only to return to the same position. The adversary's cost is 0 if $h > 3$ and 2 if $h = 3$: in such case, the adversary serves both q_4 and q_3 with the server initially located in q_3.

The adversary requests the sequence $q_{i+1}, q_i, \ldots, q_1$. It is easily verified that DC incurs cost $d_i = 2i$, and its servers return to the initial position $q_i, \ldots, q_1$, so we can iterate $\overrightarrow{S_i}$ again. Moreover, a pull of $p_i = 1 = P_i$ is exerted in both directions.

For $i < h$, the adversary does not have to move at all, thus $A_i = 0$. For $i = h$, the offline can serve the sequence with cost $A_h = 2$, by using the server in q_h to serve request in q_{h+1} and then moving it back to q_h.

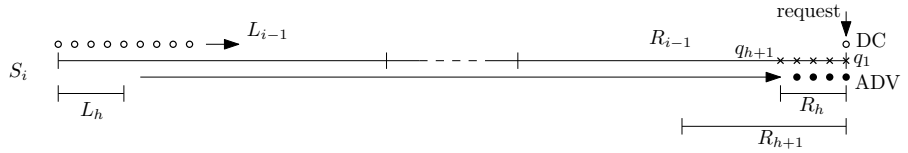
For strategy $\overleftarrow{S_i}$, we just number the points of Q in the opposite direction (q_1 will be leftmost and q_{h+1} rightmost). The request sequence, analysis, and assumptions about initial position are the same.

Strategies S_i for $i > h$: We define the strategy S_i for $i > h$, assuming that $S_1, \ldots, S_{i-1}$ are already defined. Let I denote the working interval for S_i. We assume that, initially, all offline and DC servers lie in the leftmost (or analogously rightmost) type-$(i-1)$ interval of I. Indeed, for S_k this is achieved by the initial configuration, and for $i < k$ we will ensure this condition before applying strategy S_i. In this case our phase consists of left-to-right step followed by right-to-left step (analogously, if all servers start in the rightmost interval, we apply first right-to-left step followed by left-to-right step to complete the phase).

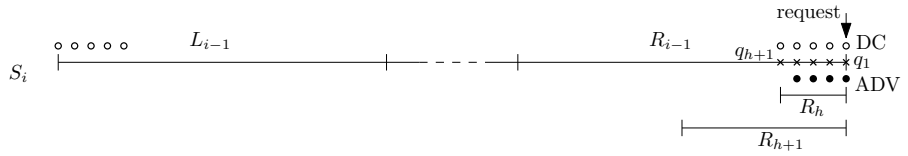
For each $h \leq j < i$, let L_j and R_j denote the leftmost and the rightmost type-j interval contained in I respectively.

Left-to-right step:

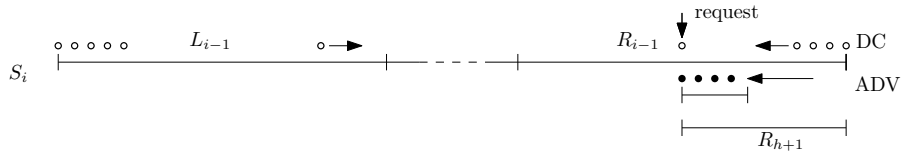
1. The adversary moves all its servers from L_{i-1} to R_h, specifically to the points $q_1, \ldots, q_h$ to prepare for the strategy $\overrightarrow{S_1}$. Next, point q_1 is requested, which forces DC to move one server to q_1, thus satisfying the initial conditions of $\overrightarrow{S_1}$. The figure below illustrates the servers' positions after these moves are performed.



2. For $j = 1$ to h: Apply $\overrightarrow{S}_j$ to interval R_h until the $(j+1)$-th server of DC arrives at the point q_{j+1} of R_h. Then, complete the request sequence $\overrightarrow{S}_j$, so that DC servers will reside in points $q_{j+1}, \ldots, q_1$, ready for strategy $\overrightarrow{S}_{j+1}$. The figure below illustrates the servers' positions after all those moves (i.e., the whole outer loop, for $j = 1 \ldots, h$) are performed.



3. For $j = h+1$ to $i-1$: keep applying S_j to interval R_j until the $(j+1)$-th server arrives in R_j. To clarify, S_j stands for either $\overrightarrow{S}_j$ or $\overleftarrow{S}_j$, depending on the locations of servers within R_j. In particular, the first S_j for any j is $\overleftarrow{S}_j$. Note that there is exactly one DC server in the working interval of S_i moving toward R_j from the left: the other servers in that working interval are either still in L_{i-1} or in R_j. Since R_j is the rightmost interval of R_{j+1} and $L_{i-1} \cap R_{j+1} = \emptyset$, the resulting configuration is ready for strategy $\overleftarrow{S}_{j+1}$. The figure below illustrates the very beginning of this sequence of moves, for $j = h+1$, right after the execution of the first step (of this three-step description) of $\overleftarrow{S}_{j+1}$.



Right-to-left step: Same as Left-to-right, just replace $\overrightarrow{S}_j$ by $\overleftarrow{S}_j$, R_j intervals by L_j, and L_j by R_j.

Bounding Costs: We begin with a simple but useful observation that follows directly from the definition of DC. For any subset X of $i \leq k$ consecutive DC servers, let us call *center of mass* of X the average position of servers in X. We call a request *external* with respect to X, when it is outside the convex hull of X and *internal* otherwise.

Lemma 2.3. *For any sequence of internal requests with respect to X, the center of mass of X remains the same.*

Proof. Follows trivially since for any internal request, DC moves precisely two servers towards it, by an equal amount in opposite directions. □

Let us derive bounds on d_i, A_i, p_i, and P_i in terms of these quantities for $j < i$. First, we claim that the cost A_i incurred by the adversary for strategy S_i during a phase can be upper bounded as follows:

$$A_i \leq 2\left(s_i h + \sum_{j=1}^{i-1} A_j \frac{s_i}{p_j}\right) = 2s_i\left(h + \sum_{j=h}^{i-1} \frac{A_j}{p_j}\right) \tag{2.1}$$

In the inequality above, we take the cost for left-to-right step multiplied by 2, since left-to-right and right-to-left step are symmetric. The term $s_i h$ is the cost incurred by the adversary in the beginning of the step, when moving all its servers from the left side of I to the right. The costs $A_j \frac{s_i}{p_j}$ are incurred during the phases of S_j for $j = 1, \ldots, i-1$, because A_j is an upper bound on the cost of the adversary during a phase of strategy S_j and $\frac{s_i}{p_j}$ is an upper bound on the number of phases of S_j during S_i. This follows because S_j (during left to right phase) executes as long as the $(j+1)$-th server moves from left of I to right of I. It travels a distance of at most s_i and receives a pull of p_j during each iteration of S_j in R. Finally, the equality in (2.1) follows, as $A_j = 0$ for $j < h$.

We now lower bound the cost of DC. Let us denote $\delta := (1 - 2\varepsilon)$. The length of $I \setminus (L_{i-1} \cup R_{i-1})$ is δs_i and all DC servers moving from right to left have to travel at least this distance. Furthermore, as $\frac{\delta s_j}{P_j}$ is a lower bound for the number of iterations of strategy S_j, we obtain:

$$d_i \geq 2\left(\delta s_i i + \sum_{j=1}^{i-1} d_j \frac{\delta s_i}{P_j}\right) = 2\delta s_i\left(i + \sum_{j=1}^{i-1} \frac{d_j}{P_j}\right) \tag{2.2}$$

It remains to show the upper and lower bounds on the pull P_i and p_i exerted on external servers due to the (right-to-left step of) strategy S_i. Suppose S_i is being executed in interval I. Let x denote the closest DC server strictly to the left of I. Let X denote the set containing x and all DC servers located in I. During the right-to-left step of S_i, all requests are internal with respect to X. So by Lemma 2.3, the center of the mass of X remains unchanged. As i servers moved from right to left during right-to-left step of S_i, this implies that x should have been pulled to the left by the same total amount, which is at least $i\delta s_i$ and at most is_i. Hence,

$$P_i := is_i \qquad\qquad p_i := i\delta s_i \tag{2.3}$$

Due to a symmetric argument, during the left-to-right step, the same amount of pull is exerted to the right.

Now we are ready to prove Theorem 2.1.

Proof of Theorem 2.1. The proof is by induction. In particular, we will show that the following holds for each $i \in [h, k]$:

$$\frac{d_i}{P_i} \geq 2i\delta^{i-h} \qquad \text{and} \qquad \frac{A_i}{p_i} \leq \frac{2(i+1)}{h+1}\delta^{-(i-h)} \tag{2.4}$$

Setting $i = k$, this implies the theorem as the competitive ratio r_k of DC satisfies

$$r_k \geq \frac{d_k}{A_k} \geq \frac{d_k/P_k}{A_k/p_k} \geq \frac{2k}{\frac{2(k+1)}{h+1}} \frac{\delta^{k-h}}{\delta^{-(k-h)}} = \frac{k(h+1)}{k+1}\delta^{2(k-h)} \ .$$

Therefore, as $\delta = (1-2\varepsilon)$, it is easy to see that $r_k \to \frac{k(h+1)}{k+1}$ when $\varepsilon \to 0$:

Induction base ($i = h$): For the base case we have $a_h = 2$, $d_h = 2h$, and $p_h = P_h = 1$, so $\frac{d_h}{P_h} = 2h$ and $\frac{A_h}{p_h} = 2$, i.e., (2.4) holds.

Induction step ($i > h$): Using (2.2), (2.3), and induction hypothesis, we obtain

$$\frac{d_i}{P_i} \geq \frac{2\delta}{i}\left(i + \sum_{j=1}^{i-1}\frac{d_j}{P_j}\right) \geq \frac{2\delta}{i}\left(i + \sum_{j=1}^{i-1} 2j\delta^{j-h}\right) \geq \frac{2\delta}{i}\delta^{i-1-h}(i + i(i-1)) = 2i\delta^{i-h} \ ,$$

where the last inequality follows from the fact that $\sum_{j=1}^{i-1} 2j = i(i-1)$. Similarly, we prove the second part of (2.4). The first inequality follows from (2.1) and (2.3), the second from the induction hypothesis:

$$\begin{aligned}
\frac{A_i}{p_i} &\leq \frac{2}{i\delta}\left(h + \sum_{j=h}^{i-1}\frac{A_j}{p_j}\right) \leq \frac{2}{i\delta}\left(h + \sum_{j=h}^{i-1}\frac{2(j+1)}{h+1}\delta^{-(j-h)}\right) \\
&\leq \frac{2}{i\delta}\delta^{-(i-1-h)}\left(\frac{h(h+1) + 2\sum_{j=h}^{i-1}(j+1)}{h+1}\right) \\
&\leq \frac{2}{i\delta^{i-h}}\frac{i(i+1)}{h+1} = \frac{2(i+1)}{h+1}\delta^{-(i-h)} \ ,
\end{aligned}$$

The last inequality follows from $2\sum_{j=h}^{i-1}(j+1) = i(i+1) - h(h+1)$. □

2.3 Upper Bound

In this section, we give an upper bound on the competitive ratio of DC that matches the lower bound from the previous section.

We begin by introducing some notation. We denote the optimal offline algorithm by OPT. Let r be a request issued at time t. Let X denote the configuration of DC (i.e. the multiset of points in the line where DC servers are located) and Y the configuration of OPT before serving request r. Similarly, let X' and Y' be the corresponding configurations after serving r.

In order to prove our upper bound, we will define a potential function $\Phi(X, Y)$ such that

$$DC(t) + \Phi(X', Y') - \Phi(X, Y) \leq c \cdot OPT(t), \tag{2.5}$$

where $c = \frac{k(h+1)}{k+1}$ is the desired competitive ratio, and $DC(t)$ and $OPT(t)$ denote the cost incurred by DC and OPT at time t. Coming up with a potential function Φ that satisfies (2.5) is sufficient, as c-competitiveness follows from summing this inequality over time.

For a set of points A, let D_A denote the sum of all $\binom{|A|}{2}$ pairwise distances between points in A. Let $M \subseteq X$ be some fixed set of h servers of DC and $\mathcal{M}(M, Y)$ denote the minimum weight perfect matching between M and Y, where the weights are determined by the distances. Abusing the notation slightly, we will denote by $\mathcal{M}(M, Y)$ both the matching and its cost. We denote

$$\Psi_M(X, Y) := \frac{k(h+1)}{k+1} \cdot \mathcal{M}(M, Y) + \frac{k}{k+1} \cdot D_M \ .$$

Then the potential function is defined as follows:

$$\begin{aligned} \Phi(X, Y) &= \min_M \Psi_M(X, Y) + \frac{1}{k+1} \cdot D_X \\ &= \min_M \left(\frac{k(h+1)}{k+1} \cdot \mathcal{M}(M, Y) + \frac{k}{k+1} \cdot D_M \right) + \frac{1}{k+1} \cdot D_X \ . \end{aligned}$$

Note that this generalizes the potential considered in [29, 30] for the case of $h = k$. In that setting, all the online servers are matched and hence $D_M = D_X$ and is independent of M, and thus the potential above becomes k times that minimum cost matching between X and Y plus D_x. On the other hand in our setting, we need to select the right set M of DC servers to be matched to the offline servers based on minimizing $\Psi_M(X, Y)$.

Let us first give a useful property concerning minimizers of Ψ, which will be crucial later in our analysis. Note that $\Psi_M(X, Y)$ is not simply the best matching between X and Y, but also includes the term D_M which makes the argument slightly subtle.

Lemma 2.4. *Let X and Y be the configurations of* DC *and* OPT *and consider some fixed offline server at location $y \in Y$. There exists a minimizer M of Ψ that contains some DC server x which is adjacent to y. Moreover, there is a minimum cost matching $\mathcal{M}$ between M and Y that matches x to y.*

We remark that the statement does not necessarily hold simultaneously for every offline server, but only for a single fixed offline server y. Moreover, we note that the adjacency in the lemma statement and the proof is defined as for the DC algorithm (cf. Section 2.1); specifically, as if there was a request at y's position.

Proof of Lemma 2.4. Let M' be some minimizer of $\Psi_M(X, Y)$ and $\mathcal{M}'$ be some associated minimum cost matching between M' and Y. Let x' denote the online server currently matched to y in $\mathcal{M}'$ and suppose that x' is not adjacent to y. Let x denote the server in X adjacent to y on the path from y to x'.

We will show that we can always modify the matching (and M') without increasing the cost of Φ, so that y is matched to x. We consider two cases depending on whether x is matched or unmatched.

1. If $x \in M'$: Let y' denote the offline server which is matched to x in $\mathcal{M}'$. To create new matching $\mathcal{M}$, we swap the edges and match x to y and x' to y'. The cost of the edge connecting y in the matching reduces by exactly $d(x', y) - d(x, y) = d(x', x)$. On the other hand, the cost of the matching edge for y' increases by $d(x', y') - d(x, y') \leq d(x, x')$, due to triangle inequality. Thus, the new matching has no larger cost. Moreover, the set of matched servers does not change, i.e., $M = M'$, and hence $D_M = D_{M'}$, which implies that $\Psi_M(X, Y) \leq \Psi_{M'}(X, Y)$.

2. If $x \notin M'$: In this case, we set $M = M' \setminus \{x'\} \cup \{x\}$ and we form $\mathcal{M}$, where y is matched to x and all other offline servers are matched to the same server as in $\mathcal{M}'$. Now, the cost of the matching reduces by $d(x', y) - d(x, y) = d(x, x')$. Moreover, $D_M \leq D_{M'} + (h-1) \cdot d(x, x')$, as the distance of each server in $M' \setminus \{x'\}$ to x can be greater than the distance to x' by at most $d(x, x')$. This gives

$$\begin{aligned}\Psi_M(X, Y) - \Psi_{M'}(X, Y) &\leq -\frac{(h+1)k}{k+1} \cdot d(x, x') + \frac{k(h-1)}{k+1} \cdot d(x, x') \\ &= -\frac{2k}{k+1} \cdot d(x, x') < 0 \ ,\end{aligned}$$

   and hence $\Psi_M(X, Y)$ is strictly smaller than $\Psi_{M'}(X, Y)$. □

We are now ready to prove Theorem 2.2 for the line.

Proof. Recall, that we are at time t and request r is arriving. We divide the analysis into two steps: (i) OPT serves r, and then (ii) DC serves r. As a consequence, whenever a server of DC serves r, we can assume that a server of OPT is already there.

For all following steps considered, M is the minimizer of $\Psi_M(X,Y)$ in the beginning of the step. It might happen that, after change of X, Y during the step, a better minimizer can be found. However, an upper bound for $\Delta\Psi_M(X,Y)$ is sufficient to bound the change in the first term of the potential function.

OPT moves: If OPT moves one of its servers by distance d to serve r, the value of $\Psi_M(X,Y)$ increases by at most $\frac{k(h+1)}{k+1}d$. As $OPT(t) = d$ and X does not change, it follows that

$$\Delta\Phi(X,Y) \leq \frac{k(h+1)}{k+1} \cdot OPT(t) \ ,$$

and hence (2.5) holds. We now consider the second step, when DC moves.

DC moves: We consider two cases depending on whether DC moves a single server or two servers.

1. Suppose DC moves its rightmost server (the leftmost server case is identical) by distance d. Let y denote the offline server at r. By Lemma 2.4 we can assume that y is matched to the rightmost server of DC. Thus, the cost of the minimum cost matching between M and Y decreases by d. Moreover, D_M increases by exactly $(h-1)d$ (as the distance to rightmost server increases by d for all servers of DC). Thus, $\Psi_M(X,Y)$ changes by

$$-\frac{k(h+1)}{k+1} \cdot d + \frac{k(h-1)}{k+1} \cdot d = -\frac{2k}{k+1} \cdot d \ .$$

   Similarly, D_X increases by exactly $(k-1)d$. This gives us that

$$\Delta\Phi(X,Y) \leq -\frac{2k}{k+1} \cdot d + \frac{k-1}{k+1} \cdot d = -d \ .$$

   As $DC(t) = d$, this implies that (2.5) holds.

2. We now consider the case when DC moves 2 servers x and x', each by distance d. Let y denote the offline server at the request r. By Lemma 2.4 applied to y, we can assume that M contains at least one of x or x', and that y is matched to one of them (say x) in some minimum cost matching $\mathcal{M}$ of M to Y.

We note that D_X decreases by precisely $2d$. In particular, the distance between x and x' decreases by $2d$, and for any other server of $X \setminus \{x, x'\}$ its total distance to other servers does not change. Moreover, $DC(t) = 2d$. Hence, to prove (2.5), it suffices to show

$$\Delta\Psi_M(X, Y) \leq -\frac{k}{k+1} \cdot 2d \ . \tag{2.6}$$

To this end, we consider two sub-cases.

(a) *Both x and x' are matched:* In this case, the cost of the matching $\mathcal{M}$ does not increase as the cost of the matching edge (x, y) decreases by d and the move of x' can increase the cost of the matching by at most d. Moreover, D_M decreases by precisely $2d$ (due to x and x' moving closer). Thus, $\Delta\Psi_M(X, Y) \leq -\frac{k}{k+1} \cdot 2d$, and hence (2.6) holds.

(b) *Only x is matched (to y) and x' is unmatched:* In this case, the cost of the matching $\mathcal{M}$ decreases by d. Moreover, D_M can increase by at most $(h-1)d$, as x can move away from each server in $M \setminus \{x\}$ by distance at most d. So

$$\Delta\Psi_M(X, Y) \leq -\frac{(h+1)k}{k+1} \cdot d + \frac{k(h-1)}{k+1} \cdot d = -\frac{2k}{k+1} \cdot d \ ,$$

i.e., (2.6) holds. □

2.4 Extension to Trees

We now consider tree metrics. Specifically, we prove Theorem 2.2.

Part of the analysis carries over from the previous section. Observe that Lemma 2.4 holds for trees: we only used the triangle inequality and the fact that there exists a unique path between any two points. The main difference in the proof is that the set of servers adjacent to the request can now have arbitrary size (i.e., it no longer contains at most two servers) and that it can change as the move is executed, see Figure 2.4. To cope with this, we analyze elementary moves, as did Chrobak and Larmore [30]. Recall that an elementary move is a part when the set of servers adjacent to the request remains fixed.

Proof of Theorem 2.2. We use the same potential as before, i.e, we let

$$\Psi_M(X, Y) := \frac{k(h+1)}{k+1} \cdot \mathcal{M}(M, Y) + \frac{k}{k+1} \cdot D_M \ ,$$

and define

$$\Phi(X, Y) = \min_M \Psi_M(X, Y) + \frac{1}{k+1} \cdot D_X \ .$$

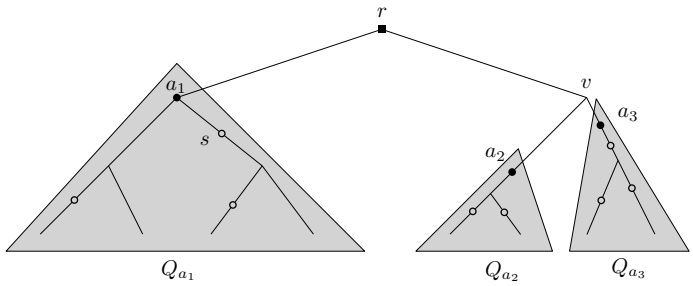


Figure 2.4: Beginning of elementary move: server a_1 just covered server s removing him from the set of servers adjacent to request r. Servers a_1, a_2, and a_3 will move towards r, until a_3 reaches subroot v removing a_2 from the list of adjacent servers and completing thereby this elementary move.

To prove the theorem, we show that for any time t the following holds:

$$DC(t) + \Phi(X', Y') - \Phi(X, Y) \leq c \cdot OPT(t), \tag{2.7}$$

where $c = \frac{k(h+1)}{k+1}$.

As in the analysis for the line, we split the analysis in two parts: (i) OPT serves r, and then (ii) DC serves r. As a consequence, whenever a server of DC serves r, we can assume that a server of OPT is already there.

OPT moves: If OPT moves a server by distance d, only the matching cost is affected in the potential function, and it can increase by at most $d \cdot k(h+1)/(k+1)$. Therefore

$$\Delta\Phi(X, Y) \leq \frac{k(h+1)}{k+1} \cdot OPT(t) \ ,$$

and hence (2.7) holds.

DC moves: Instead of focusing on the whole move done by DC to serve request r, we prove that (2.7) holds for each elementary move.

Consider an elementary move where q servers are moving by distance d. Let A denote the set of those active servers. Clearly, $|A| = q$. Let also M be a minimizer of $\Psi_M(X, Y)$ at the beginning of the step. Let us imagine for now, that the requested point r is the root of the whole tree. For $a \in A$ let Q_a denote the set of DC servers in the subtree rooted at a's location, including a, see Figure 2.4. We set $q_a := |Q_a|$ and $h_a := |Q_a \cap M|$. Finally, let $A_M := A \cap M$.

By Lemma 2.4, we can assume that one of the servers in A is matched to the offline server in r. Thus the move of this server decreases the cost of

$\mathcal{M}(M,Y)$ by d. The move of all other servers of A_M can increase the cost of $\mathcal{M}(M,Y)$ by at most $(|A_M|-1)\cdot d$. We get that

$$\Delta\mathcal{M}(M,Y) \le (|A_M|-2)\cdot d \ .$$

In order to calculate the change in D_X and D_M, it is convenient to consider the moves of active servers sequentially rather than simultaneously.

We start with D_X. Clearly, each $a \in A$ moves further away from $q_a - 1$ servers in X by distance d and gets closer to the remaining $k - q_a$ ones by the same distance. Thus, the change of D_X associated with a is $(q_a - 1 - (k - q_a))d = (2q_a - k - 1)d$. Therefore we have

$$\Delta D_X = \sum_{a\in A}(2q_a - k - 1)d = (2k - q(k+1))\, d \ ,$$

as $\sum_{a\in A} q_a = k$.

Similarly, for D_M, we first note that it can change only due to moves of servers in A_M. Specifically, each $a \in A_M$ moves further away from $h_a - 1$ servers in M and gets closer to the remaining $h - h_a$ of them. Thus, the change of D_M associated with a is $(2h_a - h - 1)d$. Therefore we have

$$\Delta D_M = \sum_{a\in A_M}(2h_a - h - 1)d \le (2h - |A_M|(h+1))\, d \ ,$$

since $\sum_{a\in A_M} h_a \le \sum_{a\in A} h_a = h$.

Using above inequalities, we see that the change of potential is at most

$$\begin{aligned}
&\frac{k(h+1)d}{k+1}(|A_M|-2) + \frac{k\cdot d}{k+1}(2h - |A_M|(h+1)) + \frac{d}{k+1}(2k - q(k+1)) \\
&\le \frac{d}{k+1}\big(k(h+1)(|A_M|-2) + k\,(2h - |A_M|(h+1)) + 2k - q(k+1)\big) \\
&= \frac{d}{k+1}(-q(k+1)) = -q\cdot d \ ,
\end{aligned}$$

since

$$\begin{aligned}
&k(h+1)(|A_M|-2) + k(2h - |A_M|(h+1)) + 2k \\
&= -2k(h+1) + k(h+1)|A_M| - |A_M|k(h+1) + 2kh + 2k \\
&= -2k(h+1) + 2k(h+1) = 0
\end{aligned}$$

Thus, (2.7) holds, as $DC(t) = q\cdot d$. □

2.5 Competitive Ratio of DC for the Paging Problem

The paging problem is the special case of the k-server on a star graph, where all edges have weight $\frac{1}{2}$ and all requests appear at the leaves. It known that (h,k)-paging has a deterministic competitive ratio of $\frac{k}{k-h+1}$. However, we are not aware of any explicit proof showing that the DC algorithm also achieves this ratio. We give such a proof using a potential function.

Let X and Y denote the configurations of DC and OPT respectively. Note that any server of DC can only be at the root or at a leaf, and servers of OPT can only be at leaves.

We define

$$\Phi(t) = \frac{-k-h+1}{2(k-h+1)}\ell + \frac{k}{k-h+1}|Y \setminus X|$$

Where ℓ is the number of DC servers at the root.

Analysis: As usual, we consider separately moves of DC and OPT. We assume that, whenever a point is requested, first OPT moves a server there and then DC moves its servers.

Offline moves: When optimal moves any single server from one leaf to another it pays 1. The first term of the potential is not affected while the second can increase by at most one. We get that $\Delta\Phi \leq \frac{k}{k-h+1} = \frac{k}{k-h+1} \cdot \mathrm{OPT}$.

DC moves: Let us now consider moves of DC. We distinguish between two cases depending on whether it moves one or more servers.

- $\ell > 0$: In this case, DC moves one server from the root to the requested leaf, so DC pays $1/2$. The number of servers at the root ℓ decreases by 1 and the second term decreases by 1. We get

  $$\Delta\Phi = \frac{k+h-1}{2(k-h+1)} - \frac{k}{k-h+1} = \frac{-k+h-1}{2(k-h+1)} = -\frac{1}{2}$$

  and hence $\mathrm{DC} + \Delta\Phi = 0$.

- $\ell = 0$: In the case, DC moves all the servers from the leaves toward the root (and then we go to the case above). In that case DC occurs a cost of $k/2$. Let us call a the number of online servers that coincide with servers of OPT before the move of DC. Then ℓ is increasing by k while $|Y \setminus X|$ increases by a. We get that

  $$\Delta\Phi = \frac{-k-h+1}{2(k-h+1)}k + \frac{k}{k-h+1}a$$

  Observe that $a \leq h-1$, as there is an OPT at the current request that was not covered when DC started moving. Thus we can upper bound $\Delta\Phi$ as:

$$\Delta\Phi \leq \frac{-k-h+1+2(h-1)}{2(k-h+1)}k = \frac{-k+h-1}{2(k-h+1)}k - \frac{k-h+1}{2(k-h+1)}k = -\frac{k}{2}.$$

Overall we get that $\mathrm{DC} + \Delta\Phi \leq \frac{k}{2} - \frac{k}{2} = 0$.

Chapter 3

The (h, k)-Server Problem on Bounded Depth Trees

3.1 Introduction

In this chapter, we consider the (h, k)-server problem on special tree metrics, in particular trees of bounded depth. Our motivation comes from the fact that for (weighted) star graphs (trees of depth 1) we have a very good understanding of the problem, while for more complex metrics like the line and arbitrary trees, no $o(h)$-competitive algorithm is known. Moreover, as we mentioned in the previous chapter, for trees of depth 1, all algorithms that are k-competitive for $k = h$, attain also the optimum competitive ratio for the (h, k) setting. In contrast, in the line we showed that the Double Coverage algorithm, which attains the optimal competitive ratio for $k = h$, does not perform well in the (h, k)-setting. Thus, it is natural to consider trees of depth 2, which are the simplest possible extension of the weighted star, and try to get a better understanding on the effect of the structure of the metric space on the competitive ratio of the (h, k)-server problem.

We first show that already in trees of small depth, all the previously known algorithms (beyond uniform metrics), specifically the Double Coverage (DC) algorithm and the Work Function Algorithm (WFA) have competitive ratio $\Omega(h)$. This suggests that we need substantially new ideas in order to deal with the (h, k)-server problem.

Theorem 3.1. *The competitive ratio of DC in depth-2 trees is $\Omega(h)$, even when $k/h \to \infty$.*

In particular, DC is unable to use the extra servers in a useful way. For the WFA, we present the following lower bound.

Theorem 3.2. *The competitive ratio of the WFA is at least* $h + 1/3$ *in a depth-*3 *tree for* $k = 2h$.

This lower bound can also be extended to the line metric. Note that for the line it is known that the WFA is h-competitive for $k = h$ [17], while our lower bound for $k = 2h$ is strictly larger than h. In other words, our result shows that there exist sequences where the WFA performs strictly worse with $2h$ servers than using h servers! A similar lower bound was shown in the previous chapter for the Double Coverage algorithm. Interestingly, our lower bound exactly matches the upper bound $(h + 1)\mathrm{OPT}_h - \mathrm{OPT}_k$ implied by results of [51, 17] for the WFA in the line. We describe the details in Section 3.6.

Our main result is the first $o(h)$-competitive algorithm for depth-d trees with the following guarantee.

Theorem 3.3. *There is an algorithm that is* $O_d(1)$*-competitive on any depth-*d *tree, whenever* $k = \delta h$ *for* $\delta > 1$. *More precisely, its competitive ratio is* $O(d \cdot 2^d)$ *for* $\delta \in [2^d, +\infty)$, *and* $O(d \cdot (\frac{\delta^{1/d}}{\delta^{1/d}-1})^d)$ *for* $\delta \in (1, 2^d)$. *If* δ *is very small, i.e.* $\delta = 1 + \epsilon$ *for* $0 < \epsilon \leq 1$, *the latter bound becomes* $O(d \cdot (2d/\epsilon)^d)$.

The algorithm is designed to overcome the drawbacks of DC and WFA, and can be viewed as a more aggressive and cost-sensitive version of DC. It moves the servers more aggressively at non-uniform speeds towards the region of the current request, giving a higher speed to a server located in a region containing many servers. It does not require the knowledge of h, and is simultaneously competitive against all h strictly smaller than k.

Finally, we give an improved general lower bound. Bar-Noy and Schieber (cf. [19, p. 175]) showed that there is no better than 2-competitive algorithm for the (h, k)-server problem in general metrics, by constructing their lower bound in the line metric. Our next result shows that even a 2-competitive algorithm is not possible. In particular, we present a construction in a depth-2 HST showing that no 2.4-competitive algorithm is possible.

Theorem 3.4. *There is no* 2.4*-competitive deterministic algorithm for trees of depth 2, even when* $k/h \to \infty$, *provided that* h *is larger than some constant independent of* k.

This shows that depth-2 trees are qualitatively quite different from depth-1 trees (same as weighted star graphs) which allow a ratio $k/(k - h + 1)$. We have not tried to optimize the constant 2.4 above, but computer experiments suggest that the bound can be improved to about 2.88.

Preliminaries

A *depth-d tree* is an edge-weighted rooted tree with each leaf at depth exactly d. In the (h, k)-server problem in a depth-d tree, the requests arrive only at leaves,

and the distance between two leaves is defined as the distance in the underlying tree. A depth-d *hierarchically separated tree* (HST) is a depth-d tree with the additional property that the distances decrease geometrically away from the root (see e.g. [15]). We will first present our algorithm for general depth-d trees (without the HST requirement), and later show how to (easily) extend it to arbitrary trees with bounded diameter, where requests are also allowed in the internal nodes.

Work Function Algorithm. Consider a request sequence $\sigma = r_1, r_2, \ldots, r_m$. For each $i = 1, \ldots, m$, let $w_i(A)$ denote the optimal cost to serve requests $r_1, r_2, \ldots, r_i$ and end up in the configuration A, which is specified by the set of locations of the servers. The function w_i is called *work function*. The Work Function Algorithm (WFA) decides its moves depending on the values of the work function. Specifically, if the algorithm is in a configuration A and a request $r_i \notin A$ arrives, it moves to a configuration X such that $r_i \in X$ and $w_i(X) + d(A, X)$ is minimized. For more background on the WFA, see [19].

Organization

In Section 3.2, we describe the lower bound for the DC in depth-2 HSTs. The shortcomings of the DC might help the reader to understand the motivation behind the design of our algorithm for depth-d trees, which we describe in Section 3.3. Its extension to the bounded-diameter trees is discussed in Section 3.4. In Section 3.5 we describe the general lower bound (Theorem 3.4) and the lower bound for the WFA (Theorem 3.2) for depth-3 HSTs. The extension of Theorem 3.2 to the line is discussed in Section 3.6.

3.2 Lower Bound for Double Coverage on Depth-2 HSTs

We now show a lower bound of $\Omega(h)$ on the competitive ratio of the DC algorithm.

Let T be a depth-2 HST with $k+1$ subtrees and edge lengths chosen as follows. Edges from the root r to its children have length $1-\epsilon$, and edges from the leaves to their parents length ϵ for some $\epsilon \ll 1$. Let T_u be a subtree rooted at an internal node $u \neq r$. A *branch* B_u is defined as T_u together with the edge e connecting T_u to the root. We call B_u *empty*, if there is no online server in T_u nor in the interior of e. Since T contains $k+1$ branches, at least one of them is always empty.

The idea behind the lower bound is quite simple. The adversary moves all its h servers to the leaves of an empty branch B_u, and keeps requesting those leaves until DC brings h servers to T_u. Then, another branch has to become

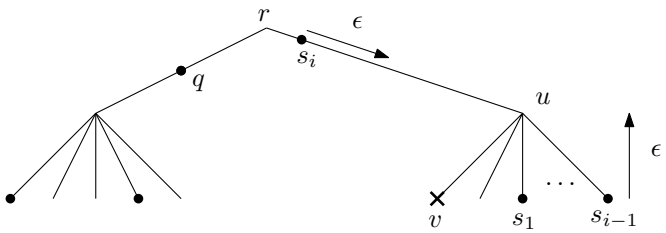


Figure 3.1: Move of DC during Step i. Servers $s_1, \ldots, s_{i-1}$ are moving towards u by distance ϵ and s_i is moving down the edge (r, u) by the same distance. While s_i is in the interior of (r, u), no server q from some other branch is adjacent to v because the unique path between v and q passes through s_i.

empty, and the adversary moves all its servers there, starting a new *phase*. The adversary can execute an arbitrary number of such phases.

The key observation is that DC is "too slow" when bringing new servers to T_u, and incurs a cost of order $\Omega(h^2)$ during each phase, while the adversary only pays $O(h)$.

Theorem 3.1. *The competitive ratio of* DC *in depth-2 HSTs is* $\Omega(h)$*, even when* $k/h \to \infty$*.*

Proof. We describe a phase, which can be repeated arbitrarily many times. The adversary places all its h servers at different leaves of an empty branch B_u and does not move until the end of the phase. At each time during the phase, a request arrives at such a leaf, which is occupied by some offline server, but contains no online servers. The phase ends at the moment when the hth server of DC arrives to T_u.

Let ALG denote the cost of the DC algorithm and ADV the cost of the adversary during the phase. Clearly, ADV $= 2h$ in each phase: The adversary moves its h servers to T_u and does not incur any additional cost until the end of the phase. However, we claim that ALG $= \Omega(h^2)$, no matter where exactly the DC servers are located when the phase starts. To see that, let us call Step i the part of the phase when DC has exactly $i - 1$ servers in T_u. Clearly, Step 1 consists of only a single request, which causes DC to bring one server to the requested leaf. So the cost of DC for Step 1 is at least 1. To bound the cost in the subsequent steps, we make the following observation.

Observation 3.5. *At the moment when a new server s enters the subtree T_u, no other DC servers are located along the edge $e = (r, u)$.*

This follows from the construction of DC, which moves only servers adjacent to the request. At the moment when s enters the edge e, no other server above s can be inside e; see Figure 3.1.

We now focus on Step i for $2 \leq i \leq h$. There are already $i-1$ servers in T_u, and let s_i be the next one which is to arrive to T_u.

Crucially, s_i moves if and only if all the servers of DC in the subtree T_u move from the leaves towards u, like in Figure 3.1: When the request arrives at v, s_i moves by ϵ and the servers inside T_u pay together $(i-1)\epsilon$. However, such a moment does not occur, until all servers $s_1, \ldots, s_{i-1}$ are again at leaves, i.e. they incur an additional cost $(i-1)\epsilon$. To sum up, while s_i moves by ϵ, the servers inside T_u incur cost $2(i-1)\epsilon$.

When Step i starts, the distance of s_i from u is at least $1-\epsilon$, and therefore s_i moves by distance ϵ at least $\lfloor \frac{1-\epsilon}{\epsilon} \rfloor$ times, before it enters T_u. So, during Step i, DC pays at least

$$\left\lfloor \frac{1-\epsilon}{\epsilon} \right\rfloor (2(i-1)\epsilon + \epsilon) \geq \frac{1-2\epsilon}{\epsilon} \cdot \epsilon\big(2(i-1)+1\big) = (1-2\epsilon)(2i-1)$$

By summing over all steps $i = 1, \ldots, h$ and choosing $\epsilon \leq 1/4$, we get

$$\mathrm{ALG} \geq 1 + \sum_{i=2}^{h} (1-2\epsilon)(2i-1) \geq \sum_{i=1}^{h} (1-2\epsilon)(2i-1) = (1-2\epsilon)h^2 \geq \frac{h^2}{2}$$

To conclude the proof, we note that $\mathrm{ALG} \,/\, \mathrm{ADV} \geq (h^2/2)/(2h) = h/4 = \Omega(h)$ for all phases. □

3.3 Algorithm for Depth-d Trees

In this section we prove Theorem 3.3.

Recall that a depth-d tree is a rooted-tree and we allow the requests to appear only at the leaves. However, to simplify the algorithm description, we will allow the online servers to reside at any node or at any location on an edge (similar to that in DC). To serve a request at a leaf v, the algorithm moves all the adjacent servers towards v, where the speed of each server coming from a different branch of the tree depends on the number of the online servers "behind" it.

To describe the algorithm formally, we state the following definitions. Let T be a depth-d tree. For a point $x \in T$ (either a node or a location on some edge), we define T_x as the subtree consisting of all points below x including x itself, and we denote k_x the number of the online servers inside T_x. If s is a server located at a point x, we denote $T_s = T_x$ and $k_s = k_x$. We also denote $T_x^- = T_x \setminus \{x\}$, and k_x^- the corresponding number of the algorithm's servers in T_x^-.

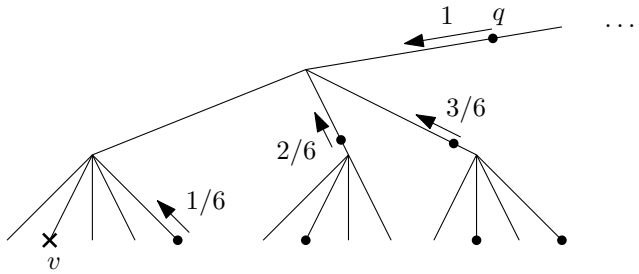


Figure 3.2: A request at v, and Phase 2 of Algorithm 1. Note that k_q^- equals 6 in the visualised case. Speed is noted next to each server moving.

3.3.1 Algorithm Description

Suppose that a request arrives at a leaf v; let A be the set of algorithm's servers adjacent to v. The algorithm proceeds in two phases, depending on whether there is a server along the path from v to the root r or not. We set speeds as described in Algorithm 1 below and move the servers towards v either until the phase ends or the set A changes. This defines the elementary moves (where A stays unchanged). Note that if there are some servers in the path between v and the root r, only the lowest of them belongs to A. Figure 3.2 shows the progress of Phase 2.

Algorithm 1: Serving request at leaf v.

Phase 1: While there is no server along the path $r - v$
For each $s \in A$: move s at speed k_s/k

Phase 2: While no server reached v; Server $q \in A$ moves down along the path $r - v$
For server q: move it at speed 1
For each $s \in A \setminus \{q\}$: move it at speed k_s/k_q^-

We note two properties, the first of which follows directly from the design of Algorithm 1.

Observation 3.6. *No edge $e \in T$ contains more than one server of Algorithm 1 in its interior.*

Note that during both the phases, the various servers move at non-uniform speeds depending on their respective k_s. The following observations about these speeds will be useful.

Observation 3.7. *During Phase 1, the total speed of servers is 1. This follows as $\sum_{s \in A} k_s = k$. Analogously, during Phase 2, the total speed of servers inside T_q^- is 1, if there are any. This follows as $\sum_{s \in A \setminus \{q\}} k_s = k_q^-$.*

The intuition behind the algorithm is the following. Recall that the problem with DC is that it moves its servers too slowly towards an active region when requests start arriving there. In contrast, we change the speeds of the servers adjacent to v to make the algorithm more aggressive. From each region, an adjacent server moves at speed proportional to the number of the online servers in that region. This way, if a region has many servers and not many requests appear there (we call such regions excessive), servers move quickly from there to a more active region. Moreover, in Phase 2, server q is viewed as a *helper* coming to aid the servers inside T_q^-. The second main idea is to keep the total speed of the helper proportional to the total speed of the servers inside T_q^-. This prevents the algorithm from becoming overly aggressive and keeps the cost of the moving helpers comparable to the cost incurred within T_q^-.

3.3.2 Analysis

We will analyze the algorithm based on a suitable potential function $\Phi(t)$. Let $\mathrm{ALG}(t)$ and $\mathrm{OPT}(t)$ denote the cost of the algorithm and of the adversary respectively, for serving the request at time t. Let $\Delta_t\Phi = \Phi(t) - \Phi(t-1)$ denote the change of the potential at time t. We will ensure that Φ is non-negative and bounded from above by a function of h, k, d, and length of the longest edge in T. Therefore, to show R-competitiveness, it suffices to show that the following holds at each time t: $\mathrm{ALG}(t) + \Delta_t\Phi \leq R \cdot \mathrm{OPT}(t)$.

To show this, we split the analysis into two parts: First, we let the adversary move a server (if necessary) to serve the request. Then we consider the move of the algorithm. Let $\Delta_t^{OPT}\Phi$ and $\Delta_t^{ALG}\Phi$ denote the changes in Φ due to the move of the adversary and the algorithm respectively. Clearly, $\Delta_t\Phi = \Delta_t^{OPT}\Phi + \Delta_t^{ALG}\Phi$, and thus it suffices to show the following two inequalities:

$$\Delta_t^{OPT}\Phi \leq R \cdot OPT(t) \tag{3.1}$$

$$ALG(t) + \Delta_t^{ALG}\Phi \leq 0 \tag{3.2}$$

Potential function

Before we define the potential, we need to formalize the notion of excess and deficiency in the subtrees of T. Let $d(a, b)$ denote the distance of points a and b. For $e = (u, v) \in T$, where v is the node closer to the root, we define $k_e := k_u + \frac{1}{d(u,v)} \sum_{s \in e} d(s, v)$. Note that this is the number of online servers in T_u, plus the possible single server in e counted fractionally, proportionally to its position along e. For an edge e, let $\ell(e)$ denote its level with the convention that the edges from leaf to their parents have level 1, and the edges from root to its children have level d. For $\ell = 1, \ldots, d$, let β_ℓ be some geometrically increasing constants that will be defined later. For any point $x \in T$, similarly

to k_x and k_x^-, we denote h_x and h_x^- the number of servers of the adversary in T_x and T_x^- respectively. For and edge e we define the *excess* E_e *of* e and the *deficiency* D_e *of* e as follows

$$E_e = \max\{k_e - \lfloor \beta_{\ell(e)} \cdot h_u \rfloor, 0\} \cdot d(u,v) \quad D_e = \max\{\lfloor \beta_{\ell(e)} \cdot h_u \rfloor - k_e, 0\} \cdot d(u,v).$$

Note that these compare k_e to h_u with respect to the *excess threshold* $\beta_{\ell(e)}$. We call an edge *excessive*, if $E_e > 0$, otherwise we call it *deficient*. Let us state a few basic properties of these two terms.

Observation 3.8. *Let e be an edge containing an algorithm's server s in its interior. If e is excessive, it cannot become deficient unless s moves upwards completely outside of the interior of e. Similarly, if e is deficient, it cannot become excessive unless s leaves interior of e completely.*

Note that no other server can pass through e while s still resides there and the contribution of s to k_e is a nonzero value strictly smaller than 1, while $\lfloor \beta_\ell h_u \rfloor$ is an integer.

Observation 3.9. *Let e be an edge and s be an algorithm's server in its interior moving by a distance x. Then either D_e or E_e changes exactly by x.*

This is because k_e changes by $x/d(u,v)$, and therefore the change of D_e (resp. E_e) is x.

Observation 3.10. *If an adversary server passes through the edge e, change in D_e (resp. E_e) will be at most $\lceil \beta_\ell \rceil \cdot d(u,v)$.*

To see this, note that $\lfloor \beta_{\ell(e)} h_u \rfloor \le \lfloor \beta_{\ell(e)}(h_u - 1) \rfloor + \lceil \beta_{\ell(e)} \rceil$.

We now fix the excess thresholds. We first define β depending on $\delta = k/h$ as,

$$\beta = 2 \text{ if } \delta \ge 2^d, \text{ and } \beta = \delta^{1/d} \text{ for } \delta \le 2^d.$$

For convenience in the calculations, we denote $\gamma := \frac{\beta}{\beta - 1}$. Note that, for all possible $\delta > 1$, our choices satisfy $1 < \beta \le 2$, $\gamma \ge 2$ and

$$\beta \le \delta^{1/d}. \tag{3.3}$$

For each $\ell = 1, \ldots, d$, we define the excess threshold for all edges in the level ℓ as $\beta_\ell := \beta^{\ell - 1}$.

Now, we can define the potential. Let

$$\Phi := \sum_{e \in T} \left(\alpha^D_{\ell(e)} D_e + \alpha^E_{\ell(e)} E_e \right),$$

where the coefficients α_ℓ^D and α_ℓ^E are as follows:

$$\begin{aligned}
\alpha_\ell^D &:= 2\ell - 1 && \text{for } \ell = 1, \dots, d \\
\alpha_d^E &:= \gamma(1 + \frac{1}{\beta}\alpha_d^D) \\
\alpha_\ell^E &:= \sum_{i=\ell}^{d-1} \gamma^{i-\ell+1} \left(2 + \frac{1}{\beta}\alpha_i^D\right) + \gamma^{d-\ell}\alpha_d^E && \text{for } \ell = 1, \dots, d-1.
\end{aligned}$$

Note that $\alpha_\ell^D > \alpha_{\ell-1}^D$ and $\alpha_\ell^E < \alpha_{\ell-1}^E$ for all $1 < \ell \leq d$. The latter follows as the multipliers $\gamma^{i-\ell+1}$ and $\gamma^{d-\ell}$ decrease with increasing ℓ and moreover the summation in the first term of α_ℓ^E has fewer terms as ℓ increases.

To prove the desired competitive ratio for Algorithm 1, the idea will be to show that the *good* moves (when a server enters a region with deficiency, or leaves a region with excess) contribute more than the *bad* moves (when a server enters a region with excess, or leaves a region that is already deficient).

As the dynamics of the servers can be decomposed into elementary moves, it suffices to only analyze these. We will assume that no servers of A are located at a node. This is without loss of generality, as only the moving servers can cause a change in the potential, and each server appears at a node just for an infinitesimal moment during its motion. Note that this assumption implies that each edge e containing a server $s \in A$ is either excessive or deficient, i.e. either $E_e > 0$ or $D_e > 0$.

The following two lemmas give some properties of the deficient and excessive subtrees, which will be used later in the proof of Theorem 3.3.

Lemma 3.11 (Excess in Phase 1)**.** *Consider a moment during Phase 1 and assume that no server of A resides at a node. For the set $E = \{s \in A \mid s \in e, E_e > 0\}$ of servers which are located in excessive edges, and for $D = A \setminus E$, the following holds.*

$$\sum_{s \in D} k_s \leq \frac{1}{\beta} k \qquad \textit{and} \qquad \sum_{s \in E} k_s \geq \frac{1}{\gamma} k.$$

Proof. During Phase 1, each server of the algorithm resides in T_s for some $s \in E \cup D$; therefore $k = \sum_{s \in E} k_s + \sum_{s \in D} k_s$. For each $s \in D$, let $\ell(s)$ be the level of the edge e containing s. We have that $k_s \leq \lfloor \beta_{\ell(s)} \cdot h_s \rfloor$, otherwise E_e would be positive. Therefore we have

$$\sum_{s \in D} k_s \leq \sum_{s \in D} \lfloor \beta_{\ell(s)} h_s \rfloor \leq \sum_{s \in D} \beta_d \cdot h_s \leq \beta_d \cdot h = \beta^{d-1} \frac{k}{\delta} \leq \beta^{d-1} \frac{k}{\beta^d} = \frac{1}{\beta} k,$$

where the last inequality comes from (3.3).

To prove the second inequality, observe that

$$\sum_{s\in E} k_s = k - \sum_{s\in D} k_s \geq k - \frac{1}{\beta}k = (1-\frac{1}{\beta})k = \frac{1}{\gamma}k. \qquad \square$$

Lemma 3.12 (Excess in Phase 2)**.** *Consider a moment during Phase 2 and assume that no server of A resides at a node. Let ℓ be the level of the edge containing q and $A' = A \setminus \{q\}$. For the set $E = \{s \in A' \mid s \in e, E_e > 0\}$ of servers which are located in excessive edges, and for $D = A' \setminus E$, the following holds. If $k_q^- \geq \lfloor \beta_\ell h_q^- \rfloor$, then we have*

$$\sum_{s\in D} k_s \leq \frac{1}{\beta}k_q^- \qquad \text{and} \qquad \sum_{s\in E} k_s \geq \frac{1}{\gamma}k_q^-.$$

Proof. The proof is quite similar to the proof of Lemma 3.11. Instead of using the fact that $\delta \geq \beta^d = \beta \cdot \beta^{d-1} = \beta \cdot \beta_d$, we now crucially use the fact that $\beta_\ell = \beta \cdot \beta_{\ell-1}$. However, now we have to be more careful with the counting of the servers of the adversary.

For each $s \in D$, let $\ell(s)$ be the level of the edge e containing s. Similarly to the proof of Lemma 3.11, we have $k_s \leq \lfloor \beta_{\ell(s)} \cdot h_s \rfloor \leq \beta_{\ell-1} \cdot h_s$ for each server $s \in D$, since $\ell(s) \leq \ell - 1$. Recall that we assume that the adversary has already served the request. Let a be the server of the adversary located at the requested point. Since no online servers reside in the path between q and the requested point, a does not belong to T_s for any $s \in D \cup E$. Therefore we have $\sum_{s\in D} h_s \leq \sum_{s\in(D\cup E)} h_s \leq h_q^- - 1$. We get that

$$\sum_{s\in D} k_s \leq \sum_{s\in D} \beta_{\ell-1} \cdot h_s \leq \beta_{\ell-1}(h_q^- - 1). \tag{3.4}$$

To finish the proof of the first inequality, observe that our assumption that $k_q^- \geq \lfloor \beta_\ell h_q^- \rfloor$ implies

$$k_q^- \geq \lfloor \beta_\ell h_q^- \rfloor \Rightarrow \beta_\ell h_q^- \leq k_q^- + 1 \Leftrightarrow h_q^- \leq \frac{k_q^- + 1}{\beta_\ell}.$$

Therefore, from (3.4) we have

$$\sum_{s\in D} k_s \leq \beta_{\ell-1}(h_q^- - 1) \leq \beta_{\ell-1}(\frac{k_q^- + 1}{\beta_\ell} - 1) = \frac{\beta_{\ell-1}}{\beta_\ell}k_q^- + \frac{1}{\beta} - \beta_{\ell-1} \leq \frac{1}{\beta}k_q^-.$$

For the second inequality, note that $\frac{1}{\gamma} = (1 - \frac{1}{\beta})$. Since $k_q^- = \sum_{s\in D} k_s + \sum_{s\in E} k_s$, we have

$$\sum_{s\in E} k_s \geq k_q^- - \frac{1}{\beta}k_q^- = \frac{1}{\gamma}k_q^-. \qquad \square$$

Proof of Theorem 3.3

We now show the main technical result, which directly implies Theorem 3.3.

Theorem 3.13. *The competitive ratio of Algorithm 1 in depth-d trees is* $O(d \cdot \gamma^d)$.

This implies Theorem 3.3 as follows. If $\delta \geq 2^d$, we have $\beta = 2$ and $\gamma = 2$, and we get the competitive ratio $O(d \cdot 2^d)$. For $1 < \delta < 2^d$, we have $\beta = \delta^{1/d}$ and therefore the competitive ratio is $O(d \cdot (\frac{\delta^{1/d}}{\delta^{1/d}-1})^d)$. In particular, if $\delta = (1+\epsilon)$ for some $0 < \epsilon \leq 1$, we have $\beta = (1+\epsilon)^{1/d} \geq 1 + \frac{\epsilon}{2d}$. This implies that $\gamma = \frac{\beta}{\beta-1} \leq \frac{\beta}{\epsilon/(2d)} = \frac{\beta 2d}{\epsilon}$. Using (3.3), we get that

$$\gamma^d \leq \beta^d \cdot (\frac{2d}{\epsilon})^d \leq \delta \cdot (\frac{2d}{\epsilon})^d = O((\frac{2d}{\epsilon})^d).$$

Thus, we get the ratio $O(d \cdot (\frac{2d}{\epsilon})^d)$.

We now prove Theorem 3.13.

Proof of Theorem 3.13. As Φ is non-negative and bounded from above by a function of h, k, d, and the length of the longest edge in T, it suffices to show the inequalities (3.1) and (3.2).

We start with (3.1) which is straightforward. By Observation 3.10, the move of a single adversary's server through an edge e of length x_e changes D_e or E_e in the potential by at most $\lceil \beta_{\ell(e)} \rceil x_e$. As the adversary incurs cost x_e during this move, we need to show the following inequalities:

$$\lceil \beta_\ell \rceil x_e \cdot \alpha_\ell^D \leq R \cdot x_e \quad \text{for all } 1 \leq \ell \leq d$$
$$\lceil \beta_\ell \rceil x_e \cdot \alpha_\ell^E \leq R \cdot x_e \quad \text{for all } 1 \leq \ell \leq d.$$

As we show in Lemma 3.16 below, $\lceil \beta_\ell \rceil \alpha_\ell^D$ and $\lceil \beta_\ell \rceil \alpha_\ell^E$ are of order $O(d \cdot \gamma^d)$. Therefore, setting $R = \Theta(d \cdot \gamma^d)$ will satisfy (3.1).

We now consider (3.2) which is much more challenging to show. Let us denote A_E the set of edges containing some server from A in their interior. We call an *elementary step* a part of the motion of the algorithm during which A and A_E remain unchanged, and all the servers of A are located in the interior of the edges of T. Lemmas 3.14 and 3.15 below show that (3.2) holds during an elementary step, and the theorem would follow by summing (3.2) over all the elementary steps. □

Lemma 3.14. *During an elementary step in Phase 1 of Algorithm 1, the inequality* (3.2) *holds.*

Proof. Without loss of generality, let us assume that the elementary step lasted exactly 1 unit of time. This makes the distance traveled by each server equals to its speed, and makes calculations cleaner.

Let ALG denote the cost incurred by the algorithm during this step. Note that ALG $= 1$, since by Observation 3.7, the total speed of the servers in A, is 1.

To estimate the change of the potential $\Delta\Phi$, we decompose A into two sets, called D and E. D is the set of the servers of A residing in deficient edges, and E are the servers residing in excessive edges. Next, we evaluate $\Delta\Phi$ due to the movement of the servers from each class separately.

The movement of servers of E is *good*, i.e. decreases the excess in their edges, while the movement of servers of D increases the deficiency. By taking the largest possible α^D (which is α_d^D) and the smallest possible α^E (which is α_d^E) coefficient in the estimation, we can bound the change of the potential due to the move of the servers of A as

$$\Delta\Phi \leq \alpha_d^D \sum_{s\in D} \frac{k_s}{k} - \alpha_d^E \sum_{s\in E} \frac{k_s}{k} \leq \frac{1}{\beta}\alpha_d^D - \frac{1}{\gamma}\alpha_d^E,$$

where the last inequality holds due to the Lemma 3.11. We get that

$$\text{ALG} + \Delta\Phi \leq 1 + \frac{1}{\beta}\alpha_d^D - \frac{1}{\gamma}\alpha_d^E = 1 + \frac{1}{\beta}\alpha_d^D - \frac{1}{\gamma}\cdot\gamma(1+\frac{1}{\beta}\alpha_d^D) = 0. \qquad \square$$

Lemma 3.15. *During an elementary step in Phase 2 of Algorithm 1, the inequality* (3.2) *holds.*

Proof. Similarly to the proof of the preceding lemma, we denote by ALG the cost incurred by the algorithm and we assume (without loss of generality) that the duration of the elementary step is exactly 1 time unit, so that the speed of each server equals the distance it travels. As the speed of the server q is 1, it also moves by distance 1. Moreover, by Observation 3.7, the servers in T_q^- (if any) move in total by $\sum_{s\in A\setminus\{q\}} \frac{k_s}{k_q^-} = 1$, and therefore ALG ≤ 2. We denote by ℓ the level of the edge containing q. To estimate the change of the potential, we consider two cases.

1. *When $k_q^- \geq \lfloor \beta_\ell h_q^- \rfloor$.* Here the movement of q increases the excess in the edge containing q. Let us denote E (resp. D) the servers of $A\setminus\{q\}$ residing in the excessive (resp. deficient) edges. By taking the largest possible α^D (which is $\alpha_{\ell-1}^D$) and the smallest possible α^E (which is $\alpha_{\ell-1}^E$) coefficient in the estimation, we can upper bound the change of the potential due to the move of the servers in T_q^- as,

$$\alpha_{\ell-1}^D \sum_{s\in D} \frac{k_s}{k_q^-} - \alpha_{\ell-1}^E \sum_{s\in E} \frac{k_s}{k_q^-} \leq \frac{1}{\beta}\alpha_{\ell-1}^D - \frac{1}{\gamma}\alpha_{\ell-1}^E,$$

where the above inequality holds due to the Lemma 3.12. As the movement of q itself causes an increase of Φ by α_ℓ^E, we have

$$ALG + \Delta\Phi \leq 2 + \frac{1}{\beta}\alpha_{\ell-1}^D - \frac{1}{\gamma}\alpha_{\ell-1}^E + \alpha_\ell^E.$$

To see that this is non-positive, recall that $\alpha_\ell^E = \sum_{i=\ell}^{d-1} \gamma^{i-\ell+1}\left(2 + \frac{1}{\beta}\alpha_i^D\right) + \gamma^{d-\ell}\alpha_d^E$. Therefore

$$\begin{aligned}\frac{1}{\gamma}\alpha_{\ell-1}^E &= \frac{1}{\gamma}\sum_{i=\ell-1}^{d-1}\gamma^{i-\ell+2}\big(2 + \frac{1}{\beta}\alpha_i^D\big) + \frac{1}{\gamma}\gamma^{d-\ell+1}\alpha_d^E \\ &= \big(2 + \frac{1}{\beta}\alpha_{\ell-1}^D\big) + \sum_{i=\ell}^{d-1}\gamma^{i-\ell+1}\big(2 + \frac{1}{\beta}\alpha_i^D\big) + \gamma^{d-\ell}\alpha_d^E \\ &= 2 + \frac{1}{\beta}\alpha_{\ell-1}^D + \alpha_\ell^E.\end{aligned}$$

2. *When $k_q^- < \lfloor \beta_\ell h_q^- \rfloor$.* This case is much simpler. All the movement inside of T_q^- might contribute to the increase of deficiency at level at most $\ell - 1$. On the other hand, q then causes a decrease of deficiency at level ℓ and we have ALG $+\Delta\Phi \leq 2 + \alpha_{\ell-1}^D - \alpha_\ell^D$. This is less or equal to 0, as $\alpha_\ell^D \geq \alpha_{\ell-1}^D + 2$ $\square$.

Lemma 3.16. *For each $1 \leq \ell \leq d$, both $\lceil \beta_\ell \rceil \alpha_\ell^D$ and $\lceil \beta_\ell \rceil \alpha_\ell^E$ are of order $O(d \cdot \gamma^d)$.*

Proof. We have defined $\alpha_\ell^D = 2\ell - 1$, and therefore

$$\lceil \beta_\ell \rceil \alpha_\ell^D \leq 2 \cdot \beta^\ell(2\ell - 1) \leq 2 \cdot \beta^d(2d - 1) = O(d \cdot \gamma^d),$$

since we have chosen $1 < \beta \leq 2$ and $\gamma \geq 2$ for any possible δ.

Now we focus on α_ℓ^E. Note that $\alpha_d^E = \gamma(1 + \frac{1}{\beta}\alpha_d^D)$, which is $O(d \cdot \gamma)$, as $\beta > 1$ and $\alpha_d^D = O(d)$. For α_ℓ^E, we have

$$\begin{aligned}\alpha_\ell^E &= \sum_{i=\ell}^{d-1}\gamma^{i-\ell+1}\left(2 + \frac{1}{\beta}\alpha_i^D\right) + \gamma^{d-\ell}\alpha_d^E \leq (2 + \frac{1}{\beta}\alpha_{d-1}^D)\sum_{i=\ell}^{d-1}\gamma^{i-\ell+1} + \gamma^{d-\ell}\alpha_d^E = \\ &= (2 + \frac{1}{\beta}\alpha_{d-1}^D)\sum_{j=1}^{d-\ell}\gamma^j + \gamma^{d-\ell}\alpha_d^E = (2 + \frac{1}{\beta}\alpha_{d-1}^D)\frac{\gamma \cdot (\gamma^{d-\ell} - 1)}{\gamma - 1} + \gamma^{d-\ell}\alpha_d^E \\ &\leq \beta(2 + \frac{1}{\beta}\alpha_{d-1}^D)\gamma^{d-\ell} + \gamma^{d-\ell}\alpha_d^E = \gamma^{d-\ell}(2\beta + 2d - 3 + \alpha_d^E) = O(\gamma^{d-\ell+1} \cdot d),\end{aligned}$$

where we used that $\frac{\gamma}{\gamma-1} = \beta$, $1 < \beta \leq 2$ and $\alpha_d^E = O(\gamma \cdot d)$. Therefore, as $\beta_\ell \leq \gamma^{\ell-1}$, we have $\lceil \beta_\ell \rceil \alpha_\ell^E = O(\gamma^d d)$, and this concludes the proof. $\square$

3.4 Algorithm for Bounded-Diameter Trees

Since Algorithm 1 works for depth-d trees with arbitrary edge lengths, we can embed any diameter-d tree into a depth-d tree with a very small distortion by adding fake paths of short edges to all nodes.

More precisely, let T be a tree of diameter d with arbitrary edge lengths, and let α be the length of the shortest edge of T (for any finite T such α exists). We fix $\epsilon > 0$ a small constant. We create an embedding T' of T as follows. We choose the root r arbitrarily, and to each node $v \in T$ such that the path from r to v contains ℓ edges, we attach a path containing $d - \ell$ edges of total length $\epsilon\alpha/2$. The leaf at the end of this path we denote v'. We run Algorithm 1 in T' and each request at $v \in T$ we translate to $v' \in T'$. We maintain the correspondence between the servers in T and the servers in T', and the same server which is finally moved to v' by Algorithm 1, we also use to serve the request $v \in T$.

For the optimal solutions on T and T' we have $(1+\epsilon)\,\mathrm{OPT}(T) \geq \mathrm{OPT}(T')$, since any feasible solution in T we can be converted to a solution in T' with cost at most $(1+\epsilon)$ times higher. By Theorem 3.13, we know that the cost of Algorithm 1 in T' is at most $R \cdot \mathrm{OPT}(T')$, for $R = \Theta(d \cdot \gamma^{d+1})$, and therefore we have $\mathrm{ALG}(T') \leq (1+\epsilon)R \cdot \mathrm{OPT}(T)$.

3.5 Lower Bounds

In this section we prove Theorems 3.4 and 3.2. We first show a general lower bound on the competitive ratio of any algorithm for depth-2 HSTs. Then we give lower bound on the competitive ratio of WFA. Similarly to the previous section, given a tree T and a point $x \in T$ (either a node or a location on some edge), we define T_x as the subtree consisting of all points below x including x itself. If u is a parent of a leaf, we call T_u an *elementary subtree*.

3.5.1 General Lower Bound for Depth-2 HSTs

We now give a lower bound on the competitive ratio of any deterministic online algorithm on depth-2 HSTs. In particular, we show that for sufficiently large h, any deterministic online algorithm has competitive ratio at least 2.4.

The metric space is a depth-2 HST T with the following properties: T contains at least $k+1$ elementary subtrees and each one of them has at least h leaves. To ease our calculations, we assume that edges of the lower level have length $\epsilon \ll 1$ and edges of the upper level have length $1 - \epsilon$. So the distance between leaves of different elementary subtrees is 2.

Theorem 3.4 (restated)**.** *For sufficiently large h, even when $k/h \to \infty$, there is no 2.4-competitive deterministic online algorithm, even for depth-2 HSTs.*

Proof. Without loss of generality both the online and offline algorithms are "lazy", i.e. they move a server only for serving a request (this is a folklore k-server property, see e.g. [19]). Since requests only appear at leaves of T, all the offline and online servers are always located at the leaves. We say that a server is inside an elementary subtree T_u, if it is located at some leaf of T_u. If there are no online servers at the leaves of T_u, we say that T_u is *empty*. Observe that at any given time there exists at least one empty elementary subtree.

Let $\mathcal{A}$ be an online algorithm. The adversarial strategy consists of arbitrarily many iterations of a phase. During a phase, some offline servers are moved to an empty elementary subtree T_u and requests are made there until the cost incurred by $\mathcal{A}$ is sufficiently large. At this point the phase ends and a new phase may start in another empty subtree. Let ALG and ADV denote the cost of $\mathcal{A}$ and the adversary respectively during a phase. We will ensure that for all phases $\mathrm{ALG} \geq 2.4 \cdot \mathrm{ADV}$. This implies the lower bound on the competitive ratio of $\mathcal{A}$.

We describe a phase of the adversarial strategy. The adversary moves some $\ell \leq h$ servers to the empty elementary subtree T_u and makes requests at leaves of T_u until $\mathcal{A}$ brings m servers there. In particular, each request appears at a leaf of T_u that is not occupied by a server of $\mathcal{A}$. We denote by $s(i)$ the cost that $\mathcal{A}$ has to incur for serving requests inside T_u until it moves its ith server there (this does not include the cost of moving the server from outside T_u). Clearly, $s(1) = 0$ and $s(i) \leq s(i+1)$ for all $i > 0$. The choice of ℓ and m depends on the values $s(i)$ for $2 \leq i \leq h$. We will now show that for any values of $s(i)$'s, the adversary can choose ℓ and m such that $\mathrm{ALG} \geq 2.4 \cdot \mathrm{ADV}$.

First, if there exists an i such that $s(i) \geq 3i$, we set $\ell = m = i$. Intuitively, the algorithm is too slow in bringing its servers to T_u in this case. Both $\mathcal{A}$ and the adversary incur a cost of $2i$ to move i servers to T_u. However, $\mathcal{A}$ pays a cost of $s(i)$ for serving requests inside T_u, while the adversary can serve all those requests at zero cost (all requests can be located at leaves occupied by offline servers). Overall, the cost of $\mathcal{A}$ is $2i + s(i) \geq 5i$, while the offline cost is $2i$. Thus we get that $\mathrm{ALG} \geq 2.5 \cdot \mathrm{ADV}$.

Similarly, if $s(i) \leq (10i - 24)/7$ for some i, we choose $\ell = 1$ and $m = i$. Roughly speaking, in that case the algorithm is too "aggressive" in bringing its first i servers, thus incurring a large movement cost. Here, the adversary only moves one server to T_u. Each request is issued at an empty leaf of T_u. Therefore $\mathcal{A}$ pays for each request in T_u and the same holds for the single server of the adversary. So, $\mathrm{ALG} = 2i + s(i)$ and $\mathrm{ADV} = 2 + s(i)$. By our assumption on $s(i)$, this gives

$$\mathrm{ALG} - 2.4 \cdot \mathrm{ADV} = 2i + s(i) - 4.8 - 2.4 \cdot s(i) = 2i - 4.8 - 1.4 \cdot s(i) \geq 0.$$

We can thus restrict our attention to the case that $s(i) \in (\frac{10i-24}{7}, 3i)$, for all $2 \leq i \leq h$. Now we want to upper bound the offline cost, for $1 < \ell < h$ and

$m = h$. Clearly, the offline movement cost is 2ℓ, and for the time that $\mathcal{A}$ has less than ℓ servers in T_u, the offline cost is zero. It remains to count the offline cost during the time that $\mathcal{A}$ has at least ℓ servers inside T_u.

For the part of the request sequence when $\mathcal{A}$ has $\ell \leq j < h$ servers in T_u, it incurs a cost $s(j+1) - s(j)$. Since the problem restricted to an elementary subtree is equivalent to the paging problem, using the lower bound result of [66] for paging[1], we get that the adversary incurs a cost of at most $(s(j+1) - s(j)) \cdot \frac{j-\ell+1}{j} + 2\epsilon j \leq (s(j+1) - s(j)) \cdot \frac{h-\ell}{h-1} + 2\epsilon j$. Also, there are $h - \ell$ requests where $\mathcal{A}$ brings new servers. Clearly, those requests cost to the adversary at most $(h-\ell)2\epsilon$. Thus the cost of the adversary inside T_u is at most

$$\sum_{j=\ell}^{h-1} \Big((s(j+1) - s(j)) \cdot \frac{h-\ell}{h-1} + 2\epsilon j \Big) + (h-\ell)2\epsilon$$
$$= \frac{h-\ell}{h-1} \big(s(h) - s(\ell) \big) + 2\epsilon \sum_{j=\ell}^{h-1} j + (h-\ell)2\epsilon.$$

For any value of h, we can take ϵ small enough, such that the terms involving ϵ tend to zero. Thus the upper bound on the offline cost is arbitrarily close to $2\ell + (s(h) - s(\ell)) \cdot \frac{h-\ell}{h-1}$, where the first term is the cost of moving ℓ servers to T_u. Let us denote $c = s(h)/h$. Note that assuming h is large enough, $c \in (1,3)$. We get that

$$\frac{\text{ALG}}{\text{ADV}} \geq \frac{2h + s(h)}{2\ell + (s(h) - s(\ell)) \cdot \frac{h-\ell}{h-1}} \geq \frac{2h + ch}{2\ell + (ch - \frac{10\ell - 24}{7}) \cdot \frac{h-\ell}{h-1}} \tag{3.5}$$
$$= \frac{2h + ch}{2\ell + h \cdot (c - \frac{10\ell - 24}{7h}) \cdot \frac{h-\ell}{h-1}} \tag{3.6}$$

We now show that for every value of $c \in (1,3)$, there is an $\ell = \lfloor \beta h \rfloor$, where $\beta \in (0,1)$ is a constant depending on c, such that this ratio is at least 2.4. First, as h is large enough, (3.6) is arbitrarily close to

$$\frac{2h + ch}{2\ell + h \cdot (c - 10\ell/7h) \cdot (1 - \ell/h)} = \frac{2h + ch}{2\ell + h\big(c - c\ell/h - 10\ell/(7h) + 10\ell^2/(7h^2)\big)}$$

$$\geq \frac{2h + ch}{2\beta h + h\big(c - c(\beta h - 1)/h - 10(\beta h - 1)/(7h) + 10\beta^2 h^2/(7h^2)\big)}$$

[1] Sleator and Tarjan [66] show that an adversary with h servers can create a request sequence against an online algorithm with k servers such that the cost of the adversary is at most $\frac{k-h+1}{k}$ ALG $+(k-h+1) \cdot D$, where D is the distance between two leaves.

$$\frac{2+c}{2\beta+c-c\beta+c/h-10\beta/7+10/(7h)+10\beta^2/7} \to \frac{2+c}{(10/7)\cdot\beta^2+(4/7-c)\beta+c}$$

We choose $\beta := (c-4/7)/(20/7)$. Note that, as $c \in (1,3)$, we have that $\beta < 1$ and hence $\ell < h$. The expression above then evaluates to $(2+c)/(c-(c-\frac{4}{7})^2/\frac{40}{7})$. By standard calculus, this expression is minimized at $c = (2\sqrt{221}-14)/7$ where it attains a value higher than 2.419. □

3.5.2 Lower Bound for WFA on Depth-3 HSTs

We give an $\Omega(h)$ lower bound on the competitive ratio of the Work Function Algorithm (WFA) in the (h,k)-setting. More precisely, we show a lower bound of $h+1/3$ for $k=2h$. We first present the lower bound for a depth-3 HST. In Section 3.6 we show how the construction can be adapted to work for the line. We also show that this exactly matches the upper bound of $(h+1)\cdot \mathrm{OPT}_h - \mathrm{OPT}_k$ implied by results of [51, 17] for the line.

The basic idea behind the lower bound is to trick the WFA to use only h servers for servicing requests in an "active region" for a long time before it moves its extra available online servers. Moreover, we make sure that during the time the WFA uses h servers in that region, it incurs an $\Omega(h)$ times higher cost than the adversary. Finally, when the WFA brings its extra servers, the adversary moves all its servers to some different region and starts making requests there. So eventually, WFA is unable to use its additional servers in a useful way to improve its performance.

Theorem 3.2 *(restated) The competitive ratio of the WFA in a depth-3 HST for $k=2h$ is at least $h+1/3$.*

Proof. Let T be a depth-3 HST. We assume that the lengths of the edges in a root-to-leaf path are $\frac{1-\epsilon}{2}, \frac{\epsilon-\epsilon'}{2}, \frac{\epsilon'}{2}$, for $\epsilon' \ll \epsilon \ll 1$. So the diameter of T is 1 and the diameter of depth-2 subtrees is ϵ. We also assume that $N = \frac{1}{\epsilon}$ is integer such that $N \gg \frac{3h}{2}$. Let L and R be two subtrees of depth 2. Inside each one of them, we focus on 2 elementary subtrees L_1, L_2 and R_1, R_2 respectively (see figure 3.3), each one containing exactly h leaves. All requests appear at leaves of those elementary subtrees. Initially all servers are at leaves of L. Thus, both the WFA and the adversary always have their servers at leaves. This way, we say that some servers of the WFA (or the adversary) are inside a subtree, meaning that they are located at some leaves of that subtree.

The adversarial strategy consists of arbitrary many iterations of a phase. At the beginning of the phase, all online and offline servers are in the same subtree, either L or R. At the end of the phase, all servers have moved to the other subtree (R or L resp.), so a new phase may start. Before describing

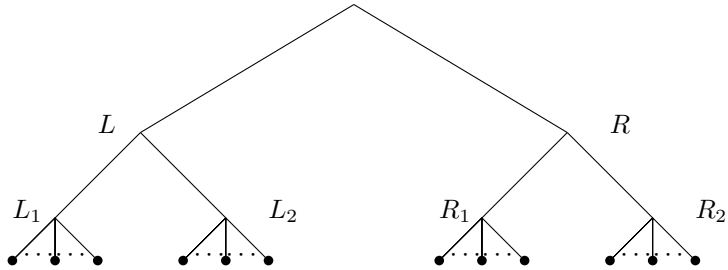


Figure 3.3: The tree where the lower bound for WFA is applied: All requests arrive at leaves of L_1, L_2, R_1 and R_2.

the phase, we give the basic strategy, which is repeated many times. For any elementary subtree T with leaves $t_1, \ldots, t_h$, any $1 \leq \ell \leq h$ and any $c > 0$, we define strategy $S(T, \ell, c)$.

Strategy $S(T, \ell, c)$:

1. While the WFA has $i < \ell$ servers in T: Request points $t_1, \ldots, t_\ell$ in an adversarial way (i.e each time request a point where the WFA does not have a server).

2. If $\ell \geq 2$ and the WFA has $i \geq \ell$ servers in T, then: While optimal cost to serve all requests using $\ell - 1$ servers (starting at $t_1, \ldots, t_{\ell-1}$) is smaller than c, request points $t_1, \ldots, t_\ell$ in a round-robin way.

Note that the 2nd part might not be executed at all, depending on the values of ℓ and c.

We now describe a left-to-right phase, i.e., a phase which starts with all servers at L and ends with all servers at R. A right-to-left phase is completely symmetric, i.e., we replace R by L and R_i by L_i. Recall that $N = \frac{1}{\epsilon}$.

Left-to-right phase:
Step 1: For $\ell = 1$ to h: Apply strategy $S(R_1, \ell, 2)$
Step 2: For $i = 1$ to $N + 1$: For $\ell = 1$ to h: Apply strategy $S(R_2, \ell, 2\epsilon)$ For $\ell = 1$ to h: Apply strategy $S(R_1, \ell, 2\epsilon)$

Intuitively, the WFA starts with no server at R and at the end of Step 1, it has h servers there (more precisely, in R_1). During Step 2, the WFA moves all its servers to R, as we show later.

Let ALG and ADV be the online and offline cost respectively. We now state two basic lemmas for evaluating ALG and ADV during each step. Proofs of

those lemmas, require a characterization of work function values, which comes later on. Here we just give the intuition behind the proofs.

Lemma 3.17. *During Step 1,* $\text{ALG} \geq h^2 - \epsilon' \cdot (h-1)h$ *and* $\text{ADV} = h$.

Intuitively, we will exploit the fact that the WFA moves its servers too slowly towards R. In particular, we show (Lemma 3.37) that whenever the WFA has $i < h$ servers in R, it incurs a cost of at least $(2 - 2\epsilon') \cdot i$ inside R before it moves its $(i+1)$th server there. This way we get that the cost of the algorithm is at least $h + \sum_{i=1}^{h-1}(2 - 2\epsilon') \cdot i = h^2 - \epsilon' \cdot (h-1)h$. On the other hand, the adversary moves its h servers by distance 1 (from L to R_1) and then it serves all requests at zero cost.

Lemma 3.18. *During Step 2,* $\text{ALG} \geq 2h^2 + h - 3\epsilon h^3 - 2\frac{\epsilon'}{\epsilon} \cdot (h-1) \cdot h$ *and* $\text{ADV} = (2 + 2\epsilon)h$.

Roughly speaking, to prove this lemma we make sure that for almost the whole Step 2, the WFA has h servers in R and they incur a cost h times higher than the adversary. The additional servers move to R almost at the end of the phase. The cost of the adversary is easy to calculate. There are $N+1 = 1/\epsilon+1$ iterations, where in each one of them the adversary moves its h servers from R_1 to R_2 and then back to R_1, incurring a cost of $2\epsilon \cdot h$.

The theorem follows from lemmata 3.17 and 3.18. Summing up for the whole phase, the offline cost is $(3 + 2\epsilon) \cdot h$ and the online cost is at least $3h^2 + h - 3\epsilon h^3 - \frac{\epsilon'}{\epsilon}2(h-1)h - \epsilon'(h-1)h$. We get that

$$\frac{\text{ALG}}{\text{ADV}} \geq \frac{3h^2 + h - 3\epsilon h^3 - \frac{\epsilon'}{\epsilon}2(h-1)h - \epsilon'(h-1)h}{(3+2\epsilon)h} \to \frac{3h^2+h}{3h} = h + \frac{1}{3}$$

□

Work Function values

We now give a detailed characterization of how the work function evolves during left-to-right phases. For right-to-left phases the structure is completely symmetric.

Basic Properties: We state some standard properties of work functions that we use extensively in the rest of this section. Proofs of those properties can be found e.g. in [19]. First, for any two configurations X, Y at distance $d(X, Y)$ the work function w satisfies,

$$w(X) \leq w(Y) + d(X, Y).$$

Also, let w and w' be the work function before and after a request r respectively. For a configuration X such that $r \in X$, we have that $w'(X) = w(X)$. Last,

let C and C' be the configurations of the WFA before and after serving r respectively. The work function w' satisfies $w'(C) - w'(C') = d(C, C')$.

Notation: Let (L^i, R^{k-i}) denote a configuration which has i servers at L and $(k-i)$ servers at R. Let $w(L^i, R^{k-i})$ be the minimum work function value of a configuration (L^i, R^{k-i}). If we need to make precise how many servers are in each elementary subtree, we denote by (L^i, r_1, r_2) a configuration which has i servers at L, r_1 servers at R_1 and r_2 servers at R_2. Same as before, $w(L^i, r_1, r_2)$ denotes the minimum work function value of those configurations.

Definition: For two configurations X and Y, we say that X *supports* Y, if $w(Y) = w(X) + d(X, Y)$. The set of configurations that are not supported by any other configuration is called the *support* of work function w. The following observation will be useful later in our proofs.

Initialization: For convenience we assume that at the beginning of the phase the minimum work function value of a configuration is zero. Note that this is without loss of generality: If $m = \min_X w(X)$ when the phase starts, we just subtract m from the work function values of all configurations. We require the invariant that at the beginning of the phase, for $0 \leq i \leq k$:

$$w(L^{k-i}, R^i) = i.$$

This is clearly true for the beginning of the first phase, as all servers are initially at L. We are going to make sure, that the symmetric condition (i.e. $w(L^i, R^{k-i}) = m + i$) holds at the end of the phase, so a right-to-left phase may start.

We now state two auxiliary lemmata which would be helpful later.

Lemma 3.19. *Consider a left-to-right phase. At any time after the end of strategy $S(R_1, \ell, 2)$, for $1 \leq \ell \leq h$, we have that $w(L^{k-\ell+1}, R^{\ell-1}) = w(L^{k-\ell}, R^{\ell})+1$.*

At a high-level, the reason that this lemma holds is that any schedule $\mathcal{S}$ that uses no more than $\ell - 1$ servers in R_1, incurs a cost of at least 2 during the execution of $S(R_1, \ell, 2)$. Thus, by moving an ℓth server to R_1 (cost 1), and serving all requests of $S(R_1, \ell, 2)$ at zero cost, we obtain a schedule which is cheaper than $\mathcal{S}$ by at least 1. The formal proof is deferred to the end of this section.

Lemma 3.20. *Consider a left-to-right phase. At any time after the end of strategy $S(R_1, \ell, 2)$, for $1 \leq \ell \leq h$, we have that $w(L^{k-i}, R^i) = w(L^{k-\ell}, R^{\ell}) + (\ell - i)$ for any $0 < i < \ell$.*

Proof. We use backwards induction on i. Note that the base case $i = \ell - 1$ is true due to Lemma 3.19. Let t be any time after the end of strategy $S(R_1, \ell, 2)$ with work function w.

Induction Step: Assume that the lemma is true for some $i < \ell$. We show that the lemma holds for $i-1$. Clearly, time t is after the end of $S(R_1, i, 2)$. From Lemma 3.19 we have that

$$w(L^{k-i+1}, R^{i-1}) = w(L^{k-i}, R^i) + 1. \tag{3.7}$$

From the inductive hypothesis we have that

$$w(L^{k-i}, R^i) = w(L^{k-\ell}, R^\ell) + (\ell - i). \tag{3.8}$$

By (3.7), (3.8) we get that

$$w(L^{k-i+1}, R^{i-1}) = w(L^{k-i}, R^i) + 1 = w(L^{k-\ell}, R^\ell) + \ell - (i-1). \qquad \square$$

First Step: We now focus on the first step of the phase. Using the two lemmata above, we can characterize the structure of the work function at the end of the execution of $S(R_1, \ell, 2)$, for $1 \le \ell \le h$.

Lemma 3.21. *Consider a left-to-right phase. When strategy $S(R_1, \ell, 2)$ ends, for any $1 \le \ell \le h$, the following two things hold:*

1. *For all $0 \le i < \ell$, $w(L^{k-i}, R^i) = w(L^{k-\ell}, R^\ell) + (\ell - i)$*
2. *For all $\ell < i \le k$, $w(L^{k-i}, R^i) = w(L^{k-\ell}, R^\ell) + (i - \ell)$*

In other words, having ℓ servers in R is the lowest state, and all other states are tight with respect to it.

Proof. For $i < \ell$, the lemma follows from Lemma 3.20. We focus on $i \ge \ell$: Until the end of $S(R_1, \ell, 2)$ there are $\ell \le i$ points in R requested, so $w(L^{k-i}, R^i)$ does not change. Since at the beginning $w(L^{k-i}, R^i) = i$, we have that for all $i \ge \ell + 1$

$$w(L^{k-i}, R^i) - w(L^{k-\ell}, R^\ell) = i - \ell. \qquad \square$$

This characterization of work function values is sufficient to give us the lower bound on the cost of the WFA during Step 1. The reader might prefer to move to Lemma 3.37 which estimates the cost of the WFA during the time interval that it has $i < h$ servers in R, and implies Lemma 3.17.

Second step: By lemma 3.21, at the beginning of the second step, the work function values satisfy:

$$w(L^{2h-i}, R^i) = w(L^h, R^h) + (h - i), \text{ for } 0 \le i < h, \tag{3.9}$$

$$w(L^{2h-i}, R^i) = w(L^h, R^h) + (i - h), \text{ for } h < i \le 2h, \tag{3.10}$$

We note that (3.9) holds for the entire second step, due to Lemma 3.20.

Characterizing the structure of the work function during second step is more complicated. Note that Step 2 consists of $N+1$ iterations, where in each iteration strategies $S(R_2,\ell,2\epsilon)$ and $S(R_1,\ell,2\epsilon)$ are applied. The following auxiliary lemma will be helpful in our arguments about the evolution of the work function during step 2.

Lemma 3.22. *For any $0 \leq i < h$, the optimal schedule to serve c consecutive iterations of Step 2 using $h+i$ servers in R, starting from a configuration (L^{h-i},h,i) and ending up in a configuration $(L^{h-i},h-i',i+i')$, for $0 \leq i' \leq h-i$, has cost $c \cdot 2\epsilon(h-i)+\epsilon \cdot i'$.*

Proof. Consider a schedule $\mathcal{S}$ serving c consecutive iterations of Step 2 using $h+i$ servers in R, starting from a configuration (L^{h-i},h,i). For $j=1,\ldots,c$, let $(L^{h-i},h-a_{j-1},i+a_{j-1})$ be the configuration of $\mathcal{S}$ at the beginning of jth iteration. Let also $(L^{h-i},i+b_j,h-b_j)$ be the configuration of $\mathcal{S}$ at the end of the execution of $S(R_2,h,2\epsilon)$ during jth iteration. At the end of jth iteration, $\mathcal{S}$ is in configuration $(L^{h-i},h-a_j,i+a_j)$ and the $(j+1)$th iteration starts. Note that $0 \leq a_j,b_j \leq h-i$, for all $1 \leq j \leq c$. Clearly, $a_0=0$, since we assume that $\mathcal{S}$ starts at a configuration (L^{h-i},h,i).

We now calculate the minimum cost of $\mathcal{S}$ depending on the values of a_j and b_j. Consider the jth iteration. During executions of $S(R_2,\ell,2\epsilon)$ there are $h-b_j-(i+a_{j-1}))$ servers that move from R_1 to R_2, incurring a movement cost of $\epsilon(h-b_j-i-a_{j-1})$. The cost of serving requests in R_2 is at least $2\epsilon \cdot b_j$, which is incurred during executions of $S(R_2,\ell,2\epsilon)$ for $h-b_j < \ell \leq h$. Similarly, the movement cost for moving servers from R_2 to R_1 is $\epsilon(h-a_j-i-b_j)$ and the cost incurred inside R_1 is at least $2\epsilon a_j$. We get that the total cost of $\mathcal{S}$ is at least

$$\epsilon \cdot \sum_{j=1}^{c}(h-b_j-i-a_{j-1})+(h-a_j-i-b_j)+2a_j+2b_j = \epsilon \cdot \sum_{j=1}^{c} 2h-2i+a_j-a_{j-1}$$

$$= c \cdot 2\epsilon(h-i)+\epsilon \cdot \sum_{j=1}^{c} a_j-a_{j-1} = c \cdot 2\epsilon(h-i)+\epsilon \cdot a_c.$$

By setting $i'=a_c$, the lemma follows. □

We get the following corollary.

Corollary 3.23. *The optimal schedule to serve c consecutive iterations of Step 2 using $h+i$ servers in R, starting from a configuration (L^{h-i},h,i) has cost $c \cdot 2\epsilon(h-i)$, and it ends up in a configuration (L^{h-i},h,i).*

The following definition is going to be helpful for characterizing the changes in the work function during various iterations of Step 2.

Definition 3.24. *An iteration of Step 2 is called good if the following conditions hold:*

(i) During the whole iteration, for all $0 \leq i < h$ *and* $0 < j \leq h - i$*, there is no configuration* $C \in (L^{h-i}, R^{h+i})$ *which is supported by a configuration* $C' \in (L^{h-i-j}, R^{h+i+j})$.

(ii) At the beginning of the iteration, for all $0 \leq i < h$*, we have that* $w(L^{h-i}, R^{h+i}) = w(L^{h-i}, h, i)$ *and for all* $0 < i' \leq h - i$*,*

$$w(L^{h-i}, h - i', i + i') = w(L^{h-i}, h, i) + i' \cdot \epsilon. \tag{3.11}$$

Let us get some intuition behind this definition. The first condition implies that during good iterations the work function values of configurations (L^{h-i}, R^{h+i}) are not affected by configurations (L^{h-j}, R^{h+j}) for $j \neq i$. This makes it easier to argue about $w(L^{h-i}, R^{h+i})$. Moreover, by basic properties, it implies that the WFA does not move servers from L to R (see Observation 3.25 below). Since the WFA has h servers in R when Step 2 starts, our goal is to show that there are too many consecutive good iterations in the beginning of Step 2. This implies that for the largest part of Step 2, the WFA has h servers in R. Then, it suffices to get a lower bound on the cost of the WFA during those iterations. The second condition implies that at the beginning of a good phase, any optimal schedule for configurations (L^{h-i}, R^{h+i}) is at a configuration (L^{h-i}, h, i).

We now state some basic observations that follow from the definition of a good iteration and will be useful in our analysis.

Observation 3.25. *During a good iteration the WFA does not move any server from* L *to* R.

To see why this is true, consider any time during the iteration when the WFA moves from a configuration C to C', such that $C \in (L^{h-i}, R^{h+i})$, for some $0 \leq i < h$. By basic properties of work functions, C is supported by C'. Since the iteration is good, C is not supported by any configuration (L^{h-i-j}, R^{h+i+j}), i.e. $C' \notin (L^{h-i-j}, R^{h+i+j})$. Thus the WFA does not move a server from L to R.

The following observation comes immediately from Corollary 3.23 and the definition of a good iteration.

Observation 3.26. *Consider a good iteration and let* w, w' *be the work function at the beginning and at the end of the iteration respectively. Then for all* $0 \leq i \geq h$ *we have that*

$$w'(L^{h-i}, R^{h+i}) - w(L^{h-i}, R^{h+i}) = 2\epsilon(h - i).$$

Moreover, using Lemma 3.22 for $c = 1$, we get the following observation.

Observation 3.27. *At the end of a good iteration,* (3.11) *holds.*

The next lemma gives us a sufficient condition for an iteration to be good.

Lemma 3.28. *Consider an iteration of Step 2 with initial work function w. If w satisfies* (3.11) *and for all $0 \leq i < h$ and $0 < j \leq h - i$,*

$$w(L^{h-i}, R^{h+i}) + 3\epsilon \cdot h < w(L^{h-i-j}, R^{h+i+j}) + j,$$

then the iteration is good.

Proof. Let t be any time during the iteration with work function w'. For any configuration $C \in (L^{h-i}, R^{h+i})$, a possible schedule to that serves all requests until time t and ends up in C is to simulate the optimal schedule ending up in any configuration (L^{h-i}, R^{h+i}), and then moving at most $i \leq h$ servers within R. The cost of this schedule is at most

$$w'(L^{h-i}, R^{h+i}) + \epsilon h \leq w(L^{h-i}, R^{h+i}) + 2\epsilon(h-i) + \epsilon h \leq w(L^{h-i}, R^{h+i}) + 3\epsilon h, \tag{3.12}$$

where the first inequality holds due to Corollary 3.23. Since the right hand side of (3.12) is smaller than $w(L^{h-i-j}, R^{h+i+j}) + j \leq w'(L^{h-i-j}, R^{h+i+j}) + j$, we get that there is no configuration $C \in (L^{h-i}, R^{h+i})$ which is supported by a configuration (L^{h-i-j}, R^{h+i+j}) and thus the iteration is good. □

Counting good iterations We now count the number of consecutive good iterations in the beginning of Step 2. For all $0 \leq i < h$ and $0 < j \leq h - i$, let $D_{i,j} = w(L^{h-i}, R^{h+i}) - w(L^{h-i-j}, R^{h+i+j})$. The next lemma estimates the change in $D_{i,j}$ after a good iteration.

Lemma 3.29. *Consider a good iteration and let w, w' be the work function at the beginning and at the end of the iteration. Then, for all $0 \leq i < h$ and $0 < j \leq h - i$, we have that*

$$D'_{i,j} = w'(L^{h-i}, R^{h+i}) - w'(L^{h-i-j}, R^{h+i+j}) = D_{i,j} + 2\epsilon j.$$

Proof. By Observation 3.26 we have that

$$\begin{aligned} &w'(L^{h-i}, R^{h+i}) - w'(L^{h-i-j}, R^{h+i+j}) = \\ &= w(L^{h-i}, R^{h+i}) + 2\epsilon(h-i) - w(L^{h-i-j}, R^{h+i+j}) - 2\epsilon(h-i-j) \\ &= w(L^{h-i}, R^{h+i}) - w(L^{h-i-j}, R^{h+i+j}) + 2\epsilon j. \end{aligned}$$

□

Now we are ready to estimate the number of consecutive good iterations after Step 2 starts.

Lemma 3.30. *The first* $\lceil N - \frac{3h}{2} \rceil$ *iterations of Step 2 are good.*

Proof. By Observation 3.27, we have that at the end of a good iteration, (3.11) holds. Thus, by Lemma 3.28, if at the end of the current iteration $D_{i,j} < j - 3\epsilon h$ for all $0 \leq i < h$, $0 < j \leq h - i$, then the next iteration is good.

At the beginning of Step 2, $D_{i,j} = -j$, due to (3.10). From Lemma 3.29 we have that after each good iteration, $D_{i,j}$ increases by $2\epsilon j$. Thus, all iterations will be good, as long as $D_{i,j} < j - 3\epsilon h$. Since

$$(\lceil N - \frac{3h}{2} \rceil - 1) \cdot 2\epsilon j < (N - \frac{3h}{2}) \cdot 2\epsilon j = 2j - 3j\epsilon h \leq 2j - 3\epsilon h,$$

we get that at the beginning of the $(\lceil N - \frac{3h}{2} \rceil)$th iteration of Step 2, $D_{i,j} < -j + 2j - 3\epsilon h = j - 3\epsilon h$. Thus the first $\lceil N - \frac{3h}{2} \rceil$ iterations of Step 2 are good. □

We get the following corollary.

Corollary 3.31. *After* $\lceil N - \frac{3h}{2} \rceil$ *iterations of Step 2, WFA still has* h *servers in* R.

This holds due to the fact that at the beginning of Step 2, the WFA has h servers in R, and from Observation 3.25 we know that the WFA does not move a server from L to R during good iterations.

Work function values for good iterations. Now we proceed to the characterization of the work function during good iterations. Since we want to estimate the cost of the WFA during iterations when it has h servers at R, we focus on configurations (L^h, R^h). Specifically, we need the corresponding versions of Lemmata 3.19, 3.20 and 3.21. Recall that, by definition of a good iteration, the local changes in the values of $w(L^h, R^h)$, define the work function values of all configurations (L^h, R^h).

We begin with the corresponding versions of Lemmata 3.19 and 3.20 for good iterations of Step 2.

Lemma 3.32. *Consider a good iteration of Step 2. Let* t_1 *be the time when* $S(R_2, \ell, 2\epsilon)$ *ends, for* $1 \leq \ell \leq h$ *and* t_2 *the time when* $S(R_2, h, 2\epsilon)$ *ends. At any time* $t \in [t_1, t_2]$, *we have that* $w(L^h, h - \ell + 1, \ell - 1) = w(L^h, h - \ell, \ell) + \epsilon$.

Proof. The proof is same as proof of Lemma 3.19 with replacing (L^{k-i}, R^i) by $(L^h, h - i, i)$, scaling distances by ϵ and noting that by (3.11), any optimal schedule for configurations (L^h, R^h) starts the iteration from a configuration $(L^h, h, 0)$. □

Lemma 3.33. *Consider a good iteration of Step 2. Let t_1 be the time when $S(R_2,\ell,2\epsilon)$ ends, for $1 \le \ell \le h$ and t_2 the time when $S(R_2,h,2\epsilon)$ ends. At any time $t \in [t_1,t_2]$, we have that, for any $1 \le i < \ell$, $w(L^h,h-i,i) = w(L^h,h-\ell,\ell) + \epsilon(\ell - i)$.*

The proof of this lemma is same as the proof of Lemma 3.20, i.e. by backwards induction on i. We just need to replace (L^{k-i},R^i) by $(L^h,h-i,i)$ and scale all distances by ϵ.

The next two lemmata are the analogues of Lemma 3.21 for good iterations of Step 2, and characterize the structure of the work function at the end of $S(R_2,\ell,2\epsilon)$ and $S(R_1,\ell,2\epsilon)$, for $1 \le \ell \le h$.

Lemma 3.34. *Consider a good iteration during Step 2. At the moment when the execution of $S(R_2,\ell,2\epsilon)$ ends, for $1 \le \ell \le h$, the following two things hold:*

1. *For all $0 \le i < \ell$, $w(L^h,h-i,i) = w(L^h,h-\ell,\ell) + \epsilon(\ell - i)$*
2. *For all $\ell < i \le h$, $w(L^h,h-i,i) = w(L^h,h-\ell,\ell) + \epsilon(i - \ell)$*

Proof. For $0 \le i < \ell$, the lemma follows from Lemma 3.33. For $i \ge \ell$ the proof is same as proof of Lemma 3.21 by replacing $w(L^{k-i},R^i)$ by $w(L^h,h-i,i)$ and using the fact that at the beginning of the iteration $w(L^h,h-i,i) = w(L^h,h,0) + \epsilon \cdot i$. □

Lemma 3.35. *Consider a good iteration during Step 2. At the moment when an execution of $S(R_1,\ell,2\epsilon)$ ends, for $1 \le \ell \le h$, the following two things hold:*

1. *For all $0 \le i < \ell$, $w(L^h,i,h-i) = w(L^h,\ell,h-\ell) + \epsilon(\ell - i)$*
2. *For all $\ell \le i \le h$, $w(L^h,i,h-i) = w(L^h,\ell,h-\ell) + \epsilon(i - \ell)$*

Proof. The proof is completely symmetric to the proof of Lemma 3.34. Note that, due to Lemma 3.34, after $S(R_2,h,2\epsilon)$ ends, i.e at the beginning of $S(R_1,1,2\epsilon)$, we have that $w(L^h,R^h) = w(L^h,0,h)$ and that $w(L^h,h-i,i) = w(L^h,0,h) + \epsilon(h-i)$, or equivalently

$$w(L^h,i,h-i) = w(L^h,0,h) + \epsilon \cdot i. \tag{3.13}$$

Note that (3.13) is symmetric to $w(L^h,h-i,i) = w(L^h,h,0) + \epsilon \cdot i$, which we used in proof of Lemma 3.34, so we can apply the symmetric proof. □

End of the phase: We now show that at the end of the phase the work function has the desired structure and that the WFA has all its $2h$ servers in R, so a new right-to-left phase may start.

Lemma 3.36. *After N iterations of Step 2, $w(L^{h-i}, R^{h+i}) = w(L^0, R^{2h}) + (h-i) = 3h - i$, for all $1 \leq i \leq h-1$.*

Proof. By basic properties of work functions, $w(L^{h-i}, R^{h+i}) \leq w(L^0, R^{2h}) + (h-i)$. Assume for contradiction that there exists some i such that $w(L^{h-i}, R^{h+i}) < w(L^0, R^{2h}) + (h-i)$. If there are many such i's, we prove the contradiction for the largest one and then we can repeat. This assumption implies that for any $j > i$ we have that

$$\begin{aligned} w(L^{h-j}, R^{h+j}) + (j-i) &= w(L^0, R^{2h}) + (h-j) + (j-i) \\ &= w(L^0, R^{2h}) + (h-i) > w(L^{h-i}, R^{h+i}). \end{aligned}$$

Thus, the configuration C that defines the value of $w(L^{h-i}, R^{h+i})$ is not supported by any configuration (L^{h-j}, R^{h+j}) for $j > i$. Let $\mathcal{S}$ be the schedule that defines the value of $w(L^{h-i}, R^{h+i}) = W(C)$. Since C is not supported by any configuration (L^{h-j}, R^{h+j}) for $j > i$, $\mathcal{S}$ uses exactly $h+i$ servers in R. We now evaluate the cost of $\mathcal{S}$. Moving $h+i$ servers from L to R costs $h+i$. Serving Step 1 using $h+i$ servers in R costs 0. By Lemma 3.22, the optimal way to serve N iterations of Step 2 using $h+i$ servers in R has cost $2N \cdot \epsilon(h-i) = 2(h-i)$. Overall we get that $w(L^{h-i}, R^{h+i}) = h + i + 2(h-i) = 3h - i$.

On the other hand, we have that $w(L^0, R^{2h}) = 2h$ for the whole phase. Thus

$$w(L^{h-i}, R^{h+i}) = 3h - i = w(L^0, R^{2h}) + (h-i),$$

contradiction. □

By setting $i' = h - i$, Lemma 3.36 gives us that

$$w(L^{i'}, R^{2h-i'}) = w(L^0, R^{2h}) + i', \text{ for } 1 \leq i' \leq h. \tag{3.14}$$

Moreover, by setting $i' = 2h - i$ in (3.9), we get that for $h < i' \leq 2h$,

$$w(L^{i'}, R^{2h-i'}) = w(L^h, R^h) + i' - h = w(L^0, R^{2h}) + h + i' - h = w(L^0, R^{2h}) + i', \tag{3.15}$$

where the first equality holds due to (3.14). From (3.14), (3.15), we get that after N iterations of Step 2,

$$w(L^{i'}, R^{2h-i'}) = w(L^0, R^{2h}) + i', \text{ for all } 1 \leq i' \leq 2h. \tag{3.16}$$

Note that $w(L^0, R^{2h})$ does not change during the whole phase, since exactly $2h$ points of R are requested. Thus, (3.16) holds until the end of the phase and then a new right-to-left phase may start, satisfying the assumption about the initial work function. Last, to see that at the end of the phase the WFA has all its $2h$ servers in R, assume that after N iterations of Step 2, it is in some configuration $(L^{i'}, R^{2h-i'})$ for some $0 \leq i' \leq (h-1)$. Since (3.16) holds until the end of the phase, during the next iteration, the WFA moves to the configuration (L^0, R^{2h}).

Bounding Costs

We now bound the online and offline costs. We begin with Step 1.

Lemma 3.37. *During the time when the WFA uses $\ell < h$ servers in R, it incurs a cost of at least $(2 - 2\epsilon') \cdot \ell$.*

Proof. By construction of the phase, when the execution of $S(R_1, \ell + 1, 2)$ starts, the WFA has ℓ servers in R. We evaluate the cost of the WFA during the part of $S(R_1, \ell + 1, 2)$ when it has ℓ servers in R. Let t be the time when the WFA moves its $(\ell + 1)$th server to R. Let w and w' be the work functions at the beginning of $S(R_1, \ell + 1, 2)$ and at time t respectively. We have that,

$$w'(L^{k-\ell-1}, R^{\ell+1}) = w(L^{k-\ell-1}, R^{\ell+1}) = w(L^{k-\ell}, R^{\ell}) + 1, \tag{3.17}$$

where the first equality comes from the fact that there are exactly $\ell + 1$ points of R requested since the beginning of the phase and the second by Lemma 3.21. Let C be the configuration of the WFA at time $t' - 1$. At time t, the WFA moves a server by a distance of 1, thus by basic properties of work functions, we have that $w'(C) = w'(L^{k-\ell-1}, R^{\ell+1}) + 1$. This combined with (3.17) gives that

$$w'(C) = w(L^{k-\ell}, R^{\ell}) + 2. \tag{3.18}$$

Let $X \in (L^{k-\ell}, R^{\ell})$ be a configuration such that (i) $w'(X) = w'(L^{k-\ell}, R^{\ell})$ and (ii) X, C have their $k - \ell$ servers of L at the same points. Note that $X, C \in (L^{k-\ell}, \ell, 0)$, since R_1 is the only subtree of R where requests have been issued. Since both X and C have ℓ servers in R_1, and only $\ell + 1$ leaves of R_1 have been requested, we get that X and C can differ in at most one point. In other words, $d(X, C) \leq \epsilon'$. By basic properties of work functions we get that

$$w'(C) \leq w'(X) + d(X, C) \leq w'(L^{k-\ell}, R^{\ell}) + \epsilon'. \tag{3.19}$$

From (3.18), (3.19) we get that

$$w'(L^{k-\ell}, R^{\ell}) - w(L^{k-\ell}, R^{\ell}) \geq w'(C) - \epsilon' - (w'(C) - 2) = 2 - \epsilon'. \tag{3.20}$$

Let $\mathcal{S}_\ell$ be the optimal schedule to serve all requests from the beginning of $S(R_2, \ell + 1, 2\epsilon)$ until time t, using ℓ servers starting at $t_1, \ldots, t_\ell$. From (3.20) we have that $\text{cost}(\mathcal{S}_\ell) \geq 2 - \epsilon'$. All requests are located in $\ell + 1$ leaves of an elementary subtree (which is equivalent to the paging problem) in an adversarial way. It is well known (see e.g. [19]) that the competitive ratio of any online algorithm for the paging problem for such a sequence is at least ℓ, i.e. it has cost at least $cost(\mathcal{S}_\ell) - \ell \cdot \epsilon'$, where $\ell \cdot \epsilon'$ is the allowed additive term. This implies that the cost incurred by the WFA is at least

$$\ell \cdot \text{cost}(\mathcal{S}_\ell) - \ell \cdot \epsilon' \geq \ell(2 - \epsilon') - \ell \cdot \epsilon' = (2 - 2\epsilon') \cdot \ell. \qquad \square$$

Lemma 3.17 *(restated) During Step 1,* ALG $\geq h^2 - \epsilon' \cdot h(h-1)$ *and ADV*$= h$.

Proof. Clearly, ADV$= h$; the adversary moves h servers by distance 1 and then serves all requests at zero cost. By lemma 3.37 we get that WFA incurs a cost of at least $\sum_{\ell=1}^{h-1}(2-2\epsilon') \cdot \ell$ inside R. Moreover, it pays a movement cost of h to move its h servers from L to R. Overall, we get that the online cost during Step 1 is $(2-2\epsilon')\sum_{\ell=1}^{h-1}\ell + h = 2\sum_{\ell=1}^{h-1}\ell + h - 2\epsilon'\sum_{\ell=1}^{h-1}\ell = h^2 - \epsilon' \cdot h(h-1)$. □

We now focus on Step 2. We charge the WFA only for the first $\lceil N - \frac{3h}{2} \rceil$ iterations (when it uses h servers in R, due to Corollary 3.31), plus the movement cost of the extra servers from L to R.

Lemma 3.38. *Consider a good iteration of Step 2 such that the WFA uses only h servers in R. During the time interval that WFA serves requests in subtree R_1 or R_2 using ℓ servers, it incurs a cost of at least $2(\epsilon - \epsilon')\ell$.*

Proof. The proof is similar to the proof of Lemma 3.37, with the following modifications. We scale distances by ϵ, since now the servers move from R_1 to R_2 (distance ϵ) instead of moving from L to R_1 (distance 1). We charge the WFA for serving requests in R_2 using ℓ servers during executions of $S(R_2, \ell + 1, 2\epsilon)$, using Lemma 3.34 (instead of Lemma 3.21) and replacing (L^{k-i}, R^i) by $(L^h, h-i, i)$. Similarly, we charge the WFA for serving requests in R_1 using ℓ servers during executions of $S(R_1, \ell + 1, 2\epsilon)$ using Lemma 3.35. □

Lemma 3.39. *Consider a good iteration of Step 2 such that the WFA uses only h servers in R to serve the requests. The cost of the WFA in such an iteration is at least $2\epsilon \cdot h^2 - 2\epsilon' \cdot (h-1) \cdot h$.*

Proof. From Lemma 3.38, we get that the cumulative cost for all executions of S in R_2 is $\sum_{\ell=1}^{h-1} 2(\epsilon - \epsilon')\ell + \epsilon \cdot h$, where the last $\epsilon \cdot h$ term counts the cost of moving h servers from R_1 to R_2. Similarly, the WFA incurs the same cumulative cost during executions of S in R_1. We get that the total cost of the WFA during the iteration is at least

$$2 \cdot \Big(\sum_{\ell=1}^{h-1} 2(\epsilon - \epsilon')\ell + \epsilon \cdot h \Big) = 2\epsilon \cdot h + 4\sum_{\ell=1}^{h-1} \epsilon \cdot \ell - 4\sum_{\ell=1}^{h-1} \epsilon' \cdot \ell$$

$$2\epsilon\big(h + (h-1)h\big) - 2\epsilon' \cdot (h-1) \cdot h = 2\epsilon \cdot h^2 - 2\epsilon' \cdot (h-1) \cdot h. \qquad \square$$

Lemma 3.18 *(restated) During Step 2,* ALG $\geq 2h^2 + h - 3\epsilon h^3 - 2\frac{\epsilon'}{\epsilon} \cdot (h-1) \cdot h$ *and* ADV $= (2+\epsilon)h$.

Proof. Second step consists of $N + 1 = 1/\epsilon + 1$ iterations, where in each one of them the adversary incurs a cost of $2\epsilon \cdot h$. Thus the offline cost is $(2 + 2\epsilon)h$.

Let us now count the online cost during Step 2. By Corollary 3.31, there are at least $\lceil N - \frac{3h}{2} \rceil$ iterations such that the WFA has h servers in R. By lemma 3.39 we get that during each one of those iterations, the online cost is at least $2\epsilon \cdot h^2 - 2\epsilon' \cdot (h-1) \cdot h$. For the rest of Step 2, the WFA incurs a cost of at least h, as it moves h servers from L to R. We get that overall, during Step 2, the cost of WFA is at least

$$\begin{aligned}
&(N - \frac{3h}{2}) \cdot (2\epsilon \cdot h^2 - 2\epsilon' \cdot (h-1) \cdot h) + h \\
=&2N\epsilon \cdot h^2 - 2N\epsilon' \cdot (h-1) \cdot h - 3\epsilon h^3 + 3\epsilon' h^2 (h-1) + h \\
\geq&2h^2 + h - 3\epsilon h^3 - 2\frac{\epsilon'}{\epsilon} \cdot (h-1) \cdot h.
\end{aligned}$$

□

Remaining Proofs

We now prove Lemma 3.19, whose proof was omitted earlier. First, we state an observation which follows directly from the definition of the left-to-right phase.

Observation 3.40. *Let t be any time during a left-to-right phase with work function w. If $w(L^{k-i}, R^i) = w(L^{k-j}, R^j) + (j-i)$ for some $0 \leq i < j \leq k$, then for all j' such that $i \leq j' \leq j$ we have that $w(L^{k-j'}, R^{j'}) = w(L^{k-j}, R^j) + (j-j')$.*

To see why this is true, note that by basic properties $w(L^{k-j'}, R^{j'}) \leq w(L^{k-j}, R^j) + (j-j')$. Moreover, if strict inequality holds, we have that $w(L^{k-j'}, R^{j'}) + (j'-i) < w(L^{k-j}, R^j) + (j-j') + (j'-i) = w(L^{k-i}, R^i)$. We get that $w(L^{k-i}, R^i) - w(L^{k-j'}, R^{j'}) > j' - i$, a contradiction.

Lemma 3.19 *(restated) Consider a left-to-right phase. At any time after the end of strategy $S(R_1, \ell, 2)$, for $1 \leq \ell \leq h$, we have that $w(L^{k-\ell+1}, R^{\ell-1}) = w(L^{k-\ell}, R^\ell) + 1$.*

Proof. We prove the lemma by induction. For the base case $\ell = 1$, the lemma is clearly true since any schedule serving requests uses at least one server at R. We prove the lemma for $2 \leq \ell \leq h$, assuming that it is true for all values $1, \ldots, \ell - 1$.

Let t be any time after the end of strategy $S(R_1, \ell, 2)$, for $1 < \ell \leq h$ with work function w. By basic properties of work functions, $w(L^{k-\ell+1}, R^{\ell-1}) \leq w(L^{k-\ell}, R^\ell) + 1$. We will assume that $w(L^{k-\ell+1}, R^{\ell-1}) < w(L^{k-\ell}, R^\ell) + 1$ and obtain a contradiction. The lemma then follows.

Assume that $w(L^{k-\ell+1}, R^{\ell-1}) < w(L^{k-\ell}, R^\ell) + 1$. Let $\mathcal{S}_{\ell-1}$ and $\mathcal{S}_\ell$ be the schedules that define the values of $w(L^{k-\ell+1}, R^{\ell-1})$ and $w(L^{k-\ell}, R^\ell)$ respectively. Let C be a configuration $(L^{k-\ell+1}, R^{\ell-1})$ such that $w(C) = w(L^{k-\ell+1}, R^{\ell-1})$. By Observation 3.40, C is not supported by any configuration (L^{k-i}, R^i), for $i \geq \ell$. Also, by the inductive hypothesis, C is not supported

by configurations (L^{k-i}, R^i) for $i < \ell - 1$. We get that C is in the support of w. Without loss of generality $\mathcal{S}_{\ell-1}$ is lazy, i.e. it moves a server only to serve a request (this is a standard property, explained in the proof of Theorem 3.4). Thus, from the beginning of the phase until time t, $\mathcal{S}_{\ell-1}$ never moves a server from R to L. We get that $\mathcal{S}_{\ell-1}$ uses exactly $\ell - 1$ servers in R. On the other hand, by construction of the phase, $\mathcal{S}_\ell$ has at least ℓ servers in R during the execution of $S(R_1, \ell, 2)$.

We now compare the costs of $\mathcal{S}_{\ell-1}$ and $\mathcal{S}_\ell$. Before $S(R_1, \ell, 2)$ starts, the schedules are the same, since there are exactly $\ell - 1$ points of R requested. During $S(R_1, \ell, 2)$, $\mathcal{S}_\ell$ incurs a cost of 1 to move a server from L to R and then serves all requests at zero cost, while $\mathcal{S}_{\ell-1}$ incurs a cost of at least 2 (by construction of the strategy). When $S(R_1, \ell, 2)$ ends, the set of leaves of R_1 occupied by servers of $\mathcal{S}_{\ell-1}$ is a subset of the leaves occupied by servers of $\mathcal{S}_\ell$. Thus, the cost incurred by $\mathcal{S}_\ell$ after the end of $S(R_1, \ell, 2)$ is smaller or equal to the cost incurred by $\mathcal{S}_{\ell-1}$ during the same time interval. Overall, we get that $\text{cost}(\mathcal{S}_{\ell-1}) \geq \text{cost}(\mathcal{S}_\ell) + 1$, i.e., $w(L^{k-\ell+1}, R^{\ell-1}) \geq w(L^{k-\ell}, R^\ell) + 1$, a contradiction. □

3.6 Tight Bounds for WFA on the Line

In this section we show that the lower bound of section 3.5.2 for the WFA can also be applied on the line. Moreover we show that there exists a matching upper bound.

Lower Bound. The lower bound strategy is the same as the one described for depth-3 HSTs, we just need to replace subtrees by line segments. More precisely, the lower bound strategy is applied in an interval I of length 1. Let L and R be the leftmost and rightmost regions of I, of length $\epsilon/2$, for $\epsilon \ll 1$. Similarly, L_1, L_2 and R_1, R_2 are smaller regions inside L and R respectively. Again, the distance between L_1 and L_2 (R_1 and R_2 resp.) is much larger than their length. In each of those regions we focus on h points, where the distance between 2 consecutive points is $\epsilon' \ll \epsilon$. Whenever the adversary moves to such a region, it places its servers those h equally spaced points inside it. Similarly to the proof of Theorem 3.2, we can get a lower bound $h + 1/3$ on the competitive ratio of the WFA on the line when $k = 2h$ (the changes affect only the dependence on terms involving ϵ and ϵ', which are negligible).

Upper Bound We now show an upper bound on the cost of WFA for the (h, k)-server problem in the line, which is implied by combining results of [17, 51].

Most known upper bounds for the WFA do not bound the algorithm's actual cost directly. Instead, they bound its *extended cost*, defined as the maximum increase of the work function value for any single configuration. To define it formally, let w and w' be the work functions before and after serving the request at time t. Then, if M denotes the metric space, the extended cost at time t is defined as follows:

$$\text{ExtCost}(t) = \max_{X \in M^k} \{w'(X) - w(X)\} \ ,$$

and the total extended cost of a sequence of requests is

$$\text{ExtCost} = \sum_t \text{ExtCost(t)} \ .$$

For a given sequence of requests, let WFA_i and OPT_i denote the overall cost incurred by WFA and OPT using i servers respectively. Similarly, let ExtCost_i denote the total extended cost over configurations of i servers. Chrobak and Larmore [31] showed that the extended cost satisfies the following inequality:

$$\text{WFA}_i + \text{OPT}_i \leq \text{ExtCost}_i \ . \tag{3.21}$$

In [17] it was shown that WFA with h servers is h-competitive in the line by proving the following inequality:

$$\text{ExtCost}_h \leq (h+1)\,\text{OPT}_h + O(1) \tag{3.22}$$

Moreover, Koutsoupias [51] showed that the extended cost is a non-increasing function of the number of servers, which implies

$$\text{ExtCost}_k \leq \text{ExtCost}_h \tag{3.23}$$

for all request sequences. Putting (3.21), (3.22) and (3.23) together, we get

$$\text{WFA}_k + \text{OPT}_k \leq \text{ExtCost}_k \leq \text{ExtCost}_h \leq (h+1)\,\text{OPT}_h + O(1) \ ,$$

which implies that WFA_k is $(h+1)$-competitive. In fact, a slightly stronger inequality holds:

$$\text{WFA}_k \leq (h+1)\,\text{OPT}_h - \text{OPT}_k + O(1) \ .$$

Matching Bounds. We now show that the lower and upper bounds above are matching for $k = 2h$. In particular, it suffices to show that for our lower bound instance $(h + 1/3)\text{OPT}_h$ goes arbitrary close to $(h+1)\text{OPT}_h - \text{OPT}_k$, or equivalently

$$\frac{2\mathrm{OPT}_h}{3} \to \mathrm{OPT}_k \tag{3.24}$$

As we showed in the proof of theorem 3.2, for every phase of the lower bound strategy, $\mathrm{OPT}_h = (3 + 2\epsilon) \cdot h$. Moreover it is clear that $\mathrm{OPT}_k = 2h$; during a phase the minimum work function value using $k = 2h$ servers increases by exactly $2h$. We get that

$$\frac{2\mathrm{OPT}_h}{3} = \frac{2 \cdot (3 + 2\epsilon) \cdot h}{3} = 2h\frac{3 + 2\epsilon}{3} \to 2h = \mathrm{OPT}_k.$$

3.7 Concluding Remarks

Several intriguing open questions remain, and we list some of them here.

Open Problem 3.1. *Is the dependence on d in Theorem 3.3 necessary? In other words, can we get a $O(1)$-competitive algorithm for depth-d trees for $k \gg h$?*

While Theorem 3.4 gives a separation between depth-1 and depth-2 trees, it is unclear whether a lower bound which increases substantially with depth is possible. Note that a lower bound of $g(d)$ for depth d, where $g(d) \to \infty$ as $d \to \infty$ (provided that h is large enough), would be very surprising. This would imply that there is no $O(1)$-competitive algorithm for general metric spaces.

Open Problem 3.2. *Can we get an $o(h)$-competitive algorithm for other metric spaces?*

An interesting metric is the line. Here, we showed that both DC and the WFA have competitive ratio $\Omega(h)$, and we do not even know any good candidate algorithm. Designing an algorithm with such a guarantee would be very interesting. Very recently, Coester et. al. [34] showed a lower bound of 3.146 on the competitive ratio of deterministic algorithms for the (h, k)-server problem in the line (for sufficiently large h), even if $k/h \to \infty$. This improves our lower bound of 2.4 for general metric spaces, and remains up to date the best known lower bound for the (h, k)-server problem.

Chapter 4

Competitive Algorithms for Generalized k-Server in Uniform Metrics

4.1 Introduction

In the *generalized k-server problem*, we are given k-servers $s_1, \ldots, s_k$, where server s_i lies in a metric space M_i with distance function d_i. At each time step a request $r = (r_1, r_2, \ldots, r_k) \in M_1 \times \ldots \times M_k$ arrives and must be served by moving some server s_i to the point $r_i \in M_i$. The goal is to minimize the total distance traveled by the servers.

The generalized k-server problem is a substantial generalization of the standard k-server problem. In particular, the k-server problem corresponds to the 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. [55]. For that reason, it is widely believed that a deeper understanding of this problem should lead to powerful new techniques for designing online algorithms [55, 63]. According to Koutsoupias and Taylor [55], this problem "may act as a stepping stone towards building a robust (and less ad hoc) theory of online computation".

Previous Work

In their seminal paper, Koutsoupias and Taylor [55] studied the special case where $k = 2$ and both the metrics M_1 and M_2 are lines. This is called the CNN

problem and it has attracted a lot of attention [4, 28, 48, 47]. They showed that, even for this special case, the problem is qualitatively very different from the standard k-server problem. In particular they showed that many successful k-server algorithms or their natural generalizations are not competitive. They also showed that no memoryless randomized algorithm is competitive, while it is well-known that for the standard k-server problem in the line there exists a k-competitive (against adaptive online adversaries) randomized memoryless algorithm [35]. A similar result with a simpler analysis was discovered independently by Chrobak and Sgall [33].

Lower Bounds: For the CNN problem, Koutsoupias and Taylor [55] showed that the competitive ratio of any deterministic algorithm is at least 10.12. For uniform metrics and arbitrary k, they 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 $2^{2^{\Omega(k)}}$ [12], 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. $M_i = w_i \cdot M$ 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 [65, 63, 64]. In a breakthrough result, Sitters and Stougie [65] obtained a $O(1)$-competitive algorithm for $k = 2$ in any metric space. Recently, Sitters [63] 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.

Our Results

We consider the generalized k-server problem on uniform metrics and obtain the first $f(k)$-competitive algorithms for general k, whose competitive ratios almost match 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.

[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 $n = \max_{i=1}^{k} |M_i|$.

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 4.1. *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 [55] (we describe this instructive and simple lower bound instance in Section 4.5 for completeness).

To prove Theorem 4.1, 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 feasible for all requests during this phase can only reduce. The proof is based on a general combinatorial argument about how the set of feasible states evolves as requests arrive. In particular, for this problem we show that any sequence of requests that causes the set of feasible states to strictly reduce at each step, can have length at most 2^k until all states become infeasible.

Interestingly, this argument is based on a novel application of the polynomial or the rank method from linear algebra [49, 58, 45]. While the rank method has led to some spectacular recent successes in combinatorics and computer science [36, 37], we are not aware of any previous applications to online algorithms. We feel that 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 4.2. *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 4.3, 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 4.2.

In Section 4.5, we also give a related lower bound. In particular, we note that an $\Omega(k/\log^2 k)$ lower bound on the competitive ratio of any randomized algorithm follows directly by combining the lower bound instance of [55] with the results of [16].

Finally, we consider the weighted uniform metric case.

Theorem 4.3. *There is a $2^{2^{k+3}}$ competitive algorithm for generalized k-server on weighted uniform metrics.*

Theorem 4.3 follows by observing that a natural modification of an algorithm due to Fiat and Ricklin [43] 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 [43], with some arguments streamlined and an improved competitive ratio[2]. Finally, note that the $2^{2^{\Omega(k)}}$ lower bound [12] for weighted k-server on uniform metrics implies that the competitive ratio in Theorem 4.3 is essentially optimal.

4.2 Deterministic Algorithm for Uniform Metrics

In this section we prove Theorem 4.1. 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 $\{1, \ldots, n\}$. 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 $q^t = (q_1^t, \ldots, q_k^t)$ where for each $i \in [k]$, $q_i^t \in [n]$ denotes the location of server i. Let $r^t = (r_1^t, \ldots, r_k^t)$ denote the request vector at time t. We need to find a state with the following property:

Definition 4.4. *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}.

We describe a generic algorithm below that works in phases in Algorithm 2. Each phase is independent of all other phases. For convenience we reset the time at the beginning of each phase, i.e. we assume that the first request of the phase is at time 1.

We will show that during each phase the offline optimum moves at least once and hence pays at least 1, while the online algorithm changes its state at most 2^k times and hence pays at most $k2^k$ (as the Hamming distance between any two states is at most k). It follows that for the whole request sequence,

[2] It was first pointed out to us by Chiplunkar [26] that the competitive ratio $2^{2^{4k}}$ claimed in [43] can be improved to $2^{2^{k+O(1)}}$.

our algorithm pays at most $c^* \cdot k2^k + k2^k$, where c^* denotes the optimal cost. Here the additive term $k2^k$ accounts for the last (possibly unfinished) phase.

Algorithm 2: 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 $r^1, \ldots, r^t$ **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 $q^{t+1} = q^t$.

We call this algorithm *generic* as it can pick any arbitrary point q as long as it is feasible for all requests of the current phase $r^1, \ldots, r^t$. Note that this algorithm captures a wide variety of natural algorithms including (variants) of the Work Function Algorithm.

Fix some phase that we wish to analyze, and let ℓ denote its length. Without loss of generality, we can assume that any request r^t causes the algorithm to move, i.e. q^t is not feasible for r^t (removing requests where q^t is feasible does not affect the online cost, and can only help the offline adversary). So the online algorithm moves exactly ℓ times. Moreover, the adversary must move at least once during the phase as no location exists that satisfies all the requests $r^1, \ldots, r^\ell$ that arrive during the phase.

It suffices to show the following.

Theorem 4.5. *For any phase as defined above, its length satisfies $\ell \leq 2^k$.*

Proof. We use the rank method. Let $x = (x_1, \ldots, x_k), y = (y_1, \ldots, y_k)$ be points in $\mathbb{R}^k$, 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 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 p on q^t and $r^{t'}$

Claim 4.6. *M is an upper triangular matrix with non-zero diagonal.*

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, this implies that M has non-zero determinant and has full rank, $\mathrm{rk}(M) = \ell$.

However, we can use the structure of p to show that the rank of M is at most 2^k in a fairly straight manner. In particular, we give an explicit factorization of M as $M = AB$, where A is $\ell \times 2^k$ matrix and B is a $2^k \times \ell$ matrix. Clearly, 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 $\mathrm{rk}(M) \leq 2^k$. It remains to show the factorization.

Indeed, if we express $p(x, y)$ in terms of its 2^k monomials, we can write

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

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

Now, 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

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

Similarly, let B be the $2^k \times \ell$ matrix with rows indexed by subsets $S \in 2^{[k]}$ and columns indexed by time t'. We define

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

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

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

and hence $M = AB$ as claimed. □

We remark that an alternate way to view this result is that the length of any request sequence that causes the set of feasible states to strictly decrease at each step can be at most 2^k.

4.3 Randomized Algorithm for Uniform Metrics

In this section we give a randomized algorithm for uniform metrics with competitive ratio $O(k^3 \log k)$. As in the previous section, we assume that all metrics have the same number of points $|M_i| = n$. We also call a state feasible according to Definition 4.4.

A natural way to randomize the algorithm from the previous section, 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 [20] 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 4.2 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 $t = 1, \ldots, \ell$, 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 Lemmata 4.7 and 4.9 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 a 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. We call *dimension* the number of free coordinates in the space $S(v)$ 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 configurations satisfying r.

Lemma 4.7. *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 $S_1, \ldots, S_d$, each of dimension $d-1$, such that we have $S \cap F(r) = S_1 \cup \cdots \cup S_d$.*

Note that $|S| = n^d$ and $|S_i| = n^{d-1}$, i.e $|S_i| = \frac{1}{n}|S|$ for each $i = 1, \ldots, d$.

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

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

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

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 $i = d+1, \ldots, k$. 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 $i \in \{1, \ldots, d\}$ and therefore $c' \in S(v_i)$. □

Spaces of feasible configurations. At each time t during a phase, we maintain a set $\mathcal{F}^t$ of spaces containing configurations which were feasible with respect to the requests $r^1, \ldots, r^t$. In the beginning of the phase, we set $\mathcal{F}^1 = \{(r^1_1, *, \ldots, *), \ldots, (*, \ldots, *, r^1_k)\}$, 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 $d > 0$ which contains some infeasible configuration by $S_1, \ldots, S_d$ according to Lemma 4.7. The following observation follows easily from Lemma 4.7.

Observation 4.8. *Let us consider a phase with requests $r^1, \ldots, r^\ell$. A configuration c is feasible with respect to the requests $r^1, \ldots, r^t$ 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 $3(k+1)!$. This is worse than the competitive ratio of Algorithm 2 but will be very useful to obtain our randomized algorithm.

We describe the alternative deterministic algorithm in Algorithm 3. In contrast to the previous section, the algorithm is not defined in terms of phases, and each time t corresponds to the tth request of the whole request sequence. The partition of the request sequence into phases is made only for the analysis. In particular, if a request r^t causes $\mathcal{F}^t$ to become empty, then we say that the phase ends, we set $\mathcal{F}^t = \{(r_1^t, *, \ldots, *), \ldots, (*, \ldots, *, r_k^t)\}$ and a new phase starts. To serve a request at time t, the algorithm chooses some space $Q^t \in \mathcal{F}^t$ and moves to an arbitrary state $q^t \in Q^t$. 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. While, this is not an optimal behavior, a primitive exploitation of the structure of $\mathcal{F}^t$ already gives a reasonably good algorithm.

Algorithm 3: 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 $S_1, \ldots, S_d$ according to Lemma 4.7
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 $Q^{t-1} \in \mathcal{F}^t$ **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.

Lemma 4.9. *Let us consider a phase with requests $r^{t_1}, \ldots, r^{t_\ell}$. Then $\bigcup_{t=t_1}^{t_\ell} \mathcal{F}^t$ contains at most $k!/d!$ spaces of dimension d.*

Note that this lemma implies that the competitive ratio of Algorithm 3 is at most $k \cdot k! \cdot \sum_{d=0}^{k-1} \frac{1}{d!} \leq 3kk! \leq 3(k+1)!$, where the first factor k comes for the fact that at each request the algorithm moves, it incurs a cost of at most k.

Proof of Lemma 4.9. 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 4.7, all spaces added later have strictly lower dimension.

By the update rule on $\mathcal{F}^t$, each $(d-1)$-dimensional space is created from some d-dimensional space already present in $\bigcup_{t=t_1}^{t_\ell} \mathcal{F}^t$. By the inductive hypothesis, there could be at most $k!/d!$ distinct d-dimensional spaces and Lemma 4.7

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 $\bigcup_{t=t_1}^{t_\ell} \mathcal{F}^t$. □

In fact, Algorithm 3 can be modified such that instead of choosing Q^t and q^t arbitrarily, it chooses them in a way which ensures that it moves at most one server in each request. This implies a slightly improved competitive ratio of $3k!$. We omit those details since we already know a deterministic algorithm with competitive ratio $k2^k = o(k!)$ and our main focus here is to use the structure of $\mathcal{F}^t$ in order to get a randomized algorithm.

Randomized algorithm. Now we transform Algorithm 3 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[3].

Algorithm 4: 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 4.7
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 4.8. Here is the key property about the state of ALG.

[3]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 $k \ll n$.

Lemma 4.10. *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. We prove the lemma by induction on t. For the base case $t = 1$, at the first request the algorithm chooses a space Q^1 from $\mathcal{M}^1$ uniformly at random and the lemma follows.

Assume that the lemma is true for time $t-1$. We will show that it holds for time t. If ALG moved at time t, the statement follows trivially, since Q^t was chosen from $\mathcal{M}^t$ uniformly at random. It remains to consider the case 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 $\mathcal{M}^t$. □

Proof of Theorem 4.2. Let cost(ALG) and cost(OPT) denote the cost of the algorithm and the optimal solution respectively. 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 $\text{cost(OPT)} \geq (N-1)$. On the other hand, the expected online cost is at most,

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

where c denotes the expected cost of ALG in one phase. This implies that ALG is c-competitive.

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. At 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.

(i) 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

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

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 4.10. Thus the expected cost of the algorithm is at most $k \cdot b/|\mathcal{M}^{t-1}|$, which is at most $k \cdot (-\Delta\Phi)$.

(ii) In the second case, we have $m_t < m_{t-1}$. By Lemma 4.9, we know that $|\mathcal{M}^t| \leq k!/m_t!$ and hence

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

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 $k \cdot (-\Delta\Phi)$. □

4.4 Algorithm for Weighted Uniform Metrics

In this section we prove Theorem 4.3. Our algorithm is a natural extension of the algorithm of Fiat and Ricklin [43] 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,

$$4(c(i)+1) \cdot c(i) \leq 8c(i)^2 = c(i+1). \tag{4.1}$$

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

$$R_i = 8 \cdot c(i) \cdot R_{i-1}. \tag{4.2}$$

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 $R_k \cdot R_{k-1} < (R_k)^2 = 2^{2^{k+3}}$ 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.

4.4.1 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.

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. $P = \{p_1, \ldots, p_{c(i)}\}$.

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 $p \in M_i$, $m(p) \leftarrow$ # of requests such that $r(i) = p$; `// Assume` $m(p_1) \geq \ldots \geq m(p_n)$
$P \leftarrow \{p_1, \ldots, p_{c(i)}\}$;
for $j = 1$ ***to*** $c(i)$ **do**
  Move s_i to an arbitrary point $p \in P$;
  $P \leftarrow P - p$;
  Run ALG_{i-1} until cost incurred equals w_i ; `//` $(j+1)$`th subphase`
Terminate Phase

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).

4.4.2 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 4.11. *For $i \geq 2$, a complete phase of ALG_i consists of $(c(i)+1)$ subphases.*

Observation 4.12. *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 4.13. *For $i \geq 1$, the cost incurred by ALG_i to serve requests of a complete 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} incurs a cost of w_i. □

Using this, we get the following two simple properties.

Lemma 4.14. *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 4.13, 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 4.11 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 4.15. *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 show 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, during time intervals when ALG_i is called by ALG_k, ADV_i is an accurate estimate of ADV_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_k each time ALG_i is called, we require that for every complete phase, $\mathrm{cost}(\mathrm{ALG}_i) \leq R_i \cdot \mathrm{cost}(\mathrm{ADV}_i)$, for any initial configuration of ADV_i.

(ii) **The adversary can ignore a fraction of requests.** During time intervals when ALG_i is called by ALG_k, 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 for ALG_k. To get around this, we will require that $\mathrm{cost}(\mathrm{ALG}_i) \leq R_i \cdot \mathrm{cost}(\mathrm{ADV}_i)$, 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 $\mathrm{cost}(\mathrm{ALG}_i) \leq R_i \cdot \mathrm{cost}(\mathrm{ADV}_i)$. The key result is the following.

Theorem 4.16. *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 4.3. Indeed, for any request sequence σ, all phases except possibly the last one, are complete, so $\mathrm{cost}(\mathrm{ALG}_k) \leq R_k \cdot \mathrm{cost}(\mathrm{ADV}_k)$. 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, $\mathrm{ALG}_k(\sigma) \leq R_k \cdot \mathrm{ADV}_k(\sigma) + 2(c(k)+1)w_k$, and ALG_k is R_k-competitive. Together with loss in rounding the weights, this gives a competitive ratio of at mot $(R_k)^2 \leq 2^{2^{k+3}}$ for arbitrary weights.

We now prove Theorem 4.16.

Proof of Theorem 4.16. We prove the theorem by induction on k.

Base case ($i = 1$): As $R_1 > 6$ and $4/c(2) = 1/8 \leq 1/3$, 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 4.13, we have $\mathrm{cost}(\mathrm{ALG}_1) = 2(c(1)+1) = 6$. We show that $\mathrm{cost}(\mathrm{ADV}_1) \geq 1$. 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 $r_{t-1} \neq r_t$. 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 $\mathrm{cost}(\mathrm{ADV}_i) \geq$

$w_i/(2R_{i-1})$. This implies the theorem, as $\text{cost}(\text{ALG}_i) = 2(c(i)+1)w_i$ by Corollary 4.13 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 4.15, 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 4.14), 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 (4.1). So overall during subphase j, ADV_i uses servers $s_1, \ldots, s_{i-1}$ 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 4.13), 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. □

4.5 Lower Bounds

We present simple lower bounds on the competitive ratio of deterministic and randomized algorithms for the generalized k-server problem in uniform metrics.

Deterministic Algorithms. We show a simple construction due to [55] that directly implies a $(2^k - 1)/k$ lower bound on the competitive ratio of deterministic algorithm. Using a more careful argument, [55] 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.

Randomized Algorithms. Viewing the generalized k-server problem as a metrical service system (MSS), we can get a non-trivial lower bound for randomized algorithms. In particular, we can apply the $\Omega(\frac{\log N}{\log^2 \log N})$ lower bound due to Bartal et al. [16] on the competitive ratio of any randomized online algorithm against oblivious adversaries, for any metrical task system on N states. Of course, the MSS corresponding to a generalized k-server instance is restricted as the cost vectors may not be completely arbitrary. However, we consider the case where all metrics M_i have $n = 2$ points. Let s be an arbitrary state among the $N = 2^k$ possible states. A request in the antipodal point $\overline{s}$ only penalizes s and has cost 0 for every other state. So the space of cost vectors here is rich enough to simulate any MSS on these N states (note that if there is a general MSS request that has infinite cost on some subset S of states, then decomposing this into $|S|$ sequential requests where each of them penalizes exactly one state of S, can only make the competitive ratio worse).

This implies an $\Omega(\frac{k}{\log^2 k})$ lower bound for generalized k-server problem on uniform metrics.

4.6 Conclusion and Open Problems

In this chapter 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 4.1. *Is there an $f(k)$-competitive algorithm for the generalized k-server problem in general metric spaces?*

This problem is interesting in the context of both deterministic and randomized algorithms. In fact, answering this question requires the development of powerful new techniques for online algorithms and could lead to a much deeper theory of online computation.

Note that, even for the special case of the weighted k-server problem, no competitive algorithms are known for $k \geq 3$ beyond uniform metrics. Below,

we discuss some possible directions towards solving Open Problem 4.1.

Deterministic Algorithms. The only known candidate approach is to use the work functions. Thus, showing competitiveness seems to require understanding the structural and geometric properties of work functions. For example, in [53] competitiveness of WFA for the k-server problem is shown using a so-called quasi-convexity property. For more general problems similar properties are not known (all known results are for $k = 2$ [33, 65, 63], where a careful case analysis is applied). Such structural properties could be used either to show that the generalized WFA is competitive, or to enable the design of a new algorithm that uses the work functions. This could be for example a "dynamic" WFA, which uses the (generalized) WFA by adjusting parameters online, or any algorithm combined with WFA in an appropriate way (see e.g., [65, 24]).

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). In fact, the line metric could be an essential step towards generalizing to arbitrary metrics. It seems plausible that understanding the weighted k-server case could enable the solution of the generalized k-server problem. This was the case for $k = 2$ [63] and for weighted uniform metrics, where in Section 4.4 we used a natural modification of the weighted k-server algorithm to obtain a competitive algorithm.

Randomized Algorithms. The power of randomization in online algorithms is much less understood, and there is not even of a candidate randomized algorithm for the generalized k-server problem.

A natural first step is to understand the weighted uniform metric case. It seems plausible that an exponential improvement compared to deterministic algorithms is possible, i.e. it is natural to expect that there exists a $2^{\text{poly}(k)}$-competitive randomized algorithm against oblivious adversaries. However, even for the special case of the weighted k-server problem on uniform metrics, no better upper bound than $2^{2^{k+O(1)}}$ is known, and the best known lower bound is only $\Omega(\log k)$ and comes from the standard k-server in uniform metrics! This motivates the following question.

Open Problem 4.2. *What is the competitive ratio of randomized algorithms for weighted uniform metrics?*

Solving this problem seems to require non-trivial extensions of currently known techniques for providing both upper and lower bounds on the competitive ratio of randomized online algorithms.

Bibliography

[1] Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theor. Comput. Sci.*, 234(1-2):203–218, 2000.

[2] Susanne Albers. Improved randomized on-line algorithms for the list update problem. *SIAM J. Comput.*, 27(3):682–693, 1998.

[3] Susanne Albers and Jeffery Westbrook. Self-organizing data structures. In *Online Algorithms, The State of the Art*, pages 13–51, 1996.

[4] John Augustine and Nick Gravin. On the continuous CNN problem. In *ISAAC*, pages 254–265, 2010.

[5] Yossi Azar, Andrei Z. Broder, and Mark S. Manasse. On-line choice of on-line algorithms. In *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms (SODA)*, pages 432–440, 1993.

[6] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. *J. ACM*, 62(5):40, 2015.

[7] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. Towards the randomized k-server conjecture: A primal-dual approach. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 40–55, 2010.

[8] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. *J. ACM*, 59(4):19:1–19:24, 2012.

[9] Nikhil Bansal, Marek Eliáš, Łukasz Jeż, and Grigorios Koumoutsos. The (h, k)-server problem on bounded depth trees. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1022–1037, 2017.

[10] Nikhil Bansal, Marek Eliáš, Łukasz Jeż, Grigorios Koumoutsos, and Kirk Pruhs. Tight bounds for double coverage against weak adversaries. In *Approximation and Online Algorithms - 13th International Workshop (WAOA)*, pages 47–58, 2015.

[11] Nikhil Bansal, Marek Eliáš, Łukasz Jeż, Grigorios Koumoutsos, and Kirk Pruhs. Tight bounds for double coverage against weak adversaries. *Theory Comput. Syst.*, 62(2):349–365, 2018.

[12] Nikhil Bansal, Marek Eliáš, and Grigorios Koumoutsos. Weighted k-server bounds via combinatorial dichotomies. In *58th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 493–504, 2017.

[13] Nikhil Bansal, Marek Eliáš, Grigorios Koumoutsos, and Jesper Nederlof. Competitive algorithms for generalized k-server in uniform metrics. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 992–1001, 2018.

[14] Nikhil Bansal and Kirk Pruhs. Server scheduling to balance priorities, fairness, and average quality of service. *SIAM J. Comput.*, 39(7):3311–3335, 2010.

[15] Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *37th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 184–193, 1996.

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

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

[18] Shai Ben-David, Allan Borodin, Richard M. Karp, Gábor Tardos, and Avi Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994.

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

[20] Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992.

[21] Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. k-server via multiscale entropic regularization. In

Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC), pages 3–16, 2018.

[22] Niv Buchbinder and Joseph Naor. The design of competitive online algorithms via a primal-dual approach. *Foundations and Trends in Theoretical Computer Science*, 3(2-3):93–263, 2009.

[23] Niv Buchbinder and Joseph Naor. Online primal-dual algorithms for covering and packing. *Math. Oper. Res.*, 34(2):270–286, 2009.

[24] William R. Burley. Traversing layered graphs using the work function algorithm. *J. Algorithms*, 20(3):479–511, 1996.

[25] Jivitej S. Chadha, Naveen Garg, Amit Kumar, and V. N. Muralidhara. A competitive algorithm for minimizing weighted flow time on unrelatedmachines with speed augmentation. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing (STOC)*, pages 679–684, 2009.

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

[27] Ashish Chiplunkar and Sundar Vishwanathan. On randomized memoryless algorithms for the weighted k-server problem. In *FOCS*, pages 11–19, 2013.

[28] Marek Chrobak. SIGACT news online algorithms column 1. *SIGACT News*, 34(4):68–77, 2003.

[29] Marek Chrobak, Howard J. Karloff, Thomas H. Payne, and Sundar Vishwanathan. New results on server problems. *SIAM J. Discrete Math.*, 4(2):172–181, 1991.

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

[31] Marek Chrobak and Lawrence L. Larmore. The server problem and on-line games. In *On-line Algorithms, volume 7 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 11–64. AMS/ACM, 1992.

[32] Marek Chrobak and Lawrence L Larmore. *Metrical Service System: Deterministic Strategies.* Technical Report UCR-CS-93-1, Department of Computer Science, College of Engineering, University of California, Riverside, 1993.

[33] Marek Chrobak and Jiří Sgall. The weighted 2-server problem. *Theor. Comput. Sci.*, 324(2-3):289–312, 2004.

[34] Christian Coester, Elias Koutsoupias, and Philip Lazos. The infinite server problem. In *44th International Colloquium on Automata, Languages, and Programming, ICALP*, pages 14:1–14:14, 2017.

[35] Don Coppersmith, Peter Doyle, Prabhakar Raghavan, and Marc Snir. Random walks on weighted graphs and applications to on-line algorithms. *J. ACM*, 40(3):421–453, 1993.

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

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

[38] Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. On the additive constant of the k-server work function algorithm. In *Approximation and Online Algorithms, 7th International Workshop (WAOA)*, pages 128–134, 2009.

[39] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991.

[40] Amos Fiat and Manor Mendel. Better algorithms for unfair metrical task systems and applications. *SIAM J. Comput.*, 32(6):1403–1422, 2003.

[41] Amos Fiat, Yuval Rabani, and Yiftach Ravid. Competitive k-server algorithms. *J. Comput. Syst. Sci.*, 48(3):410–428, 1994.

[42] Amos Fiat, Yuval Rabani, Yiftach Ravid, and Baruch Schieber. A deterministic $o(k^3)$-competitive k-server algorithm for the circle. *Algorithmica*, 11(6):572–578, 1994.

[43] Amos Fiat and Moty Ricklin. Competitive algorithms for the weighted server problem. *Theor. Comput. Sci.*, 130(1):85–99, 1994.

[44] Edward F. Grove. The harmonic online k-server algorithm is competitive. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 260–266, 1991.

[45] L. Guth. *Polynomial Methods in Combinatorics*. University Lecture Series. American Mathematical Society, 2016.

[46] John Iacono. In pursuit of the dynamic optimality conjecture. In *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, pages 236–250, 2013.

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

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

[49] Stasys Jukna. *Extremal Combinatorics - With Applications in Computer Science*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2011.

[50] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, July 2000.

[51] Elias Koutsoupias. Weak adversaries for the k-server problem. In *Proc. of the 40th Symp. on Foundations of Computer Science (FOCS)*, pages 444–449, 1999.

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

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

[54] Elias Koutsoupias and Christos H. Papadimitriou. The 2-evader problem. *Inf. Process. Lett.*, 57(5):249–252, 1996.

[55] Elias Koutsoupias and David Scot Taylor. The CNN problem and other k-server variants. *Theor. Comput. Sci.*, 324(2-3):347–359, 2004.

[56] James R. Lee. Fusible HSTs and the randomized k-server conjecture. *CoRR*, abs/1711.01789, 2017.

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

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

[59] Lyle A. McGeoch and Daniel Dominic Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991.

[60] Lee Newberg. The k-server problem with distinguishable servers. *Master's Thesis, Univ. of California at Berkeley*, 1991.

[61] Kirk Pruhs. Competitive online scheduling for server systems. *SIGMETRICS Performance Evaluation Review*, 34(4):52–58, 2007.

[62] Nick Reingold, Jeffery Westbrook, and Daniel Dominic Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11(1):15–32, 1994.

[63] René Sitters. The generalized work function algorithm is competitive for the generalized 2-server problem. *SIAM J. Comput.*, 43(1):96–125, 2014.

[64] René Sitters, Leen Stougie, and Willem de Paepe. A competitive algorithm for the general 2-server problem. In *ICALP*, pages 624–636, 2003.

[65] René A. Sitters and Leen Stougie. The generalized two-server problem. *J. ACM*, 53(3):437–458, 2006.

[66] Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

[67] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985.

[68] Neal E. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, 1994.

[69] Neal E. Young. On-line file caching. *Algorithmica*, 33(3):371–383, 2002.

Summary

Algorithms for k-Server Problems

The topic of this thesis lies in the research area of optimization under uncertainty. Uncertainty arises in many practical areas like resource allocation, network design/maintenance and online advertising. In the presence of uncertainty, we wish to take decisions whose outcome is as close as possible to the optimal one we could achieve with full information. This is formalized in the framework of competitive analysis of online algorithms.

An *online algorithm* receives the input over time and when an action is needed, makes decisions without the knowledge of the future. The *competitive ratio* of such an algorithm is the maximum ratio, over all input sequences, of its cost to the optimal cost.

One of the most fundamental problems considered in competitive analysis is the k-server problem. It is a generalization of various online problems and its study has led to the development of generic techniques for online algorithms. Here, there are k servers located at points of a given metric space. At each time step, a request arrives at some point and must be served by moving a server there. The goal is to minimize the total distance traveled by the servers.

This thesis contributes to the theory of online algorithms by obtaining new results and designing new algorithms for several variants of the k-server problem which were poorly understood, as discussed next.

The (h, k)-server problem. This is the resource augmentation version of the k-server problem i.e., when the performance of the online algorithm with k servers is compared to the offline optimal solution with $h \leq k$ servers. Classic k-server results imply that for $k = h$ the competitive ratio is $\Theta(h)$, and the goal is to improve this guarantee as the ratio k/h increases. This was achieved for trees of depth 1 (this special case corresponds to the weighted caching problem), where many standard algorithms are roughly $(1 + 1/\epsilon)$-competitive when $k = (1 + \epsilon)h$, for any $\epsilon > 0$. Surprisingly however, no $o(h)$-competitive algorithm was known for any other metric space, even when (k/h) is arbitrarily large. We obtain several new results for the problem.

In Chapter 2 we consider the Double Coverage (DC) algorithm, an elegant algorithm for tree metrics, which is known to be h-competitive for $k = h$. We show that, surprisingly, when $k > h$ the competitive ratio of DC does not improve; in particular, it is exactly $\frac{k(h+1)}{k+1}$. Note that this ratio equals h for $k = h$ and increases up to $h + 1$ as k grows. In other words, the competitive ratio of DC becomes strictly worse as the number of servers increases.

In Chapter 3 we focus on trees of bounded depth. Our motivation arises from the fact that for trees of depth 1 we have a very good understanding of the problem, but for arbitrary trees we don't even know a good candidate algorithm. Thus it is natural to study trees of small depth and try to get a more systematic understanding of the problem. First, we consider the Double Coverage (DC) algorithm and the Work Function Algorithm (WFA), which are known to be $O(h)$-competitive for all trees for $h = k$. We show that they both fail to improve their performance as k increases, and they have competitive ratio $\Omega(h)$, even for trees of small depth. Then, by understanding the drawbacks of those algorithms, we present a new algorithm that is $O(1)$-competitive for constant depth trees, whenever $k = (1+\epsilon)h$ for any $\epsilon > 0$. This is the first $o(h)$-competitive algorithm for any metric space other than trees of depth 1. Finally, we show that the competitive ratio of any deterministic online algorithm is at least 2.4 even for trees of depth 2 and when k/h is arbitrarily large. This gives a surprising qualitative separation between trees of depth 1 and trees of depth 2 for the (h, k)-server problem.

The generalized k-server problem. This is a far-reaching extension of the k-server problem, where each server s_i lies in its own metric space M_i. A request is a k-tuple $r = (r_1, r_2, \ldots, r_k)$ and to serve it, we need to move some server s_i to the point $r_i \in M_i$. This problem can model a rich class of online problems, for which the techniques developed for the standard k-server problem do not apply, and it is widely believed that a deeper understanding of this problem should lead to powerful techniques for designing online algorithms. Despite much work, competitive algorithms were known only for $k = 2$.

In Chapter 4, we consider the problem on uniform metrics and present the first $f(k)$-competitive algorithms for any value of k. In particular, we obtain a deterministic algorithm for uniform metrics with competitive ratio $k \cdot 2^k$. This ratio is almost optimal, since it is long-known that the competitive ratio of any deterministic algorithm is at least $2^k - 1$. Furthermore, we design a randomized algorithm with competitive ratio $O(k^3 \cdot \log k)$. Finally, we consider the case where all metrics are uniform, but each one has different weight and we give a deterministic algorithm with competitive ratio $2^{2^{k+3}}$, which essentially matches the lower bound of $2^{2^{k-4}}$ for this case.

Curriculum Vitae

Grigorios Koumoutsos was born on March 14, 1989 in Amarousion, Greece. He studied Informatics and Telecommunications at the University of Athens, where he graduated in July 2012. He continued his studies in Paris, France, where he pursued the Parisian Master of Research in Computer Science (MPRI), a master programme organized jointly by University Paris Diderot, École normale supérieure (ENS) Paris, ENS Cachan, École Polytechnique and Telecom Paris Tech. He obtained his Master's degree in 2014 within the Algorithms and Complexity group of University Paris Diderot under the supervision of Adi Rosén.

Since September 2014 he has been a PhD student in the Combinatorial Optimization group of Eindhoven University of Technology under the supervision of Nikhil Bansal.