Competitive Analysis of k-Server Variants and Metrical Task Systems

Christian Coester

Wolfson College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2019

Acknowledgements

I would like to thank Elias Koutsoupias for guiding and supporting me throughout the last four years, being available when I needed his help, sharing his expertise, and giving me the confidence to pursue an academic career. I am grateful to James Lee for generously hosting me during my visits in Seattle, his advice and support during and after this time, and teaching me how to think more effectively as a researcher. Further, I would like to thank Seffi Naor for inviting me to Haifa and, among other things, significantly improving my understanding of the online primal-dual framework. I am thankful to Marek Chrobak and Philip Lazos for enjoyable collaborations, and Thomas Schwentick for the profound influence he has had on me especially before I came to Oxford. I would like to thank Anupam Gupta and Varun Kanade for examining this thesis and the feedback they have given me.

The financial support from EPSRC is highly appreciated.

I thank my friends in and outside Oxford for making this journey so enjoyable. Most of all, I thank my parents for supporting me in pursuing my passion and continuously laying the foundation for everything I do.

Notes on joint work

This thesis is based on joint works with Elias Koutsoupias and Philip Lazos [CKL17], with Elias Koutsoupias [CK19] and with James R. Lee [CL19]. The extent of my contribution was as follows.

For [CKL17], I found the proof of the equivalence theorem between ∞-server and (h, k)-server (Section 2.2). I also contributed to finding the remaining results (either finding them myself or jointly through discussion with my coauthors), except for the uncompetitiveness of the aggressively spawning Double Coverage variant (Lemma 2.4.6) and the reduction to bounded spaces (Section 2.5). The writing was shared.

In [CK19], I found all results, in part following significant inspiration from my coauthor. Most of the writing was carried out by me.

In [CL19], my coauthor suggested the regularizer and I found a first version of a proof for HSTs where each internal vertex has the same number of children, which we extended together to arbitrary HSTs. Writing was shared, but my coauthor contributed more to writing than me.

Abstract

In the online k-server problem, an algorithm controls k mobile servers in a metric space. One by one, requests arrive at points of the space, and the algorithm must serve each request by selecting a server to visit it. The goal is to minimize the total distance traveled by all servers. In the framework of competitive analysis, we study two variants of the k-server problem, the ∞-server problem and the k-taxi problem, and the more general metrical task systems problem.

The *∞-server problem* is the variant of the k-server problem where the number of servers is infinite, initially all starting at the same point. We obtain a surprisingly tight connection between the ∞-server problem and the resource augmentation version of the k-server problem. Using this connection, we also improve the known lower bounds for the resource augmented k-server problem.

The *k-taxi problem* generalizes the k-server problem in that a request consists not of one point, but two points s and t, representing the start and destination of a taxi request. To serve such a request, a server (taxi) must move first to s and then to t. This problem becomes particularly difficult when the cost is defined as the distance of empty runs only. Indeed, we show an exponential gap between the competitive ratio of the k-taxi problem and that of the k-server problem. A main positive result is an $O(2^k \log n)$-competitive algorithm for arbitrary n-point metrics.

Metrical task systems are a general framework subsuming many other online problems, including the k-server problem. Here, the algorithm suffers two kinds of costs, movement and service costs. For HST metrics, using an entropy regularization approach, we obtain tight bounds on the refined guarantees, i.e., movement and service costs are simultaneously optimally competitive against the optimal total cost. This also improves the refined guarantees for general metrics.

Contents

1 Introduction

1.1 Online optimization

The field of mathematical optimization is concerned with finding a best solution from a set of feasible solutions. A classical example of an optimization problem is scheduling: Given a fixed number of identical machines and a set of jobs with processing times $p_1, \ldots, p_n$, what is the best assignment of jobs to machines so as to minimize the total processing time? The scheduling problem is well known to be NP-complete. In real-life applications, besides computational complexity, one often faces another difficulty: The jobs are not known in advance. If the jobs arrive one by one and each job needs to be assigned to a machine upon arrival, even unbounded computational power would not allow finding an optimal solution in general. The study of optimization problems with incomplete information is known as *online optimization*.

An *online problem* is an optimization problem where the input is revealed piece-by-piece over time. Typically, the part of the input that is revealed over time can be modelled as a sequence of *requests*. An *online algorithm* must make an irrevocable decision as each request is revealed ("serve the request"), incurring some cost or gaining value. The goal is to either minimize the total cost (for minimization problems) or maximize the total gain (for maximization problems).

In the scheduling example, requests correspond to jobs, serving a request entails assigning it to a machine, and the total cost is the makespan, i.e., the maximal sum of processing times of all jobs assigned to any one particular machine.

The performance of an online algorithm can be evaluated by comparing it to the best *offline algorithm*, i.e., to an algorithm that knows the entire request sequence in advance. This method of analyzing online algorithms is called competitive analysis.

1.1.1 Competitive analysis

The classical measure of quality of an online algorithm is its competitive ratio. This is the worst-case ratio between the quality of the solution computed by the online algorithm and the optimal solution computed by an offline algorithm. More precisely, an algorithm A for an online minimization problem is called ρ-*competitive* if

$$cost_A(\sigma) \leq \rho \cdot opt(\sigma) + c, \tag{1.1}$$

for each request sequence σ, where $cost_A(\sigma)$ denotes the cost of A and $opt(\sigma)$ the optimal cost for request sequence σ, and c is some constant independent of σ. If the additive term c is 0, then the algorithm is also called *strictly* ρ-*competitive*. It is *competitive* if it is ρ-competitive for some ρ. The *competitive ratio* of an online algorithm is the infimum of all ρ such that it is ρ-competitive. We also use these terms for problems rather than algorithms: An online problem is ρ-competitive if there exists a ρ-competitive algorithm, and its competitive ratio is the infimum of all ρ such that it is ρ-competitive.

The competitive ratio is the online analog of the approximation ratio in the field of approximation algorithms. Since online optimization aims to determine the information theoretic limitations rather than computational difficulty, online algorithms are not necessarily required to run in polynomial time. In principle, online algorithms are not even required to be computable, but they can be any mathematical function that maps visible prefixes of request sequences to legal ways to serve the most recent request.

A common way to think of an online problem is as a game between the online algorithm and a malicious adversary. The adversary creates a sequence of requests

on which the algorithm performs poorly. The adversary also serves these requests, trying to maximize the ratio between the algorithm's cost and its own cost. The competitive ratio of a problem is the ratio between the algorithm's cost and the adversary's cost if both play optimally in this game. The concept of an adversary is particularly relevant in the context of randomized online algorithms.

1.1.2 Randomization in online optimization

For many online problems, randomized algorithms can achieve a better competitive ratio than deterministic ones. Intuitively, this is because randomization makes the online algorithm unpredictable, thus making it harder for an adversary to tailor a particularly detrimental worst-case sequence to the specific algorithm.

The precise definition of competitive ratio in the randomized setting depends on the adversary model. The most common adversary model is the *oblivious adversary*: This adversary knows the algorithm, but it has to create the input sequence without knowledge of the outcomes of random choices by the algorithm. The competitive ratio is defined as in the deterministic case except that in (1.1), $cost_A(\sigma)$ is replaced by its expectation.

A stronger type of adversary is the *adaptive online adversary*. Unlike the oblivious adversary, it does not have to construct the input sequence in advance. Instead, it needs to define each piece of the input only by the time that piece is revealed to the algorithm. In particular, it can make this decision based on past random selections by the algorithm. However, the adversary needs to serve the requests immediately as well. Thus, the adversary's cost is not necessarily the optimal cost for the input sequence in hindsight (unless it decides to choose the input sequence in advance, so that it would know how to serve it optimally, like an oblivious adversary). Now, (1.1) becomes

$$\mathbb{E}(cost_A(\sigma)) \leq \rho \cdot \mathbb{E}(cost_{ADV}(\sigma)) + c,$$

where $cost_{ADV}$ denotes the cost of the adversary, with the randomness on the right hand side coming from the fact that σ is now a random variable.

The strongest adversary is the *adaptive offline adversary*. It is defined like the adaptive online adversary except that it serves the request sequence optimally in the end.

Ben-David, Borodin, Karp, Tardos and Wigderson [BBK+94] showed important results about the relationship between the adversary models: If there exists a β-competitive algorithm against oblivious adversaries, then any α-competitive algorithm against adaptive online adversaries is also $\alpha\beta$-competitive against adaptive offline adversaries. Consequently, any α-competitive algorithm against adaptive online adversaries is α^2-competitive against adaptive offline adversaries. Moreover, they showed that if there exists an α-competitive algorithm against adaptive offline adversaries, then there also exists an α-competitive deterministic algorithm. Thus, against adaptive online adversaries, randomization can reduce the competitive ratio at best to the square-root of the deterministic competitive ratio.

It shall be noted that for deterministic algorithms, all three adversary models are equivalent. For randomized algorithms, unless stated otherwise we assume an oblivious adversaries.

1.1.3 Early literature

The terminology of competitive analysis was coined by Karlin, Manasse, Rudolph and Sleator in 1988 [KMRS88]. However, the concept already came up in the literature before this. Perhaps the first example is the aforementioned scheduling problem: In 1966, Graham [Gra66] showed that the greedy online strategy of assigning each arriving job to the least loaded machine is 2-competitive. In a seminal paper almost 20 years later, Sleator and Tarjan presented competitive ananlysis for the list update and paging problems [ST85a], which they called "amortized efficiency" at the time. Around the same time, they also introduced Splay trees [ST85b], an elegant type of self-adjusting binary search trees. When the cost is taken to be the total access time for elements, Splay trees achieve competitive ratio at most $O(\log n)$, where n is the number of different elements. The dynamic optimality conjecture, one of the most important conjectures in the fields of online

algorithms and data structures, states that Splay trees are in fact $O(1)$-competitive. Following the two papers by Sleator and Tarjan, the field of competitive analysis truly emerged. A few year later, the k-server problem was introduced [MMS88] and would become a driving force of the field.

1.2 k-server and related problems

The k-server problem is often referred to as the "holy grail" of competitive analysis. In this section, we describe this problem as well as several related problems and review important results.

For a broader exposition of competitive analysis and a review of several main results we refer to the classical books of Borodin and El-Yaniv [BE98] and Fiat and Woeginger [FW98]. A modern approach to competitive analysis, the primal-dual framework for online algorithms, is presented in a survey by Buchbinder and Naor [BN09].

1.2.1 The k-server problem

Probably the most influential problem in the area of online computation is the k-server problem, introduced by Manasse, McGeoch and Sleator [MMS88]. It is defined as follows: An online algorithm controls k mobile servers in a metric space; one by one, requests arrive at points of the metric space, and the algorithm must each request by moving a server to the requested point; the cost is the cumulative distance traveled by all servers.

If the underlying metric space is the uniform metric (i.e., the distance between any two points is 1), then the k-server problem is equivalent to the paging problem. If the metric space is a weighted star (i.e., the set of leaves of a tree of depth 1), it is equivalent to weighted paging.

Despite its simple description, the k-server problem turns out to be highly non-trivial. Several techniques developed for the k-server problem have also proved useful for other online problems. We review part of the rich history of this problem

in this subsection. A more in-depth survey of some of these and additional results can be found in [Kou09].

Deterministic algorithms

It is known that the deterministic competitive ratio of the k-server problem is at least k on any metric space with more than k points [MMS88]. The famous k-server conjecture states that this bound is tight. This has been shown to be true for $k = 2$ [MMS88], for uniform metrics [ST85a], the line (i.e., the 1-dimensional Euclidean space) [CKPV91], tree metrics [CL91] and metric spaces of $k + 1$ [MMS88] or $k + 2$ points [KP96].

For general metric spaces, the first algorithm achieving a competitive ratio depending only on k was found by Fiat et al. [FRR90]. An improved, albeit still exponential, upper bound on the competitive ratio was given by Grove [Gro91]. A breakthrough was achieved when Koutsoupias and Papadimitriou showed that the work function algorithm achieves a competitive ratio of at most $2k - 1$ [KP95], reducing the gap in the k-server conjecture to a factor of 2. To date, this is the best known upper bound for deterministic algorithms.

The Double Coverage algorithm. An elegant algorithm achieving the optimal competitive ratio of k on the line is called *Double Coverage* [CKPV91]. Observe that on the line, every request that is placed in the convex hull of the current server positions has a left and a right neighboring server, and clearly the only sensible move is to use one of these two servers to serve the request. But how should one decide which of the two servers to use? The Double Coverage algorithm simply moves both servers towards the request by the same distance, and this distance is the distance between the request and the closer of the two neighboring servers. Thus, the closest server reaches the request, but the position of the other neighboring server is also updated, which may impact the way that future requests will be served. Another view on this algorithm is that both neighboring servers move at equal speed towards the request, and as soon as one of them hits the request, both servers stop moving. If a request is placed outside the convex hull

of server positions, it has only one neighboring server and this server moves to the request. Using a potential function analysis, Chrobak et al. [CKPV91] showed that Double Coverage is k-competitive on the line.

In subsequent work, Chrobak and Larmore presented a natural generalization of Double Coverage to tree metrics [CL91]. Again, the algorithm moves all adjacent servers towards the request at equal speed until one of them reaches the request, but on trees there can be more than two adjacent servers. A server is said to be adjacent to the request if no other server lies on the path between this server and the request. If several servers are located at the same point and no other server lies on their path to the request, only one of them (chosen arbitrarily) is considered to be adjacent. One should notice that unlike on the line metric, the set of adjacent servers can change during the movement of the servers for an individual request. A proof very similar to the case of the line metric yields k-competitiveness also for trees.

The work function algorithm The upper bound of $2k - 1$ for general metrics is achieved by the *work function algorithm*. This algorithm is defined as follows: A *configuration* C is a set of k points in the metric space, corresponding to the positions of the k servers. For a fixed initial configuration, a sequence of requests $r_1, r_2, r_3, \ldots$, and a configuration C, the *work function* value $w_t(C)$ of C at time t is the minimal cost to serve the first t requests and end up in configuration C. If C_{t-1} is the server configuration before the tth request, the algorithm moves to a configuration C_t that contains r_t and minimizes the quantity

$$w_t(C_t) + d(C_{t-1}, C_t),$$

where $d(C_{t-1}, C_t)$ is the cost of moving from C_{t-1} to C_t.

Interestingly, neither minimizing only $d(C_{t-1}, C_t)$ (the greedy algorithm, which serves each request by the closest server) nor minimizing only $w_t(C_t)$ (the retrospective algorithm, always going to the configuration that the optimal algorithm would be in if the request sequence ended here) is competitive; nonetheless, balancing the objectives of these two algorithms – like the work function algorithm does – achieves the best known competitive ratio for the problem.

It is conjectured that the work function algorithm is in fact k-competitive, which would prove the k-server conjecture. This conjecture is supported by the fact that it achieves the optimal competitive ratio of k for several special cases, namely $k = 2$ [CL92], metric spaces of at most $k+2$ points [KP96], the line, weighted stars [BK04] and the case $k = 3$ in the Manhattan plane [BCL02].

Randomized algorithms

The deterministic lower bound of k can be overcome using randomization. This was first shown for uniform metrics, where Fiat et al. gave an algorithm with competitive ratio $O(\log k)$ [FKL+91]. The folklore randomized k-server conjecture states that a competitive ratio of $O(\log k)$ is achievable on every metric space, and there has been substantial progress on this question recently. Besides uniform metrics, this randomized k-server conjecture was also confirmed for weighted star metrics [BBN12] using the primal-dual framework for online-algorithms. The first algorithm with polylogarithmic competitive ratio on any finite metric space was given by Bansal et al. [BBMN15], who presented an $O(\log^2 k \log^3 n \log\log n)$-competitive algorithm, where n is the number of points of the metric space. This result is based on an embedding theorem of arbitrary metric spaces in hierarchical well-separated tree metrics (HSTs) [Bar96, FRT04]. Thanks to this embedding, any ρ-competitive algorithm for HSTs yields an $O(\rho \log n)$-competitive algorithm for general metrics. More recently, Bubeck et al. [BCL+18] gave an $O(\log^2 k)$-competitive algorithm for HSTs, which implies an $O(\log^2 k \log n)$-competitive algorithm for general n-point metrics. To state their algorithm, they introduce a continuous-time mirror descent framework for online algorithms (see also [BGMN19] for a discretized version of their algorithm). Mirror descent is a technique originating from the field of convex optimization, which recently saw applications in online decision making [ABBS10, BCN14, Haz16]. Building on this $O(\log^2 k)$-competitive algorithm for HSTs, Lee [Lee18a] introduced a technique of dynamically changing HST embeddings over time in order to obtain a polylog(k)-competitive algorithm for arbitrary metrics.[1]

[1] There is a gap in the version posted to the arXiv on February 21, 2018 [Lee18b, Lee19].

The best known lower bound on the competitive ratio of randomized algorithms which holds for every metric space with more than k points is $\Omega(\log k/\log\log k)$ [BBM06, BLMN05]. On some metric spaces, e.g. uniform metrics, a better lower bound of $\Omega(\log k)$ is known [FKL+91], matching the upper bound on these spaces and suggesting that $\Theta(\log k)$ may indeed be the right answer for general metrics.

1.2.2 Variants of the k-server problem

We discuss now some variants of the k-server problem that will be relevant in the later chapaters. Beyond the ones mentioned below, further variants of the k-server problem include the weighted k-server problem [BEK17, CV13, CS04, FR94] and the generalized k-server problem [KT04, SS06, BEKN18].

The (h,k)-server problem

The (h,k)-server problem differs from the k-server problem only in the definition of the competitive ratio: Instead of comparing the online algorithm to an offline algorithm with k servers, it is compared to an offline algorithm with h servers, for some $h \leq k$. This model is motivated by the observation that even though the (deterministic) competitive ratio of the k-server problem is linear in k, at least on some metric spaces it shrinks to a constant when the number of online and offline servers differ by any fixed multiplicative factor arbitrary close to 1. Therefore, all worst-case instances for these metrics are ones where the optimal value changes drastically in response to a small change of resources, which seems atypical and too pessimistic. The (h,k)-server problem, also known as the *weak adversaries* model of the k-server problem, is a way to obtain more realistic performance bounds.

This method of assuming different resources for the online and offline algorithms is known as *resource augmentation* and has led to spectacular success for online scheduling (see e.g. [KP00, PSTW02]).

For the k-server problem on uniform metric, Sleator and Tarjan already showed in their early paper on online paging that the competitive ratio drops to $\frac{k}{k-h+1}$ in the weak adversaries model [ST85a]. Later, this result was generalized to weighted star metrics [You94]. In particular, the online algorithm needs only $1+\epsilon$ times

as many servers as the offline algorithm to achieve a competitive ratio that can be bounded by a constant independent of the number of servers. Moreover, the competitive ratio on these spaces tends to 1 as $k/h \to \infty$. More recently, it was shown for metrics induced by the vertices of an edge-weighted tree (with the distance between two vertices being the length of their connecting path), the competitive ratio when $k = (1+\epsilon)h$ can also be bounded by a constant depending on ϵ and the combinatorial depth of the tree [BEJK19].

On general metrics, the (h,k)-server problem is much less understood: The best known upper bound on the competitive ratio on general metrics is $O(h)$, which is no improvement over the setting without resource augmentation. No algorithm is known for general metrics that performs better than disabling the $k-h$ extra servers and using h servers only. In fact, for instance on the line it was shown [BEJ+15, BEJK19] that Double Coverage and the work function algorithm perform slightly *worse* in the resource augmentation setting than disabling the $k-h$ extra servers and applying the same algorithm to h servers only. For the case that h is not fixed, the work function algorithm was shown to be $2h$-competitive simultaneously against any number $h \leq k$ of offline servers [Kou99].

With respect to lower bounds, it is known that unlike for uniform and weighted stars, the competitive ratio does not converge to 1 on general metrics when $k/h \to \infty$: Bar-Noy and Schieber (see [BE98, p. 175]) showed that the $(k,2)$-server problem on the line metric has competitive ratio exactly 2 for each $k \geq 2$. In other words, increasing the number of online servers yields no benefit in the case of 2 offline servers. For general metric spaces, in [BEJK19] a lower bound of 2.4 was shown to hold even as $k/h \to \infty$. In Chapter 2 we will obtain an improved lower bound of 3.146.

In terms of randomized algorithms, [You91] showed for the paging special case that if $k = (1+\epsilon)h$, the competitive ratio is $\Theta(\log(1/\epsilon))$, which was later established for weighted paging as well [BBN12]. More generally, for HST metrics of depth D, an upper bound of $O(D\log(1/\epsilon))$ was shown recently [BGMN19].

A major open problem is whether the (h,k)-server problem has bounded competitive ratio on general metric spaces when k is sufficiently larger than h.

The ∞-server problem

In joint work with Lazos and Koutsoupias [CKL17], we introduced the ∞-server problem, which was originally proposed to us by Kamal Jain. This is the variant of the k-server problem where $k = \infty$. Of course, this problem would be trivial if each point of the metric space was already occupied by a server in the initial configuration. To avoid this, we assume that all infinitely many servers initially reside at the same point, the *source*.

Paradoxically, even though the competitive ratio of the k-server problem goes to infinity as $k \to \infty$, once $k = \infty$ it goes back to a small constant at least on some metric spaces. For example, consider the uniform metric and observe that the ∞-server problem has competitive ratio 1 for this case. Thus, the high competitive ratio of the k-server problem can be misleading for situations where the number of servers is so high that it is not a limitation in practice, and a study of the ∞-server problem can provide a better understanding of these cases.

The ∞-server problem also allows to model applications where more servers can be bought. A price for buying new servers can be modeled easily by appropriate placement of the source in the metric space. In [CIN+01], Csirik et al. study a problem that is essentially the special case of the ∞-server problem on the uniform metric space augmented by a far away source. It is cast as a paging problem where new cache slots can be bought at a fixed price per unit and gives matching upper and lower bounds of ≈ 3.146 on the competitive ratio.

The ∞-server problem is also a considerable generalization of the ski-rental problem, since the ski-rental problem is essentially a special case of the ∞-server problem when the metric space is an isosceles triangle.

As we will see in Chapter 2, the ∞-server problem is tightly connected to the (h, k)-server problem, and we will use this connection to improve the lower bound on the (h, k)-server problem.

The k-taxi problem

The k-taxi problem was originally proposed by Karloff and introduced by Fiat et al. [FRR90] as a natural generalization of the k-server problem. In this problem, k taxis are located in a metric space and need to serve a sequence of requests. A request is a pair (s, t) of two points in the metric space, representing a passenger that wants to travel from s to t. A taxi serves the request by first moving to s and then to t.

The problem comes in two flavors, called the easy and the hard k-taxi problem. In the *easy k-taxi problem*, the cost is defined as the total distance traveled by the taxis. In the *hard k-taxi problem*, the cost is defined as only the distance of empty runs, i.e., the distance traveled while not carrying a passenger.

The hard version is motivated by the fact that for a request (s, t), the distance from s to t needs to be traveled anyway — independently of the algorithm's decisions. It therefore makes sense to exclude it from the cost and minimize only the overhead travel that actually depends on the algorithm choices. Thus, the cost of any taxi schedule differs by exactly the sum of the s-t-distances between the two versions, and in particular, the optimal offline solutions are the same for both versions. However, the different cost functions make it more difficult to approximate the optimal solution value of the hard version. Fiat et al. [FRR90] pointed out the two versions of this problem, and they were called easy taxicab problem and hard taxicab problem by Kosoresow [Kos96].

The problem was recently reintroduced as the Uber problem in [DEH+17], which studied the easy version of the problem and with the input being produced in a stochastic manner.

Besides scheduling taxis, the k-taxi problem also models other tasks such as scheduling elevators (if the metric space is the line) or transport vehicles in a factory, and other applications where people or objects need to be transported between locations.

Previous results. The first competitive algorithm for the easy k-taxi problem was given by Fiat et al. [FRR90] when they introduced the problem, with a competitive ratio exponential in k. Following the finding of the $(2k-1)$-competitive work function Algorithm for the k-server problem [KP95], the competitive ratio of the easy k-taxi problem was improved to $2k+1$: Kosoresow [Kos96] showed that if there is a c-competitive algorithm for the k-server problem, then there is a $(c+2)$-competitive algorithm for the easy k-taxi problem. This result was also established in [DEH+17] with a similar reduction.

For the hard k-taxi problem, Fiat et al. [FRR90] mentioned the existence of a competitive algorithm for $k = 2$ due to Karloff. Based on Karloff's algorithm, Kosoresow [Kos96] gave a 15-competitive algorithm for $k = 2$. No competitive algorithm is known for $k > 2$.

1.2.3 Set chasing problems and layered graph traversal

In set chasing problems, also known as *metrical service systems (MSS)* [CL96], there is a single server in a metric space, and a request is a *set* of points in the space. To service a request, the server has to move to any point in the requested set. This problem is a considerable generalization of the k-server problem: If we take the space of all k-server configurations as the metric space for MSS, then a request point r for k-server translates to the request set of all configurations containing r for MSS. For the existence of competitive algorithms, it is necessary to restrict the underlying metric space and/or the type of request sets that are allowed. (To see this, consider, for instance, the infinite uniform metric and suppose each request is the set of all points not visited by the online algorithm yet.) Prominent examples of competitive restrictions are chasing convex bodies and chasing sets of cardinality k, the latter of which is also equivalent to traversing layered graphs of width k.

Convex body chasing

The convex body chasing problem is the version of MSS where the underlying metric space is $\mathbb{R}^d$ with some norm and all request sets must be convex. The problem was introduced by Friedman and Linial [FL93], who gave a competitive algorithm for the

plane $\mathbb{R}^2$, a lower bound of $\sqrt{d}$ for $\mathbb{R}^d$ with the Euclidean norm, and conjectured that a finite competitive is achievable for any d. Twenty-five years later, this conjecture was finally confirmed by Bubeck et al. [BLLS19], who gave a $2^{O(d)}$-competitive algorithm. Very recently, Argue et al. [AGGT19] and Sellke [Sel19] independently found an elegant proof to further improve the upper bound to $O(d)$.

k-MSS and layered graph traversal

In the k-MSS problem [CL96], the metric space is arbitrary and all request sets have cardinallity at most k. In [FFK+98], this problem was shown to be equivalent to the problem of traversing layered graphs of width k. In this problem, first introduced in [PY91], the goal is to minimize the distance traveled by a searcher who seeks a target vertex in a layered graph. Each layer consists of k vertices, with edges only between adjacent layers, and a layer is not revealed until a vertex in the previous layer is visited. The best known bounds on the competitive ratio of these problems are $\Omega(2^k)$ [FFK+98] and $O(k2^k)$ [Bur96] in the deterministic case and $\Omega\left(\frac{k^2}{\log^{1+\epsilon}(k)}\right)$ and $O(k^{13})$ in the randomized case [Ram95].

1.2.4 Metrical task systems

Let (X, d_X) be a finite metric space with $|X| = n$. The *metrical task systems (MTS)* problem, introduced in [BLS92], is described as follows. The input is a sequence $\langle c_t : X \to \mathbb{R}_+ : t \geq 1\rangle$ of nonnegative cost functions on the state space X. At every time t, an online algorithm maintains a state $\rho_t \in X$. The corresponding cost is the sum of a *service cost* $c_t(\rho_t)$ and a *movement cost* $d_X(\rho_{t-1}, \rho_t)$.

MTS is a further generalization of MSS, and therefore also of the k-server problem: Indeed, a request set $S \subseteq X$ for MSS can be modeled by the cost function that takes value 0 on S and ∞ elsewhere.[2]

Deterministic algorithms for MTS are well-understood: On every n-point metric space, the optimal competitive ratio is exactly $2n - 1$ [BLS92].

[2] If cost functions are required to take finite values, one can also take the cost function that maps any point $x \in X$ to twice the distance from x to the closest point in S. By the triangle inequality, it makes no sense for an algorithm to service this cost function at a point outside of S.

Finding the optimal competitive ratio of randomized algorithms is still an open problem on general metric spaces. For the n-point uniform metric, a simple coupon-collector argument shows that the competitive ratio is $\Omega(\log n)$, and this is tight [BLS92]. The methods of [BBN12] show that the same competitive ratio holds on weighted star metrics. Recently, [BCLL19] generalized this further by designing an algorithm with competitive ratio $O(\log n)$ on HST metrics and $O(D \log n)$ on any weighted tree metric with combinatorial depth D. A long-standing conjecture is that a $\Theta(\log n)$ competitive ratio holds for an arbitrary n-point metric space. The lower bound has almost been established [BBM06, BLMN05]; for any n-point metric space, the competitive ratio is $\Omega(\log n / \log\log n)$. Following a long sequence of works (see, e.g., [Sei99, BKRS00, BBBT97, Bar96, FM03, FRT04]), an upper bound of $O(\log^2 n)$ was shown in [BCLL19], following from the aforementioned $O(\log n)$ bound on HSTs.

Refined Guarantees

The authors of [BBN10] observe that there is a more refined way to analyze competive algorithms for MTS. For a (randomized) online algorithm A and a cost sequence $\boldsymbol{c}$, we write $\mathsf{S}_A(\boldsymbol{c})$ and $\mathsf{M}_A(\boldsymbol{c})$, respectively, for the (expected) service cost and movement cost of A on cost sequence $\boldsymbol{c}$.

If there are numbers $\alpha, \alpha', \beta, \beta' > 0$ such that for every cost $\boldsymbol{c}$, it holds that

$$\mathsf{S}_A(\boldsymbol{c}) \leq \alpha \cdot \mathit{opt}(\boldsymbol{c}) + \beta$$
$$\mathsf{M}_A(\boldsymbol{c}) \leq \alpha' \cdot \mathit{opt}(\boldsymbol{c}) + \beta',$$

one says that A is *α-competitive for service costs* and *α'-competitive for movement costs.*

In [BBN10], it is shown that on every n-point HST metric, and for every $\epsilon > 0$, there is an online algorithm that is simultaneously $(1+\epsilon)$-competitive for service costs and $O((\log(n/\epsilon))^2)$-competitive for movement costs. The authors of [BCLL19] improve this slightly to show that actually there is an online algorithm for HSTs that is simultaneously 1-competitive for service costs and $O((\log n)^2)$-competitive for

movement costs. For general metrics, this yields an algorithm that is simultaneously 1-competitive for service costs and $O((\log n)^3)$-competitive for movement costs.

1.3 Contribution

We study the ∞-server problem, the k-taxi problem and metrical task systems.

Chapter 2 investigates the ∞-server problem and is based on [CKL17]. We show a surprisingly tight connection between this problem and the (h,k)-server problem. Specifically, we show that the ∞-server problem has bounded competitive ratio if and only if the (h,k)-server problem has bounded competitive ratio for some $k = O(h)$. We give a lower bound of 3.146 for the competitive ratio of the ∞-server problem, which holds in particular for the line and some simple weighted stars. It implies the same lower bound for the (h,k)-server problem on the line, even when $k/h \to \infty$, improving on the previous known bounds of 2 for the line and 2.4 for general metrics. For weighted trees and layered graphs we obtain upper bounds, although they depend on the depth. Of particular interest is the ∞-server problem on the line, which we show to be equivalent to the seemingly easier case in which all requests are in a fixed bounded interval away from the original position of the servers. This is a special case of a more general reduction from arbitrary metric spaces to bounded subspaces. Unfortunately, classical approaches (double coverage and generalizations, work function algorithm, balancing algorithms) fail even for this special case.

The k-taxi problem is examined in Chapter 3, which is based on [CK19]. We show that the hard k-taxi problem is substantially more difficult than the easy version with at least an exponential deterministic competitive ratio, $\Omega(2^k)$, admitting a reduction from the layered graph traversal problem. In contrast, the easy k-taxi problem has exactly the same competitive ratio as the k-server problem. We focus mainly on the hard version. For HST metrics, we present a memoryless randomized algorithm with competitive ratio $2^k - 1$ against adaptive online adversaries and provide two matching lower bounds: for arbitrary algorithms against adaptive adversaries and for memoryless algorithms against oblivious adversaries. Due to the known HST embedding techniques [Bar96, FRT04], the algorithm implies a

randomized $O(2^k \log n)$-competitive algorithm for arbitrary n-point metrics. This is the first competitive algorithm for the hard k-taxi problem for general finite metric spaces and general k. For the special case of $k = 2$, we obtain a precise answer of 9 for the competitive ratio in general metrics. With an algorithm based on growing, shrinking and shifting regions, we show that one can achieve a constant competitive ratio also for the hard 3-taxi problem on the line (abstracting the scheduling of three elevators).

In Chapter 4, we study metrical task systems (MTS), based on [CL19]. We show that on every n-point HST metric, there is a randomized online algorithm for MTS that is 1-competitive for service costs and $O(\log n)$-competitive for movement costs. In general, these refined guarantees are optimal up to the implicit constant. While an $O(\log n)$-competitive algorithm for MTS on HST metrics was developed in [BCLL19], that approach could only establish an $O((\log n)^2)$-competitive ratio when the service costs are required to be $O(1)$-competitive. Our algorithm can be viewed as an instantiation of online mirror descent with the regularizer derived from a multiscale conditional entropy. In fact, our algorithm satisfies a set of even more refined guarantees. Combined with the aforementioned HST embedding techniques this yields, for *any* n-point metric space, a randomized algorithm that is 1-competitive for service costs and $O((\log n)^2)$-competitive for movement costs.

2

The ∞-Server Problem

2.1 Introduction

Our main result in this chapter is an equivalence theorem between the ∞-server problem and the (h, k)-server problem, presented in Section 2.2. It states that the ∞-server problem is competitive on every metric space if and only if the (h, k)-server problem is $O(1)$-competitive on every metric space as $k/h \to \infty$. We show further that it is not even necessary to let k/h tend to infinity because in the positive case, there must also exist some $k = O(h)$. The theorem holds also if "every metric space" is replaced by "the line metric".

In Section 2.3 we present upper and lower bounds on the competitive ratio of the ∞-server problem on a variety of metric spaces. Extending the work of [CIN+01], we present a tight lower bound of approximately 3.146 for a class of metric spaces that includes every non-discrete space as well as some simple weighted stars. This lower bound is then turned into a lower bound of the same value for the (h, k)-server problem, improving on the previous lower bounds for this setting. We show how work by Bansal et al. [BEJK19] can be adapted to give an upper bound on the competitive ratio of the ∞-server problem on bounded-depth trees. We also consider layered graph metrics, which are equivalent (up to a factor of 2) to general graph metrics. We have not settled the case for their competitive

ratio, but we present a natural algorithm with tight analysis and pose challenges for further research. The main open problem is whether there exists a metric space on which the ∞-server problem is not competitive.

In Section 2.4 we show how a variety of known algorithms such as the work function and balancing algorithms fail for the ∞-server problem, even on the line metric. We focus in particular on a class of speed-adjusted variants of the Double Coverage algorithm.

Finally, we present a useful reduction from arbitrary metric spaces to bounded subspaces in Section 2.5. In particular, the ∞-server problem on the line is competitive if and only if it is competitive for the special case where requests are restricted to some bounded interval further away from the source.

2.1.1 Preliminaries

Let $M = (M, d)$ be a metric space and $s \in M$ a point, called the *source*. In the *∞-server problem on* (M, s), an unbounded number of servers starts at point s and serves a finite sequence $\sigma = (\sigma_0 = s, \sigma_1, \sigma_2, \ldots, \sigma_m)$ of requests $\sigma_i \in M$. Serving a request entails moving one of the servers to it. The cost is the total distance traveled by the servers.

We drop s in the notation if the location of the source is not relevant or understood. We refer to the action of moving a server from the source to another point as *spawning*. Throughout this chapter we use the letter d for the metric associated with the metric space.

All algorithms considered in this chapter are deterministic. An algorithm is called *lazy* if it moves only one server to serve a request at an unoccupied point and moves no server if the requested point is already covered. An algorithm is called *local* [CEFJ14] if it moves a server from a to b only if there is no server at some other point c on a shortest path from a to b, i.e., with $d(a, b) = d(a, c) + d(c, b)$. It is easy to see that any algorithm can be turned into a lazy and local algorithm without increasing its cost.

Extending the notation from Section 1.1.1, we may sometimes write $cost_A(s;\sigma)$ and $opt(s;\sigma)$ instead of $cost_A(\sigma)$ and $opt(\sigma)$ to emphasize the starting location s of servers. Moreover, we may replace opt by opt_h or opt_∞, where the index denotes the number of servers available to the offline algorithm.

The following two propositions will be useful later in the chapter.

Proposition 2.1.1. *If for every metric space there exists a competitive algorithm for the ∞-server problem, then there exists a universal competitive ratio ρ such that the ∞-server problem is* strictly *ρ-competitive on every metric space.*

Proof. We first show the existence of ρ such that the ∞-server problem is ρ-competitive (strictly or not) on every metric space. Suppose such ρ does not exist, then for every $n \in \mathbb{N}$ we can find a metric space M_n containing some point s_n such that the ∞- server problem on (M_n, s_n) is not n-competitive. Consider the metric space obtained by taking the disjoint union of all spaces M_n and gluing all the points s_n together. The ∞-server problem would not be competitive on this metric space, in contradiction to the assumption.

Analogously we can also find a universal constant c such that a ρ-competitive algorithm A with additive constant c in the definition of competitive ratio exists for general metrics. Without loss of generality, A is scale-invariant in the sense that if all distances are scaled by the same factor, it does not affect the decisions of A. Suppose A were not strictly ρ-competitive. Then there exists a request sequence σ in some metric space such that $cost_A(\sigma) > \rho \cdot opt(\sigma) + \epsilon$ for some $\epsilon > 0$. Scaling all distances by a large enough factor, the additive term would become greater than c, a contradiction. □

With a very similar argument we get:

Proposition 2.1.2. *Let $k = k(h)$ be a function of h. Suppose that for every metric space M and for all h there exists an $O(1)$-competitive algorithm for the (h,k)-server problem on M. Then there exists a universal competitive ratio ρ such that the (h,k)-server problem is strictly ρ-competitive on every metric space if all servers start at the same point.*

2.2 Equivalence of ∞-server and (h,k)-server

The goal of this section is to show the following tight connection between the ∞-server problem and the weak adversaries version of the k-server problem. Although we provide a proof for deterministic algorithms only, we remark that the proof can be extended to randomized algorithms as well.

Theorem 2.2.1. *The following are equivalent:*

(a) The ∞-server problem is competitive.

(b) The (h,k)-server problem is $O(1)$-competitive as $k/h \to \infty$.

(c) For each h there exists $k = O(h)$ so that the (h,k)-server problem is $O(1)$-competitive.

The three statements above are also equivalent if we fix the metric space to be the real line.

The implication "(c) $\Longrightarrow$ (b)" is trivial. The proof of the equivalence theorem consists in its core of two reductions. Theorem 2.2.2 contains the easier of the two reductions, which is from the ∞-server problem to the k-server problem against weak adversaries ("(b) $\Longrightarrow$ (a)"). By Proposition 2.1.1 and Proposition 2.1.2, it suffices to consider only strictly competitive algorithms. Theorem 2.2.3 proves the converse reduction for general metric spaces, and Theorem 2.2.6 specializes it to the line ("(a) $\Longrightarrow$ (c)").

As a corollary of the theorem we get the non-trivial implication "(b) $\Longrightarrow$ (c)", a potentially useful statement towards resolving the major open problem about the (h,k)-server problem: "Is Statement (b) true?" This highlights the importance of the ∞-server problem.

Theorem 2.2.2. *Fix a metric space M and consider algorithms with all servers starting at some $s \in M$. If for every h there exists $k = k(h)$ such that the (h,k)-server problem on M is strictly ρ-competitive, for some constant ρ, then there exists a strictly ρ-competitive online algorithm for the ∞-server problem on M.*

Proof. Let $A_{k(h)}$ denote an online algorithm with $k(h)$ servers that is strictly ρ-competitive against an optimal algorithm opt_h for h servers, i.e.,

$$cost_{A_{k(h)}}(\sigma) \leq \rho \cdot opt_h(\sigma) \tag{2.1}$$

for every request sequence σ. Without loss of generality, algorithm $A_{k(h)}$ is lazy.

For every request sequence σ, consider the equivalence relation $\equiv_\sigma$ on natural numbers in which $h \equiv_\sigma h'$ if and only if $A_{k(h)}$ and $A_{k(h')}$ serve σ in exactly the same way (i.e., make exactly the same moves). To every σ, we associate an equivalence class $H(\sigma)$ of $\equiv_\sigma$ such that

- $H(\sigma)$ is infinite,
- $H(\sigma r) \subseteq H(\sigma)$, for every request r.

This is done inductively on the length of σ (in a manner reminiscent of König's lemma) as follows: For the base case when σ is the empty request sequence, $H(\sigma) = \mathbb{N}$. For the induction step, suppose that we have defined $H(\sigma)$. Consider the equivalence classes of $\equiv_{\sigma r}$, a refinement of the equivalence classes of $\equiv_\sigma$. Since there are only finitely many possible ways to serve r, they partition $H(\sigma)$ into finitely many parts. At least one of these parts is infinite and we select it to be $H(\sigma r)$; if there is more than one such set, we select one arbitrarily, say the lexicographically first.

Given such a mapping H, we define the online algorithm A_∞ which serves every σ in the same way as all the online algorithms $A_{k(h)}$ for $h \in H(\sigma)$. The second property of H guarantees that A_∞ is a well-defined online algorithm.

By construction, $cost_{A_\infty}(\sigma) = cost_{A_{k(h)}}(\sigma)$ for every $h \in H(\sigma)$. To finish the proof, observe that since $H(\sigma)$ is infinite, it contains some h greater than the length of σ, and for such an h we have $opt_\infty(\sigma) = opt_h(\sigma)$. Substituting these in (2.1), we see that A_∞ is strictly ρ-competitive. □

We now show the reduction from the k-server problem against weak adversaries to the ∞-server problem on general metric spaces.

Theorem 2.2.3. *If the ∞-server problem on general metric spaces is strictly $\tilde{\rho}$-competitive, then there exists a constant ρ such that the (h,k)-server problem is ρ-competitive, for $k = O(h)$. In particular, for every $\epsilon > 0$, we can take $\rho = (3+\epsilon)\tilde{\rho}$ and any $k \geq (1 + 1/\epsilon)\tilde{\rho}h$.*

Proof. Fix some metric space M and a point $s \in M$. We will describe a strictly ρ-competitive algorithm for the (h,k)-server problem on M for the case that all servers start at s. This implies a (not necessarily strictly) ρ-competitive algorithm for any initial configuration.

The idea is to simulate a strictly $\tilde{\rho}$-competitive ∞-server algorithm, but whenever it would spawn a $(k+1)$st server, we bring all servers back to the origin and restart the algorithm. The problem is that the overhead cost for returning the servers to the origin may be very high. To compensate for this, we assume that every time the servers return to the origin, they pretend to start from a different point further away from the origin. This motivates the following notation:

Definition 2.2.4. Given a metric M, a point $s \in M$, and a value $w \geq 0$, we will use the notation $M_{s\oplus w}$ to denote the metric derived from M by increasing the distance from s to every other point by w; we will also denote the relocated point by $s \oplus w$.

Let A_∞ denote a strictly $\tilde{\rho}$-competitive online algorithm for the ∞-server problem. We now define an online algorithm A_k for k servers (all starting at s).

Definition 2.2.5 (A_k derived from A_∞)**.** Algorithm A_k runs in phases with the initial phase being the 0th phase. At the beginning of every phase, all servers of A_k are at s. In every phase i, A_k simulates the ∞-server algorithm A_∞, whose servers start at $s \oplus w_i$ for some $w_i \geq 0$. The parameters w_i are determined online, and initially $w_0 = 0$. Whenever A_∞ spawns a server from $s \oplus w_i$, algorithm A_k spawns a server from s.

The phase ends just before A_∞ spawns its $(k+1)$st server or when the request sequence ends. In the former case, all servers of A_k return to s to start the $(i+1)$st

phase. To determine the starting point of the simulated algorithm of the next phase, we set

$$w_{i+1} = \epsilon \, \frac{opt_h(s; \sigma_i)}{h} \,, \tag{2.2}$$

where σ_i is the sequence of requests during phase i.

Let n be the number of phases. The cost of A_k for the requests in phase $i < n$ equals $cost_{A_\infty}(s \oplus w_i; \sigma_i) - kw_i$; the last term is subtracted because the k servers do not have to actually travel the distance between $s \oplus w_i$ and s. However for the last phase no such term can be subtracted since we do not know how many servers are spawned during the phase, and we can only bound the cost from above by $A_\infty(s \oplus w_n; \sigma_n)$. The cost of returning the servers to s at the end of a phase can at most double the cost during the phase.

From this, we see that the total cost of A_k in phase i is

$$cost^{(i)} \leq \begin{cases} 2\,(A_\infty(s \oplus w_i; \sigma_i) - kw_i) & \text{for } i < n \\ A_\infty(s \oplus w_n; \sigma_n) & \text{for } i = n \,. \end{cases}$$

Since A_∞ is strictly $\tilde{\rho}$-competitive, we have

$$\begin{aligned} cost_{A_\infty}(s \oplus w_i; \sigma_i) &\leq \tilde{\rho} \cdot opt_\infty(s \oplus w_i; \sigma_i) \\ &\leq \tilde{\rho} \cdot opt_h(s \oplus w_i; \sigma_i) \\ &\leq \tilde{\rho} \cdot (opt_h(s; \sigma_i) + hw_i) \end{aligned}$$

and substituting this in the expression for the cost, we can bound the total cost by

$$\begin{aligned} cost_{A_k}(s; \sigma) &= \sum_{i=0}^{n} cost^{(i)} \\ &\leq 2 \sum_{i=0}^{n-1} \left(\tilde{\rho} \cdot (opt_h(\sigma_i; s) + hw_i) - kw_i\right) + \tilde{\rho} \cdot (opt_h(s; \sigma_n) + hw_n) \\ &= 2 \sum_{i=0}^{n-1} \left(\tilde{\rho} \cdot opt_h(s; \sigma_i) - (k - \tilde{\rho}h)w_i\right) + \tilde{\rho} \cdot opt_h(s; \sigma_n) + \tilde{\rho}hw_n \,. \end{aligned}$$

The parameters w_i and k were selected so that the summation telescopes, and we are left with

$$\begin{aligned} cost_{A_k}(s;\sigma) &\leq 2\,\tilde{\rho}\; opt_h(s;\sigma_{n-1}) + \tilde{\rho}\; opt_h(s;\sigma_n) + \tilde{\rho}\,\epsilon\; opt_h(s;\sigma_{n-1}) \\ &\leq (3+\epsilon)\,\tilde{\rho}\; opt_h(s;\sigma)\,. \end{aligned}$$

□

The previous reduction requires the ∞-server problem to be competitive on *every* metric space. The following variant only requires the ∞-server problem to be competitive on the line.

Theorem 2.2.6. *If the ∞-server problem on the line is ρ-competitive, then for every $h \in \mathbb{N}$ and $\epsilon > 0$, the (h,k)-server problem on the line is $(3+\epsilon)\rho$-competitive, when $k \geq 2\lceil(1+1/\epsilon)\rho h\rceil$.*

Proof. A straightforward adaptation of the proof of the previous theorem shows the existence of a $(3+\epsilon)\rho$-competitive algorithm for the interval $[0,\infty)$, when $k \geq 2(1+1/\epsilon)\rho h$. By doubling the number of online servers so that half of them are used in each half-line, we get a $(3+\epsilon)\rho$-competitive algorithm for the entire line, when $k \geq 2\lceil(1+1/\epsilon)\rho h\rceil$.

Note that the proof assumes strictly competitive algorithms. But, by a straightforward scaling argument, if the ∞-server problem on the line is ρ-competitive, then it is also strictly ρ-competitive. This in turn implies a strictly ρ-competitive online algorithm for $M_{0\oplus w}$, since this space is isometric to the subspace $\{-w\} \cup (0,\infty)$ of the line. □

In the next section we give upper and lower bounds on the competitive ratio of the ∞-server problem on some particular metric spaces.

2.3 Upper and lower bounds

Unlike the k-server problem, which is 1-competitive if and only if the metric space has at most k points and conjectured k-competitive otherwise, the situation is more diverse for the ∞-server problem. For example, on uniform metric spaces the problem is trivially 1-competitive even if the metric space consists of uncountably

many points: An optimal strategy in this case is to spawn a server to every requested point. More generally, this strategy achieves a finite competitive ratio on any metric space where distances are bounded from below and above by positive constants. This suggests that statements about the competitive ratio for the ∞-server problem cannot be as simple as the (conjectured) dichotomy for the k-server problem, which depends only on the number of points of the metric space. In this section we derive bounds on the competitive ratio for particular classes of metric spaces.

2.3.1 Weighted trees

We consider the ∞-server problem on metric spaces that can be modeled by edge-weighted trees. The points of the metric space are the nodes of the tree, and the distance between two nodes is the sum of edge weights along their connecting path. We choose the source of the metric space as the root of the tree, and define the depth of the tree as the maximal number of edges from the root to a leaf. The number of nodes can be infinite (otherwise the ∞-server problem is trivially 1-competitive), but we assume the depth to be finite.

An upper bound on the competitive ratio of such trees follows easily from an upper bound for the (h, k)-server on such trees [BEJK19] and the equivalence theorem:

Theorem 2.3.1. *The competitive ratio of the ∞-server problem on trees of depth D is at most $O(2^D \cdot D)$.*

Proof. Bansal et al. [BEJK19] showed that the competitive ratio of the (h, k)-server problem on trees of depth D is at most $O(2^D \cdot D)$ provided that k/h is large enough. Inspection of the proof in [BEJK19] shows that if all servers start at the root, it is in fact strictly $O(2^D \cdot D)$-competitive. Thus, Theorem 2.2.2 implies the result for the ∞-server problem. □

As an important special case, we see that the ∞-server problem is $O(1)$-competitive on weighted stars; rooting a weighted star at the source leaf, it becomes a tree of depth 2.

2.3.2 Non-discrete spaces and spaces with small infinite subspaces

The following theorem gives a lower bound of 3.146 on the competitive ratio of the ∞-server problem on any metric space containing an infinite subspace of a diameter that is small compared to the subspace's distance from the source. For example, every non-discrete metric space has this property, since non-discrete metric spaces contain infinite subspaces of arbitrarily small diameter. The theorem is a generalization of such a lower bound established in [CIN+01] for a variant of the paging problem where cache cells can be bought. Crucial parts of the subsequent proof are as in [CIN+01].

Theorem 2.3.2. *Let M be a metric space containing an infinite subspace $M_0 \subset M$ of finite diameter δ and a point $s \in M \setminus M_0$ such that the infimum Δ of distances between s and points in M_0 is positive. Let $\lambda > 3.146$ be the largest real solution to*

$$\lambda = 2 + \ln \lambda \,. \tag{2.3}$$

The competitive ratio of any deterministic online algorithm for the ∞-server problem on (M, s) is bounded from below by a value that converges to λ as $\Delta/\delta \to \infty$. In particular, the competitive ratio is at least λ if $M \setminus \{s\}$ contains a non-discrete part.

Proof. By scaling the metric, we can assume that $\delta = 1$. Let $p_1, p_2, p_3, \ldots$ be infinitely many distinct points in M_0.

Fix some lazy deterministic online algorithm A. We consider the request sequence that always requests the point p_i with i minimal such that p_i is not occupied by a server of A. We call a move of a server between two points in M_0 *local* (i.e., every move that does not spawn is local). Let f_j be the cumulative cost of local moves incurred to A until it spawns its jth server. Let σ_k be this request sequence that is stopped right after A spawns its kth server, for some large k. The total online cost is

$$cost_A(\sigma_k) \geq k\Delta + f_k \,. \tag{2.4}$$

Let $h = \lceil k/\lambda \rceil$. We consider several offline algorithms that start behaving the same way, so we think of it as one algorithm initially that is forked into several

algorithms later. The offline algorithms make use of only h servers and they begin by spawning them to the points $p_1, \ldots, p_h$. They do not need to move any servers until A spawns its hth server. Whenever A spawns its jth server for some $j \geq h$, every offline algorithm is forked to h distinct algorithms: Each of them moves a different server to p_{j+1} (to prepare for the next request, which will be at p_{j+1}). We will keep the invariant that each offline algorithm already has a server at the next request. To this end, whenever A does a local move from p to p', every offline algorithm that does not have a server at p moves a server from p' to p; note that the algorithm had a server at p' by the invariant, and the next request will be at p.

When A has j spawned servers ($j \geq h$), the offline algorithms are in $\binom{j}{h-1}$ different configurations, each of which occurs equally often among them. If A does a local move from p to p', there are $\binom{j-1}{h-1}$ different offline configurations for which a local move is made in the opposite direction. Thus, for each local move by A while having j spawned servers in total, a portion $\binom{j-1}{h-1}/\binom{j}{h-1} = \frac{j-h+1}{j}$ of the offline algorithms move a server in the opposite direction for the same cost.

We use the average cost of all these offline algorithms as an upper bound on the optimal cost. The cost of spawning h servers is at most $h(\Delta+1)$, and the average cost while A has j spawned servers (for $j = h, \ldots, k-1$) is at most $\frac{j-h+1}{j}(f_{j+1}-f_j)+1$ (with the "+1" coming from the move when offline algorithms fork). Hence,

$$\begin{aligned} opt(\sigma_k) &\leq h(\Delta+1) + k - h + \sum_{j=h}^{k-1} \frac{j-h+1}{j}(f_{j+1}-f_j)\,, \\ &\leq h\Delta + k + \frac{k-h}{k-1} f_k - \frac{f_h}{h} - \sum_{j=h+1}^{k-1} \frac{h-1}{j(j-1)} f_j\,, \end{aligned}$$

Note that $\frac{f_k}{k}$ is bounded from above because otherwise A would not be competitive, and it is bounded from below by 0. Thus, $L = \liminf_{k\to\infty} \frac{f_k}{k}$ exists. In the following we use the asymptotic notation $o(1)$ for terms that disappear as $k \to \infty$. We can choose arbitrarily large values of k such that $\frac{f_k}{k} = L + o(1)$. Since $h = \lceil k/\lambda \rceil$, we have $\frac{f_j}{j} \geq L + o(1)$ for all $j \geq h$. Moreover, $\sum_{j=h+1}^{k-1} \frac{1}{j-1} = \ln(\lambda) + o(1)$. This allows

us to simplify the previous bound to

$$\begin{aligned} opt(\sigma_k) &\leq \frac{k}{\lambda}\Big(\Delta + \lambda + \big(\lambda - 1 - \ln(\lambda)\big)L + o(1)\Big) \\ &= \frac{k}{\lambda}\big(\Delta + L + \lambda + o(1)\big)\,, \end{aligned}$$

where the last step uses equation (2.3).

The competitive ratio is at least

$$\begin{aligned} \frac{cost_A(\sigma_k) + O(1)}{opt(\sigma_k)} &\geq \frac{k\Delta + f_k + O(1)}{\frac{k}{\lambda}\big(\Delta + L + \lambda + o(1)\big)} \\ &= \lambda \cdot \frac{\Delta + L}{\Delta + L + \lambda} + o(1)\,. \end{aligned}$$

The fraction in the last term tends to 1 as $\Delta \to \infty$. □

This bound is tight due to a matching upper bound in [CIN+01] that shows (translated to the terminology of the ∞-server problem) that a competitive ratio of λ can be achieved on the weighted star metric where all pairwise distances are 1 except that the source is at some larger distance Δ from the other points.

The previous theorem together with the equivalence theorem also allows us to obtain a new lower bound for the k-server problem against weak adversaries.

Corollary 2.3.3. *For sufficiently large h, there is no 3.146-competitive algorithm for the (h,k)-server problem on the line, even if $k \to \infty$.*

Proof. By a scaling argument it is easy to see that if the ∞-server problem on the line is ρ-competitive, then it is also *strictly* ρ-competitive. Thus, the statement follows from Theorems 2.2.2 and 2.3.2. □

This improves upon both the previous best known lower bounds of 2 for this problem on the line [BE98, p. 175] and 2.4 on general metric spaces [BEJK19]

2.3.3 Layered graphs

A *layered graph of depth* D is a graph whose (potentially infinitely many) nodes can be arranged in layers $0, 1, \ldots, D$ so that all edges run between adjacent layers and each node – except for a single node in layer 0 – is connected to at least one node of the previous layer. The induced metric space is the set of nodes with the distance being the minimal number of edges of a connecting path. For the purposes of the ∞-server problem, the single node in layer 0 is the source. We assume $D \geq 2$ to avoid trivial cases.

Note that a connected graph is layered if and only if it is bipartite. Moreover, any graph can be embedded into a bipartite graph by adding a new node in the middle of each edge. So essentially, layered graphs capture *all* graph metrics.

Let *Move Only Outwards* (MOO) be some lazy and local algorithm for the ∞-server problem on layered graphs that moves servers along edges only in the direction away from the source. Not surprisingly, the competitive ratio of this simple algorithm is quite bad and we show that it is exactly $D - 1/2$. Nonetheless, at least for $D \leq 3$ this is actually the optimal competitive ratio.

Theorem 2.3.4. *The competitive ratio of* MOO *is exactly* $D - \frac{1}{2}$.

Proof.

Upper bound:

Consider some final configuration of the algorithm after a request sequence σ. Let n_j be the number of servers in the jth layer. Then

$$cost_{\mathrm{MOO}}(\sigma) = \sum_{j=1}^{D} j n_j.$$

To obtain an upper bound on $opt(\sigma)$, observe that every node occupied by MOO in the final configuration must have been visited by an offline server at least once. We account an offline cost of 1 for each visit of a node on layers $1, \ldots, D-2$ and an offline cost of 2 for each visit of a node on layer D. This cost of 2 covers the last two edge-traversals before visiting the layer-D-node, so this may include serving a

request on layer $D-1$. If $n_{D-1} > n_D$, then we can account another $n_{D-1} - n_D$ cost for visiting the remaining at least $n_{D-1} - n_D$ requested nodes on layer $D-1$. In summary,

$$opt(\sigma) \geq \sum_{j=1}^{D-2} n_j + 2n_D + (n_{D-1} - n_D)^+$$

where $(n_{D-1} - n_D)^+ := \max\{0, n_{D-1} - n_D\}$. The upper bound on the competitive ratio follows since

$$\begin{aligned}
\frac{cost_{\text{MOO}}(\sigma)}{opt(\sigma)} &\leq \frac{\sum_{j=1}^{D} jn_j}{\sum_{j=1}^{D-2} n_j + 2n_D + (n_{D-1} - n_D)^+} \\
&\leq \frac{(D-2)\sum_{j=1}^{D-2} n_j + (2D-1)n_D + (D-1)(n_{D-1} - n_D)^+}{\sum_{j=1}^{D-2} n_j + 2n_D + (n_{D-1} - n_D)^+} \\
&\leq D - \frac{1}{2}\,.
\end{aligned}$$

Lower bound:

Let $k, n \in \mathbb{N}$ be some large integers. We construct the following graph: Layers $0, \ldots, D-2$ consist of one node each and layers $D-1$ and D consist of infinitely many nodes each, denoted $u_0, u_1, u_2, \ldots$ and $v_0, v_1, v_2, \ldots$ respectively. For each $i \in \mathbb{N}_0$, the k nodes $v_{ik}, v_{ik+1} \ldots, v_{(i+1)k-1}$ are adjacent to each of the $2k$ nodes $u_{ik}, u_{ik+1}, u_{(i+2)k-1}$ and to no other nodes. The set of remaining edges is uniquely determined by the fact that this is a layered graph of depth D.

The request sequence consists of n rounds $0, 1, \ldots, n-1$, where each request in round i is at a node from the list $u_{ik}, u_{ik+1}, \ldots, u_{(i+1)k-1}, v_{ik}, v_{ik+1}, \ldots, v_{(i+1)k-1}$. Round i starts with requests on the nodes $u_{ik}, u_{ik+1}, \ldots, u_{(i+1)k-1}$. Then, for $j = 0, \ldots, k-1$, the adversary first requests v_{ik+j} and then requests whichever node from $u_{ik}, u_{ik+1}, \ldots, u_{(i+1)k-1}$ has been left by a MOO-server to serve the request at v_{ik+j}. Note that by definition of MOO and the graph, the server it moves to v_{ik+j} does indeed come from $u_{ik}, u_{ik+1}, \ldots, u_{(i+1)k-1}$.

In round i, MOO first pays $k(D-1)$ to move k servers to $u_{ik}, u_{ik+1}, \ldots, u_{(i+1)k-1}$ and then, for each $j = 0, \ldots, k-1$, it pays 1 to move to v_{ik+j} and $D-1$ to spawn a

new server at the group $u_{ik}, u_{ik+1}, \ldots, u_{(i+1)k-1}$. Over n rounds this makes a total cost of $n(k(D-1)+k(1+D-1)) = nk(2D-1)$.

The offline algorithm can serve requests as follows: The requests at the nodes $u_{ik}, \ldots, u_{(i+1)k-1}$ at the beginning of round i are served by spawning if $i = 0$ (for cost $(D-1)k$) and by sending servers from $v_{(i-1)k}, \ldots, v_{ik-1}$ if $i \geq 1$ (for cost k). The request at v_{ik} is served by spawning a server (cost D) and the requests at $v_{ik+1}, \ldots, v_{ik+k-1}$ are served by sending a server from a node in $u_{ik}, \ldots, u_{(i+1)k-1}$ that will not be requested any more (cost 1 each, so $k-1$ per round). Over n rounds, this adds up to an offline cost of $(D-1)k + (n-1)k + n(D+k-1) = 2nk + (D-2)k + n(D-1)$. The ratio of online and offline cost is

$$\frac{nk(2D-1)}{2nk + (D-2)k + n(D-1)} = \frac{2D-1}{2 + \frac{D-2}{n} + \frac{D-1}{k}},$$

which gets arbitrarily close to $D - \frac{1}{2}$ for n and k large enough. □

Theorem 2.3.5. *The competitive ratio of the ∞-server problem on layered graphs of depth D is exactly 1.5 for $D = 2$, exactly 2.5 for $D = 3$ and at least 3 for $D \geq 4$.*

Proof. For $D = 2$, the only possibility to move a server closer to the source is from layer 2 to layer 1. But since spawning to layer 1 is at least as good, we can restrict our attention to algorithms of the type Moo. The result follows from Theorem 2.3.4.

For $D = 3$, the upper bound follows from Theorem 2.3.4. It remains to show the lower bounds for $D \geq 3$.

Fix some large integers $k, n \in \mathbb{N}$. Consider the following layered graph of depth D. For $i = 0, \ldots, D-1$ there exists a node ℓ_i in layer i. The remaining nodes are defined inductively as all nodes obtained by the following two rules:

- There exist a set S_0 of $2k$ nodes and sets U^{S_0} and V^{S_0} of k nodes.
- Let S be a set of $2k$ nodes such that U^S and V^S exist. Then for each $S' \subset S \cup V^S$ of size $2k$ there are sets $U^{S'}$ and $V^{S'}$ of k nodes.

The nodes in the sets U^S are in layer $D-1$, the nodes in S_0 and in the sets V^S are in layer D. For a node in some set U^S, the set of adjacent nodes in layer D is $S \cup V^S$. The remaining edges are so that this is a layered graph with the layers as specified.

For purposes of the analysis below, we further define a *generation* of a node as follows: The nodes $\ell_0, \ldots, \ell_{D-1}$ and the nodes in S_0 have generation 1. The generation of nodes in U^S and V^S is the maximal generation of any node in S plus 1.

Let A be some online algorithm. We assume without loss of generality that A is lazy and local.

The adversary chooses the following request sequence against A. First, it request the nodes in S_0 until A has a server at each of them. The adversary also moves $2k$ servers to these nodes. The adversary uses only these $2k$ servers for the entire sequence of requests. The remainder of the requests consists of several rounds. We will keep the invariant that at the beginning of the ith round, the $2k$ adversary servers occupy a set S for which U^S and V^S (with nodes of generation $i+1$) exist, and the online servers occupy nodes of generation at most i. Clearly this holds before the first round. Let $U^S = \{u_1, \ldots, u_k\}$ and $V^S = \{v_1, \ldots, v_k\}$.

The requests of the ith round are divided into part a and part b, consisting of *steps* a.1,...,a.k, b.1,...,b.k that are executed in this order. Step a.j consists of the following one or two requests: First request u_j. If A moves a server from some $v \in S$ towards u_j, immediately request v. We can assume that online servers cover U^S after the end of part a (otherwise request nodes in U^S again at the end of part a until this is the case). Step b.j consists of the following two or three requests: First request v_j. Note that any path from a node of generation at most i to v_j contains a node in S, and from any node in S, the shortest paths to v_j include the ones along the nodes in U^S. Thus, since A is local, it will move a server from some $u \in U^S$ towards v_j. The second request of step b.j is at this node u and, if A moves a server from some $v \in S \cup V^S$ towards u, then the step contains a third request at v.

The adversary cost per round is at most $2k + 2$: For each $j = 1, \ldots, k$, there are at least j nodes in S that will *not* be requested during steps a.j, ..., a.k, b.1, ..., b.$(k-1)$. Hence, the adversary can serve all requests of part a for cost k by moving k servers from S towards U^S whilst keeping servers at all nodes of S that will be requested during the steps b.1,...,b.$(k-1)$. Similarly, it can serve the steps b.1,...,b.$(k-1)$ for cost $k-1$ by moving $k-1$ servers from U^S to V^S. The final step b.k of the round can be served at cost 3 using the last server in U^S to serve the requests and finish with all $2k$ offline servers in some set $S' \subseteq S \cup V^S$.

We analyze the online cost for the cases $D = 3$ and $D \geq 4$ separately.

If $D = 3$, then the cost for each step a.j is at least 2 and the cost for each step b.j is at least 3. Thus, the cost per round is at least $5k$. As k goes to infinity, the ratio of online and offline cost in each round converges to 2.5. As the number of rounds goes to infinity, the online and offline costs before the first round become negligible, which proves the lower bound of 2.5 for $D = 3$.

For $D \geq 4$, we use a potential Φ equal to the number of online servers in layer $D-1$. As we consider different subsequences of requests, we write $\Delta\Phi$ and $\Delta cost$ for the change of Φ and $cost_A$ respectively during this subsequence. During step a.j, either Φ does not change and the cost is at least 2, or Φ increases by 1 and the cost is at least 3. Thus, during step a.j we have $\Delta cost \geq 2 + \Delta\Phi$ and hence during part a we have $\Delta cost \geq 2k + \Delta\Phi$. During step b.$j$, either Φ decreases by 1 and the cost is at least 3, or Φ does not change and the cost is at least 4. Thus, during part b we have $\Delta cost \geq 4k + \Delta\Phi$. In total, this adds up to $\Delta cost \geq 6k + \Delta\Phi$ during the round. Over n rounds, this makes $\Delta cost \geq 6nk + \Delta\Phi \geq 6nk$ since Φ starts at 0 before the first round and remains nonnegative. As k and n go to infinity, the ratio of our bounds on online and offline cost converges to 3. □

It remains an open problem to close the gap between the lower bound of 3 and the upper bound of 3.5 for $D = 4$. More important is the question whether an algorithm better than Moo exist for large D, achieving a competitive ratio of less than $D - 1/2$ on layered graphs of depth D. Note that if no algorithm

with a competitive ratio of $O(1)$ as $D \to \infty$ exists, then the ∞-server problem on general metric spaces would not be competitive.

2.4 Algorithms with unbounded competitive ratio

We examine the performance of classical algorithms known for the k-server problem when applied to the ∞-server problem. The main focus of this section is a generalization of the Double Coverage algorithm with adjusted server speeds. This idea has proved successful for the (h, k)-server problem (and hence the ∞-server problem) on weighted trees [BEJK19]. However, neither of these algorithms is competitive for the ∞-server problem even on the line.

2.4.1 Work function algorithm

Recall the definition of the work function algorithm (WFA) from Section 1.2.1, which extends naturally to the ∞-server problem. Even though WFA achieves the best known competitive ratio for the k-server problem and has proved useful for many other online problems, it fails for the ∞-server problem. Intuitively, the weakness of WFA is that it chases the configuration of the optimal algorithm, and it makes no attempt to outnumber the offline algorithm by using more servers. But if the online algorithm uses only as many servers as the offline algorithm, the k-server lower bounds apply and yield an infinite competitive ratio for the ∞-server problem.

Proposition 2.4.1. *WFA is not competitive for the ∞-server problem on the line.*

Proof. Let the source be at 0 and, for some small $\delta > 0$, let $p_1, p_2, \dots \in [1, 1+\delta]$ be infinitely many distinct points. Consider the request sequence that always requests the point p_i with i minimal such that p_i is not occupied by an online server. Let σ_k be the prefix of this request sequence until WFA spawns its kth server.

We claim that the optimal way of serving σ_k is to bring k servers to the points $p_1, \dots, p_k$: Since spawning a new server costs at least 1 and using an old server costs at most δ, WFA spawns a kth server only if the cost of spawning k servers to the

points $p_1, \ldots, p_k$ is at least $1-\delta$ cheaper than the optimal cost of using $k-1$ servers to serve σ_k and end up in some fixed configuration with $k-1$ servers in the interval $[1, 1+\delta]$. Moreover, the latter cost differs by at most δ from the optimal cost of serving σ_k with $k-1$ servers (without the requirement to end up in a particular configuration). Hence, serving σ_k with k servers is at least $1-2\delta$ cheaper than doing so with $k-1$ servers. In particular, the optimal way to service σ_k is to bring k servers to the points $p_1, \ldots, p_k$, as claimed. Therefore, $opt(\sigma_k) = \sum_{i=1}^{k} p_i = k + o(1)$ as $\delta \to 0$.

Thus, the optimal offline cost increases by $1 + o(1)$ during the period when WFA has k spawned servers, and it increases by at least as much for an offline algorithm that is restricted to using k servers only. Let $cost_k$ be the cost incurred to WFA during this period. Due to the lower bound of k on the competitive ratio of the k-server problem, $cost_k$ is at least k times this increase of the optimal cost (up to an additive error of $o(1)$ as $\delta \to 0$), i.e., $cost_k \geq k + o(1)$. Thus, the total cost of WFA given the request sequence σ_n is at least

$$\sum_{k=1}^{n-1} cost_k = \Omega(n^2).$$

Meanwhile, the optimal cost is $opt(\sigma_n) = n + o(1)$. Letting n tend to infinity we obtain an unbounded competitive ratio. □

2.4.2 Balance and Balance2

The algorithm *Balance* serves a request r by sending a server x that minimizes the quantity $D_x + d(x, r)$, where D_x is the cumulative distance traveled by x so far and $d(x, r)$ is the distance between x and r. For the k-server problem, Balance is k-competitive on metric spaces with $k+1$ points [MMS88] and for weighted stars [CKPV91]. Young showed that for weighted paging against a weak adversary with h servers the competitive ratio of Balance is $k/(k-h+1)$ [You94]. On general metric spaces however, Balance has unbounded competitive ratio, even if $k = 2$ [MMS88]. It is therefore unsurprising that it is also not competitive for the ∞-server problem.

Proposition 2.4.2. *Balance is not competitive for the ∞-server problem on the line.*

Proof. Suppose all servers start at source 0 and consider the request sequence $r_0, r_1, r_2, \ldots, r_n$ where $r_i = 1 - i\epsilon$. As $\epsilon \to 0$, the optimal cost tends to 1 whereas the cost of Balance tends to $n + 1$. Since n can be arbitrarily high, this shows an unbounded competitive ratio. □

The intuitive problem of Balance is that it is not greedy enough. The algorithm *Balance2* by Irani and Rubinfeld [IR91] compensates for this weakness by giving more weight to the distance between the server and the request: To serve request r, Balance2 sends a server x that minimizes the quantity $D_x + 2d(x, r)$. Irani and Rubinfeld showed that, unlike Balance, Balance2 is competitive for two servers on arbitrary metrics (achieving a competitive ratio of at most 10) and they conjectured that it is also competitive for any other finite number of servers [IR91].

However, for the ∞-server problem this algorithm is also not competitive:

Proposition 2.4.3. *Balance2 is not competitive for the ∞-server problem on the line.*

Proof. Suppose the source is at 0 and fix some small constant $\epsilon > 0$. The request sequence consists of several phases, starting with phase 0. Phase i consists of alternating requests at $1 - 2i\epsilon$ and $1 - (2i + 1)\epsilon$. We will ensure that all requests of a phase are served by the same online server, and we call this the active server. As soon as the cumulative distance traveled by the active server exceeds $2 - (4i + 5)\epsilon$, the phase ends and a new phase begins. Note that this means that the active server of a phase will not be used to serve any request of a subsequent phase because, by definition of Balance2, the algorithm would rather spawn a new server. Thus, the first request of each phase is served by spawning a new server, which becomes the active server of that phase. While the cumulative distance of the active server is at most $2 - (4i + 5)\epsilon$ and since its distance from the next request of the phase is always exactly ϵ, its associated quantity $D_s + 2d(s, r)$ is at most $2 - (4i + 3)\epsilon$.

Hence, Balance2 rather uses this server during the phase instead of spawning a new server. Thus, it is indeed the active server that serves *all* requests of its phase.

Let n be the number of phases and choose ϵ small enough so that all requests are in the interval $[1/2, 1]$. Thus, the cost of Balance2 is $\Omega(n)$.

An offline algorithm could serve all requests with two servers only that move to 1 and $1-\epsilon$ initially and then back towards $1/2$, always covering the two points that are requested during a phase, resulting in an offline cost of less than 3. As n goes to infinity, the ratio between online and offline cost becomes arbitrarily large. □

2.4.3 Double Coverage variants

Perhaps more surprising than for WFA and balancing algorithms is that a class of algorithms extending the Double Coverage (DC) algorithm [CKPV91] is also not competitive for the ∞-server problem. Recall that the basic DC algorithm on the line serves each request by an adjacent server. If the request lies between two servers, both servers move towards it at equal speed until one of them reaches the request. A sensible extension of this algorithm seems to be to give different speeds to servers, so that they move away from the source faster than towards it.

We consider here only the half-line $[0, \infty)$ with the source at the left border 0. Let x_i be the position of the ith server from the right. We use the notation x_i both for its position and for the server itself. As servers do not overtake each other, x_i is the ith spawned server. Let $\mathcal{S} = \{s_i \geq 1 \mid i \in \mathbb{N} \text{ and } i \geq 2\}$ for a monotonic (non-decreasing or non-increasing) sequence of speeds s_i. The algorithm $\mathcal{S}$-DC is defined as follows:

- If there exist servers x_{i+1} and x_i to the left and right of the request, move them towards it with speeds s_{i+1} and 1 respectively until one of the two reaches it.

- If a request does not have a server to its right, move the rightmost server to the request.

If $s_i = 1$ for all i, this is precisely the original DC algorithm.

We will prove that $\mathcal{S}$-DC is not competitive. The intuitive reason is that servers move to the right either too slowly or too quickly: Imagine repeatedly requesting the same n points in some small interval away from the source, until $\mathcal{S}$-DC covers all n points. One case is that $\mathcal{S}$-DC spawns too slowly and is therefore defeated by an adversary covering these n positions immediately with n servers. In the other case, the adversary will also use n servers to cover the initial group of requests and then shift its group of servers slowly towards the source, always making requests at the new positions of these offline servers. As $\mathcal{S}$-DC tries to cover the new requests, it is tricked into spawning too many servers. Both cases lead to an unbounded competitive ratio.

The proof consists of several lemmas. The lemmas hold also for non-monotonic speeds and we use monotonicity only to easily combine the lemmas in the end.

A useful property of $\mathcal{S}$-DC is that its cost can be calculated using only the final positions of the servers.

Lemma 2.4.4. *Let $x_1 \geq x_2 \geq \ldots$ be the server positions of $\mathcal{S}$-DC after serving a sequence σ of requests. Then*

$$cost_{\mathcal{S}\text{-}DC}(\sigma) = \sum_{i=1}^{\infty} z_i x_i,$$

where

$$z_1 = 1 \tag{2.5}$$

$$z_i = \frac{z_{i-1}}{s_i} + 1 + \frac{1}{s_i}. \tag{2.6}$$

Proof. The position x_i of each server can be written as $x_i = r_i - l_i$ where r_i and l_i are the cumulative distances traveled by that server while moving to the right and left respectively. By definition of $\mathcal{S}$-DC, for all i we have

$$l_i = \frac{r_{i+1}}{s_{i+1}},$$

since any right move (apart from the rightmost server) is accompanied by a left move of another server. Observe that

$$\begin{aligned} cost_{\mathcal{S}\text{-}DC}(\sigma) &= \sum_{i=1}^{\infty}(r_i + l_i) \\ &= \sum_{i=1}^{\infty}(r_i + \frac{r_{i+1}}{s_{i+1}}) \\ &= \sum_{i=1}^{\infty} r_i + \sum_{i=2}^{\infty} \frac{r_i}{s_i} \\ &= r_1 + \sum_{i=2}^{\infty} r_i(1 + \frac{1}{s_i})\,. \end{aligned} \tag{2.7}$$

Similarly,

$$\begin{aligned} \sum_{i=1}^{\infty} z_i x_i &= \sum_{i=1}^{\infty} z_i(r_i - l_i) \\ &= \sum_{i=1}^{\infty} z_i r_i - \sum_{i=1}^{\infty} z_i \frac{r_{i+1}}{s_{i+1}} \\ &= z_1 r_1 + \sum_{i=2}^{\infty} r_i \left(z_i - \frac{z_{i-1}}{s_i}\right)\,. \end{aligned} \tag{2.8}$$

By equating (2.7) and (2.8) term by term, we get the desired recurrence for z_i. □

The next lemma takes care of the case when online servers spawn too slowly.

Lemma 2.4.5. *If the speeds in $\mathcal{S}$ satisfy $\liminf_{n\to\infty} \sqrt[n]{\prod_{i=2}^{n} s_i} = 1$ then $\mathcal{S}$-DC is not competitive.*

Proof. For this lower bound we have requests on n arbitrary positions in the interval $[1, 2]$, until $\mathcal{S}$-DC covers them all.

The optimal cost is at most $2n$. This can be achieved by spawning a fresh server for each requested position.

Since for every spawned online server we have $x_i \geq 1$, Lemma 2.4.4 yields

$$cost_{\mathcal{S}\text{-}DC}(\sigma) \geq \sum_{i=1}^{n} z_i.$$

Unraveling the recurrence we get that $z_i = 1 + \frac{2}{s_i} + \frac{2}{s_i s_{i-1}} + \ldots + \frac{2}{s_i \cdot \ldots \cdot s_2}$. Thus,

$$\begin{aligned} cost_{\mathcal{S}\text{-}DC}(\sigma) &\geq n + \sum_{i=1}^{n-1} \sum_{j=1}^{n-i} \frac{2}{\prod_{k=j+1}^{j+i} s_k} \\ &\geq \sum_{i=1}^{f(n)} \sum_{j=1}^{n-i} \frac{2}{\prod_{k=j+1}^{j+i} s_k} \,. \end{aligned} \tag{2.9}$$

where

$$f(n) := \left\lfloor \frac{n}{2 + 2\log_2 \prod_{i=2}^{n} s_i} \right\rfloor \leq \frac{n}{2} \,.$$

We argue that for each $i = 1, \ldots, f(n)$, it holds for at least half of the values of $j = 1, \ldots, n-i$ that $\prod_{k=j+1}^{j+i} s_k \leq 2$. Indeed, suppose this were not the case for some i. Let us partition the set $J = \{1, \ldots, n-i\}$ of j-values into subsets $J_0, \ldots, J_{i-1}$, where J_m contains precisely those numbers from J that are congruent to m modulo i. By assumption, we have $\prod_{k=j+1}^{j+i} s_k > 2$ for at least half the values $j \in J$, so this must also be true for at least half the values $j \in J_m$ for some m. However, this would mean that

$$\prod_{k=2}^{n} s_k \geq \prod_{j \in J_m} \prod_{k=j+1}^{j+i} s_k > 2^{|J_m|/2} \geq 2^{\lfloor \frac{n-i}{i} \rfloor / 2} \geq 2^{\frac{n}{2f(n)} - 1} \geq \prod_{i=2}^{n} s_i \,,$$

a contradiction because the second inequality is strict.

Thus, continuing from (2.9) we can further bound the online cost as

$$cost_{\mathcal{S}\text{-}DC}(\sigma) \geq f(n) \frac{n - f(n)}{2} \geq \frac{n f(n)}{4} \,.$$

Since the optimal cost is at most $2n$, the competitive ratio is at least $f(n)/8$. However, $f(n)$ is unbounded because

$$\frac{n}{2 + 2\log_2 \prod_{i=2}^{n} s_i} = \frac{1}{\frac{2}{n} + 2\log_2 \sqrt[n]{\prod_{i=2}^{n} s_i}}$$

and the denominator in the last term gets arbitrarily close to 0. □

The case of servers being spawned too aggressively is handled by the following lemma.

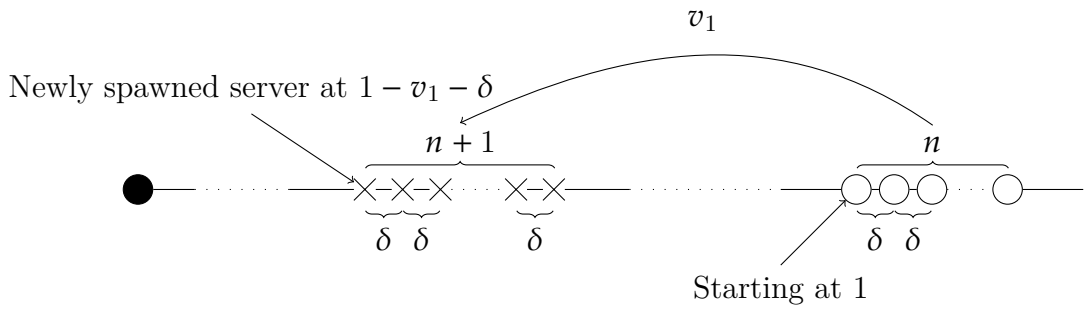

Figure 2.1: Servers (denoted by circles) and request locations (denoted by crosses) in the first phase when $\mathcal{S}$-DC spawns too aggressively.

Lemma 2.4.6. *If there exists an unbounded function $f(n)$ such that for each $k \in \mathbb{N}$ we have $\prod_{i=k}^{k+n} s_i \geq f(n)$, then $\mathcal{S}$-DC is not competitive. In particular, if $\liminf_{i\to\infty} s_i > 1$ then $\mathcal{S}$-DC is not competitive.*

Proof. Consider the setup of server and request locations depicted in Figure 2.1. We start by spawning n online servers grouped tightly, with the leftmost being at distance 1 from the source and a small gap δ between them. This is easily accomplished by repeating several requests on those points. Afterwards, we shift this group of n servers (by means of requests on new $n + 1$ points) to the left by v_1, chosen so that the $n + 1$ points are covered exactly by the n old servers plus a newly spawned one, which occupies the leftmost requested position $1 - v_1 - \delta$.

This is repeated again and again, shifting each time the leftmost n spawned servers a new distance v_k to the left via multiple requests on $n + 1$ positions. The goal each time is to pull a new server from the source *and* leave one behind forever, thus achieving an arbitrarily high competitive ratio for $\mathcal{S}$-DC variants that spawn servers too fast.

The offline cost can be calculated easily. The offline algorithm uses n servers to cover the first group of n requested points in the interval $[1, 1 + n\delta]$. Then it adds one more server and moves the group of $n + 1$ servers to the left to satisfy all of the following requests. At most, the group of offline servers will return close to the source, yielding an optimal cost of

$$opt(\sigma) \leq 2(n + 1)(1 + n\delta) = O(n), \tag{2.10}$$

where the last bound holds for δ sufficiently small.

To bound the online cost, we need to compute the values v_k first. Let ℓ_i^k and r_i^k denote the cumulative distance to the left and right respectively traveled by x_i during the left shift by v_k of the group $x_k, x_{k+1}, \ldots, x_{k+n-1}$. The nonzero values among these are

$$\begin{aligned}
\ell_k^k &= v_k \\
r_{k+1}^k &= v_k s_{k+1} \\
\ell_{k+1}^k &= v_k(1 + s_{k+1}) \\
r_{k+2}^k &= v_k(s_{k+2} + s_{k+1}s_{k+2}) \\
\ell_{k+2}^k &= v_k(1 + s_{k+2} + s_{k+1}s_{k+2}) \\
&\vdots \\
r_{k+n}^k &= v_k\left(s_{k+n} + s_{k+n-1}s_{k+n} + \ldots + \prod_{j=k+1}^{k+n} s_j\right) = v_k \sum_{i=k+1}^{k+n} \prod_{j=i}^{k+n} s_j\,. \qquad (2.11)
\end{aligned}$$

On the other hand, the new position of the server x_{k+n} pulled from the source during these moves is $1 - \sum_{i=1}^{k} v_i - k\delta$. Equating this with (2.11) and solving for v_k yields (assuming that n is even)

$$v_k = \frac{1 - \sum_{i=1}^{k-1} v_i - k\delta}{1 + \sum_{i=k+1}^{k+n} \prod_{j=i}^{k+n} s_j} \le \frac{1}{\frac{n}{2} \prod_{j=k+\frac{n}{2}}^{k+n} s_j} \le \frac{2}{n f(\frac{n}{2})}\,.$$

We will calculate the number of repetitions before the left border of the group of servers (just) passes $\frac{1}{2}$. If l is the number of repetitions, we have

$$\frac{1}{2} \le \sum_{k=1}^{l} (v_k + \delta) \le \frac{2l}{n f(\frac{n}{2})} + l\delta$$

and for sufficiently small δ this means that

$$l \ge \frac{n}{5} f\left(\frac{n}{2}\right)$$

If we do $l - 1$ repetitions, then each of them will pull a new server at least $1/2$ away from the source, resulting in an online cost of $\Omega(n) \cdot f(\frac{n}{2})$. As the offline cost is $O(n)$ and $f(n)$ is unbounded, the algorithm is not competitive. □

Since the sequence of speeds s_i is monotonic and bounded from below by 1, we have either $\lim_{i\to\infty} s_i = 1$, in which case Lemma 2.4.5 applies, or otherwise $\liminf_{i\to\infty} s_i > 1$ and Lemma 2.4.6 applies. In any case, the competitive ratio is unbounded:

Theorem 2.4.7. *Algorithm $\mathcal{S}$-DC is not competitive for any $\mathcal{S}$.*

2.5 Reduction to bounded spaces

In this section we show a reduction from the ∞-server problem on general metric spaces to bounded subspaces. Specifically, a metric space can be partitioned into "rings" of points whose distance from the source is between r^n and r^{n+1}, where $r > 1$ is fixed and $n \in \mathbb{Z}$. We show that if the ∞-server problem is strictly ρ-competitive on each ring, then it is competitive on the entire metric space.

Theorem 2.5.1. *Let M be a metric space and $s \in M$ and let $r > 1$. For $n \in \mathbb{Z}$, let $M_n = \{s\} \cup \{p \in M \mid d(s,p) \in [r^n, r^{n+1})\}$. If for each n the ∞-server problem on (M_n, s) is strictly ρ-competitive, then on (M, s) it is strictly $\frac{3r-1}{r-1}\rho$-competitive.*

Proof. Let A_n be a ρ-competitive algorithm for the ∞-server problem on (M_n, s).

For a request sequence σ, let σ_n be the subsequence of requests in M_n. Let A be the algorithm for (M, s) that uses different servers for each of the subsequences σ_n and serves them independently according to A_n.

The total online cost is

$$cost_A(\sigma) = \sum_n cost_{A_n}(\sigma_n) \le \rho \sum_n opt(\sigma_n).$$

To finish the proof, it suffices to show that

$$\sum_n opt(\sigma_n) \le \frac{3r-1}{r-1} opt(\sigma)\,. \tag{2.12}$$

Thus, we only need to analyze the offline cost. We do this for each offline server separately. Fix some offline server x. Let N_0 and N_1 be the minimal and maximal values of n such that x visits M_n. We can assume without loss of generality (by adding virtual points to the metric space) that whenever x moves from M_n to $M_{n'}$

for some $n < n'$, it travels across points $p_{n+1}, p_{n+2}, \ldots, p_{n'}$ with $d(s, p_i) = r^i$, and similarly for $n > n'$.

The movements of server x can be tracked by separate servers for the different sets M_n. We denote the servers responsible for M_n by $x_{n,1}, x_{n,2}, \ldots$ in the order in which they are spawned. When server x is in M_n, the last spawned server $x_{n,t}$ is exactly at the same position tracking the movement of x. When server x exits M_n at some point p at the boundary to M_{n-1} or M_{n+1}, this server $x_{n,t}$ freezes at p. When x later re-enters M_n at a point p' at the same boundary, then either $x_{n,t}$ moves to p' or a new server $x_{n,t+1}$ is spawned to p' – whichever is cheaper. The movement cost of the servers $x_{n,1}, x_{n,2}, \ldots$ can be partitioned into the *first spawn cost* to spawn $x_{n,1}$ at some (possibly virtual) point on the boundary of M_n and M_{n-1}, the *tracking cost* to follow the movement of x within M_n by the last spawned server $x_{n,t}$, and the *re-entering cost* incurred to relocate $x_{n,t}$ or spawn $x_{n,t+1}$ when x re-enters M_n.

The total tracking cost for all n is bounded by the distance traveled by x. The sum of first spawn costs for all n is $\sum_{n=N_0}^{N_1} r^n \leq \sum_{n=-\infty}^{N_1} r^n = r^{N_1+1}/(r-1)$, which is at most $\frac{r}{r-1}$ times the total distance travelled by x, because the latter is at least r^{N_1}.

Let us now consider the re-entering cost incurred when x enters M_n at p' after it previously exited at p. Then p and p' are at the boundary of M_n and M_{n+u} for $u \in \{-1, +1\}$. The re-entering cost is $\min\{d(p, p'), d(s, p')\}$. Let b be the distance traveled by x in M_{n+u} between the times when it is entered at p and when it is next exited. If this exiting is at p', then the re-entering cost is at most $d(p, p') \leq b$ by the triangle inequality. Otherwise, x exits M_{n+u} at a point p'' at the boundary of M_{n+u} and M_{n+2u}. If $u = 1$, then $b \geq d(p, p'') \geq d(s, p'') - d(s, p) = r^{n+2} - r^{n+1} = (r-1)r^{n+1}$ and the re-entering cost is at most $d(s, p') = r^{n+1}$. If $u = -1$, then $b \geq d(p, p'') \geq d(s, p) - d(s, p'') = r^n - r^{n-1} = \frac{r-1}{r} r^n$ and the re-entering cost is at most $d(s, p') = r^n$. In all cases, the re-entering cost is at most $\frac{r}{r-1} b$. Thus, the total re-entering cost of all servers x_n is at most $\frac{r}{r-1}$ times the total distance traveled by x.

Thus, the sum of first spawn, tracking and re-entering cost of the servers x_n is at most $\frac{3r-1}{r-1}$ times the distance traveled by x. This shows (2.12), giving the statement of the theorem. □

The last theorem can also be slightly generalized to the case where instead of *strict* ρ-competitiveness, an additive term proportional to r^n is allowed. It is not difficult to show the following specialization for the line, where the premise can be weakened to require competitiveness only on a single interval:

Corollary 2.5.2. *Let $0 < a < b$. The ∞-server problem is competitive on the line if and only if it is competitive on $(\{0\} \cup [a, b], 0)$.*

Another consequence of Theorem 2.5.1 is a reduction to spaces where the source is at a uniform distance from all other points. This models the case of a fixed cost for "buying" new servers.

Corollary 2.5.3. *Suppose there exists ρ so that the ∞-server problem is strictly ρ-competitive on any metric space where the distance from the source to any other point is the same. Then the ∞-server problem on general metric spaces is competitive.*

Proof. Follows from Theorem 2.5.1 by increasing the distance from s to the other points in M_n to r^{n+1}, making a multiplicative error of at most r. □

2.6 Open problems and conclusions

The most obvious open problem is whether the ∞-server problem is competitive on general metric spaces. A challenging special case is to resolve the question for the real line. Similarly, improving the MOO algorithm and settling the question for layered graphs remains open.

The connection between the ∞-server problem and the (h, k)-server problem seems to be particularly useful for improving lower bounds for the latter. Our lower bound of 3 for layered graphs of depth 4 as well as the lower bound of 3.146, which holds on some simple weighted stars, have quite different constructions and each of them alone exceeds the previous best lower bound for the (h, k)-server

problem. It seems plausible that extending one of these ideas may give an infinite lower bound for the (h, k)-server problem.

3
The k-Taxi Problem

3.1 Introduction

We study now the k-taxi problem. After some preliminary definitions in the next subsection, we present our results on the hard k-taxi problem in Section 3.2, which is the main part of this chapter. The equivalence of the easy k-taxi problem with the k-server problem is proved in Section 3.3.

3.1.1 Preliminaries

Let (M, d) be a metric space. A *configuration* is a multiset of k points in M, representing the positions of k taxis. In the *easy k-taxi problem* we are given an initial configuration and a sequence of requests $\sigma = (r_1, \ldots, r_n)$ where $r_i = (s_i, t_i) \in M^2$. An algorithm must move the taxis so as to serve these requests in order. A taxi serves request r_i by moving first to the start s_i of the request and then to the destination t_i. The cost is defined as the total distance traveled by all taxis.

The *hard k-taxi problem* is defined in the same way except that the movement of the serving taxi from s_i to t_i is not counted towards the cost. Thus, the cost comprises only the overhead distance traveled while not carrying a passenger.

We write $A(C; \sigma)$ to refer to the run of algorithm A on request sequence σ starting from initial configuration C. The corresponding sequence of configurations

is called a *schedule*. By $cost_A(C; \sigma)$ we denote its cost, although we will often omit the initial configuration from the notation and only write $cost_A(\sigma)$. An algorithm is *memoryless* if each decision depends only on the current configuration and the current request, but not the past configurations or requests.

Simple and relocation requests

Clearly, it cannot be a disadvantage for the adversary to replace a request (s, t) by two requests (s, s) and (s, t) because the adversary cost is unaffected by this and it forces the online algorithm to decide which taxi to send to s before learning the destination t. For the request (s, t), there is no decision to be made by the algorithm because it is clearly best to move the taxi already located at s due to the previous request. Thus, we can assume without loss of generality that the adversary gives a request of the form (s, t) with $s \neq t$ only if it is preceded by the request (s, s). We call requests of the form (s, s) *simple requests* and other requests *relocation requests*. We may also say there is a request at s to refer to a simple request (s, s).

Notice that if all requests are simple, the (easy and hard) k-taxi problem reduces to the k-server problem. Conversely, the k-taxi problem is the same as the k-server problem except that the adversary can choose to relocate a pair of online and offline servers if they occupy the same point. In the hard k-taxi problem, relocation is free, and in the easy k-taxi problem, the algorithm and adversary both pay the distance of the relocation.

Hierarchically separated trees (HSTs) [Bar96]

For $\alpha > 1$, an *α-HST* is a tree where all leaves have the same combinatorial depth and each node u has a weight w_u such that if v is a child of u, then $w_u = \alpha w_v$. The metric space of an HST consists of its leaves only, and the distance between two leaves is defined as the weight of their least common ancestor. This is the shortest path metric when the distance from u to its parent is defined as $\frac{\alpha-1}{2} w_u$ if u is an internal vertex and $\frac{\alpha}{2} w_u$ if u is a leaf. The significance of HSTs is that any metric space of n points can be probabilistically embedded into a distribution

over HSTs with distortion $O(\log n)$ [FRT04].[1] Thus, a ρ-competitive algorithm for HSTs yields a randomized $O(\rho \log n)$-competitive algorithm for general metrics.

For nodes x and y of a tree, we denote by P_{xy} their connecting path.

3.2 The hard k-taxi problem

Our first goal in this section is to prove the following theorem, giving tight bounds for the hard k-taxi problem on HSTs in two settings: for randomized algorithms against adaptive online adversaries, and for memoryless randomized algorithms against oblivious adveraries.

Theorem 3.2.1. *There is a $(2^k - 1)$-competitive memoryless randomized algorithm for the hard k-taxi problem on HSTs against adaptive online adversaries. This bound is tight in two senses: Any randomized algorithm A for the hard k-taxi problem on HSTs has competitive ratio at least $2^k - 1$ against adaptive online adversaries. If A is memoryless, then its competitive ratio is at least $2^k - 1$ even against oblivious adversaries.*

Since the randomized competitive ratio against adaptive online adversaries is at most a square-root better than the deterministic competitive ratio for any online problem [BBK+94], this also implies the existence of a 4^k-competitive deterministic algorithm for HSTs. More importantly, thanks to the probabilistic approximation of general n-point metrics by HSTs with distortion $O(\log n)$ [FRT04], Theorem 3.2.1 yields a competitive algorithm for the hard k-taxi problem on general finite metrics:

Corollary 3.2.2. *There is an $O(2^k \log n)$-competitive randomized algorithm for the hard k-taxi problem on metric spaces of n points.*

The upper bound of Theorem 3.2.1 is shown in Section 3.2.1, and the two matching lower bounds in Section 3.2.2 and Section 3.2.3.

[1] The literature contains several slightly different definitions of HSTs; they all share the property of approximating arbitrary metrics with distortion $O(\log n)$.

3.2.1 A memoryless algorithm for HSTs

We consider the following randomized algorithm, which we call FLOW, for the k-taxi problem on HSTs. Suppose a simple request arrives at a leaf s while the taxis are located at leaves $t_1, \ldots, t_k$. For each taxi we need to specify its probability to serve s. Let N be the Steiner tree of $s, t_1, \ldots, t_k$, i.e., the minimum subtree of the HST that spans these leaves. We can think of N as an electrical network by interpreting an edge of length R as a resistor with resistance R. When sending a current of size 1 through N from source s to sinks $t_1, \ldots, t_k$, the resistances determine what fraction of the current flows into which sink. Algorithm FLOW serves the request with a taxi from t_i with probability equal to the fraction of current flowing into t_i.[2] For a relocation request (s, t) after a simple request (s, s), FLOW uses the taxi already located at s.[3]

To formalize this algorithm, we need to give a mathematical description of how much current flows into each sink. Let $\mathcal{N}$ be the set of subtrees A of N comprising at least one edge and with the property that, if s_A is the (unique) node of A closest to s in N, then the leaves of A are a subset of $s_A, t_1, \ldots, t_k$. Formally, we view A as the set of all nodes and edges of this subtree. We denote by $\kappa(A) = |A \cap \{t_1, \ldots, t_k\}|$ the number of leaves of A where a taxi is located. Note that $N \in \mathcal{N}$ and $\kappa(N) \leq k$, with equality if and only if all taxis are located at different leaves. For each (sub)network $A \in \mathcal{N}$, we define (by induction on $\kappa(A)$) its resistance R_A and describe what fraction of the current entering A at s_A flows to which sink in A. If $\kappa(A) = 1$, then $A = P_{s_A t_i}$ for some i. In this case, the resistance of A is the length of this path, $R_A = d(s_A, t_i)$. Moreover, all current entering A at s_A flows to t_i.

If $\kappa(A) \geq 2$, then there is a unique node $s'_A \in A$ such that $A = P_{s_A s'_A} \cup B \cup C$ for some $B, C \in \mathcal{N}$ with $s_B = s_C = s'_A$, and $P_{s_A s'_A}$, $B \setminus \{s'_A\}$ and $C \setminus \{s'_A\}$ are disjoint

[2] Due to relocation requests, we may have several taxis at t_i. In this case, it does not matter which one we choose and what we describe is the combined probability of choosing one of them.

[3] FLOW is similar to the k-server algorithm RWALK by Coppersmith et al. [CDRS93]: Given a weighted graph (V, E) and interpreting edges as resistors as above, RWALK serves a request at s with a server from t with probability inversely proportional to the resistance of the resistor/edge $\{s, t\}$ (if the edge $\{s, t\}$ exists). RWALK is k-competitive for the metric of *effective* resistances on V.

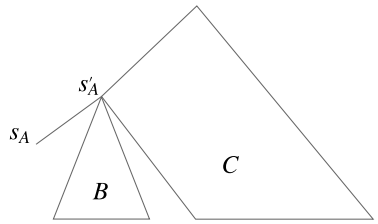

Figure 3.1: A subnetwork A with $\kappa(A) \geq 2$.

(see Figure 3.1). The resistance of A is defined as

$$R_A = d(s_A, s'_A) + \frac{R_B R_C}{R_B + R_C}. \tag{3.1}$$

Current entering A at s_A flows entirely to s'_A, where a $\frac{R_C}{R_B+R_C}$ fraction of it enters B and the remaining $\frac{R_B}{R_B+R_C}$ fraction enters C.[4]

Another interpretation of Flow is that we carry out a random walk starting at the request location, and once we reach a taxi, we select this taxi to serve the request. Whenever the random walk hits an intersection offering two possible directions to continue towards a taxi, either by entering a subtree B or a subtree C, we choose the subtree with probability inversely proportional to its resistance.

The following theorem yields the upper bound of Theorem 3.2.1. We do not actually need the HST property of geometrically decreasing weights, but only the weaker property that all requests are at the same distance from the root (in terms of the path metric extended to internal nodes).

Theorem 3.2.3. Flow *is* $(2^k - 1)$*-competitive against adaptive online adversaries for the hard* k*-taxi problem in the leaf-space of any tree with uniform root-leaf-distances.*

Proof. We use a potential equal to (2^k-1) times the value of a minimum matching M of algorithm and adversary configurations. Since M does not change upon relocation requests, we only need to consider simple requests. Whenever the adversary moves, the value of M increases by at most the distance moved by the adversary. Thus, we

[4]It is easy to verify that R_A and the current on each edge is well-defined, i.e. independent of the choice of B and C. Note that if s'_A has degree ≥ 2 in B or C, then the choice of B and C is not unique.

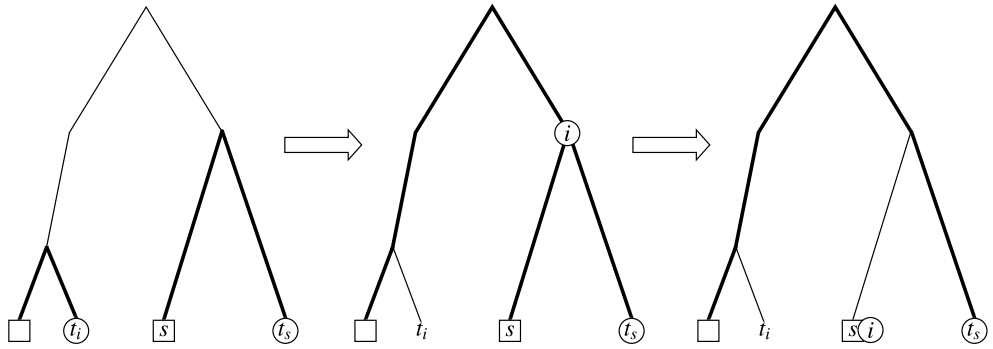

Figure 3.2: Change of the matching (bold lines) as taxi i moves from t_i to s.

only need to show for a simple request at a leaf s where the adversary already has a taxi that

$$\mathbb{E}(\mathit{cost}) + (2^k - 1)\,\mathbb{E}(\Delta M) \leq 0,$$

where the random variables *cost* and ΔM denote the cost of FLOW to serve the request and the associated increase of the minimum matching value.

For a path P we write $\ell(P)$ for its length.

Let t_s be the location of the FLOW taxi matched to the adversary taxi at s in M. Let i be a random variable for the FLOW taxi serving s, so that t_i is its location before serving the request. We can partition the movement of taxi i along the path $P_{t_i s}$ into two parts, first the movement along $P_{t_i s} \setminus P_{t_s s}$ and then the movement along $P_{t_i s} \cap P_{t_s s}$ (Figure 3.2). During the first part, the value of M can increase by at most $\ell(P_{t_i s} \setminus P_{t_s s})$. Now, the value of M is at most that of the matching M' which differs from M in that the adversary taxi at s is matched to the FLOW taxi i, and the FLOW taxi at t_s is matched to the adversary taxi previously matched to i. As taxi i finishes its movement towards s, the value of M' decreases by precisely $\ell(P_{t_i s} \cap P_{t_s s})$. The value of the new minimum matching is bounded by the value of M'. Thus, we can bound the increase of the matching by

$$\Delta M \leq \ell(P_{t_i s} \setminus P_{t_s s}) - \ell(P_{t_i s} \cap P_{t_s s}). \tag{3.2}$$

On the right hand side, we simply count edges of $P_{t_i s}$ negatively if they are also part of $P_{t_s s}$, and positively otherwise.

For a (sub)network $A \in \mathcal{N}$ let

$$m(A) = \mathbb{E}(\ell(A \cap P_{t_i s} \setminus P_{t_s s}) - \ell(A \cap P_{t_i s} \cap P_{t_s s}) \mid t_i \in A)$$

be the expected contribution of edges from A to the matching bound (3.2), conditioned on Flow using a taxi that starts in A. Moreover, let

$$c(A) = \mathbb{E}(\ell(A \cap P_{t_i s}) \mid t_i \in A)$$

be the expected movement cost of Flow incurred on edges of A, conditioned on Flow using a taxi from A. For $A = P_{s_A s'_A} \cup B \cup C \in \mathcal{N}$ as above, it follows from the definition of the algorithm that

$$c(A) = d(s_A, s'_A) + \frac{R_C c(B)}{R_B + R_C} + \frac{R_B c(C)}{R_B + R_C}. \tag{3.3}$$

Since $\mathbb{E}(\mathit{cost}) = c(N)$ and $\mathbb{E}(\Delta M) \leq m(N)$, it suffices to show

$$c(N) + (2^k - 1)m(N) \leq 0. \tag{3.4}$$

A key insight is provided by the following claim, relating $m(A)$ to $c(A)$ and R_A, which will allow us to reformulate (3.4) purely in terms of the expected cost $c(N)$ and resistance R_N.

Claim 3.2.4. *For each $A \in \mathcal{N}$ with $t_s \in A$, $m(A) = c(A) - 2R_A$.*

Proof. The proof is by induction on $\kappa(A)$. If $\kappa(A) = 1$, then $A = P_{s_A t_s}$; the condition $t_i \in A$ in the expectations defining $m(A)$ and $c(A)$ is equivalent to $t_i = t_s$. Thus, we have $m(A) = -\ell(P_{s_A t_s})$ and $c(A) = \ell(P_{s_A t_s})$. Since $R_A = d(s_A, t_s) = \ell(P_{s_A t_s})$, the claim follows.

If $\kappa(A) \geq 2$, then A can be split into the path $P_{s_A s'_A}$ and subtrees $B, C \in \mathcal{N}$ as above. Conditional on $t_i \in A$, the path $P_{s_A s'_A}$ is contained in both $P_{t_i s}$ and $P_{t_s s}$, so it contributes negatively to $m(A)$. Moreover, conditional on $t_i \in A$, the remaining part of $A \cap P_{t_i s}$ is in B with probability $\frac{R_C}{R_B + R_C}$ and in C with probability $\frac{R_B}{R_B + R_C}$. Thus,

$$m(A) = -d(s_A, s'_A) + \frac{R_C m(B)}{R_B + R_C} + \frac{R_B m(C)}{R_B + R_C}.$$

Without loss of generality let B be the subtree containing t_s. Then we can replace $m(B)$ in this formula by applying the induction hypothesis. Moreover, the edges of $P_{t_s s}$ do not intersect with C, and therefore $m(C) = c(C)$. This gives us

$$m(A) = -d(s_A, s'_A) + \frac{R_C(c(B) - 2R_B)}{R_B + R_C} + \frac{R_B c(C)}{R_B + R_C}. \tag{3.5}$$

Combining (3.3) and (3.5), we get

$$c(A) - m(A) = 2d(s_A, s'_A) + 2\frac{R_B R_C}{R_B + R_C} = 2R_A$$

and the claim follows. □

Thanks to this claim, we can rewrite (3.4) as

$$2^{k-1}c(N) \leq (2^k - 1)R_N. \tag{3.6}$$

Observe that unlike (3.4), the reformulation (3.6) no longer depends on the adversary configuration and the matching.

To show (3.6), we will need the following property: For $A \in \mathcal{N}$,

$$\kappa(A)R_A \geq h(A), \tag{3.7}$$

where $h(A)$ denotes the minimal distance between s_A and a taxi in A. This follows by an easy induction on $\kappa(A)$.

We complete the proof by showing the following slightly stronger generalization of (3.6):

Claim 3.2.5. *For $A \in \mathcal{N}$ with $s \in A$, we have $2^{\kappa(A)-1}c(A) \leq (2^{\kappa(A)} - 1)R_A$.*

Proof. We proceed again by induction on $\kappa(A)$. For $\kappa(A) = 1$ the claim is easily seen to hold with equality.

If $\kappa(A) \geq 2$, let B and C be as before. Since $s \in A$, we have $s_A = s$.

From (3.1) and (3.3), we get

$$\begin{aligned} &2^{\kappa(A)-1}c(A) - (2^{\kappa(A)} - 1)R_A \\ &= (1 - 2^{\kappa(A)-1})d(s, s'_A) + 2^{\kappa(A)-1}\left(\frac{R_C c(B)}{R_B + R_C} + \frac{R_B c(C)}{R_B + R_C}\right) \\ &\qquad - (2^{\kappa(A)} - 1)\frac{R_B R_C}{R_B + R_C}. \end{aligned} \tag{3.8}$$

We will show that the right hand side is negative. Thus, for $\kappa(A) \geq 2$ the claim even holds with strict inequality.

The simpler case is that $P_{ss'_A}$ contains the parent node of s'_A in the HST. Then both B and C are contained in the subtree of the HST rooted at s'_A; hence, all paths from s'_A to any of the taxis in B or C have length exactly[5] $h(B \cup C)$, and therefore $c(B) = c(C) = h(B \cup C) < d(s, s'_A)$. Using this, as well as $\kappa(A) < 2^{\kappa(A)} - 1$ and applying (3.7) to $B \cup C \in \mathcal{N}$, we get that term (3.8) is less than

$$\begin{aligned}(1 - 2^{\kappa(A)-1})d(s, s'_A) + 2^{\kappa(A)-1}h(B \cup C) - \kappa(A)R_{B\cup C} \\ \leq (1 - 2^{\kappa(A)-1})d(s, s'_A) + (2^{\kappa(A)-1} - 1)h(B \cup C) \\ < 0.\end{aligned}$$

In the other case, when $P_{ss'_A}$ does not contain the parent of s'_A in the HST, we can still assume without loss of generality that also B does not contain the parent of s'_A. Then

$$c(B) = h(B) = d(s, s'_A), \tag{3.9}$$

where the equality with $d(s, s'_A)$ follows from the fact that the requested node s is also a leaf in the subtree of the HST rooted at s'_A.

Since $P_{ss'_A} \cup C \in \mathcal{N}$ and $\kappa(P_{ss'_A} \cup C) = \kappa(C) < \kappa(A)$, we can apply the induction hypothesis to $P_{ss'_A} \cup C$, yielding

$$2^{\kappa(C)-1}(d(s, s'_A) + c(C)) \leq (2^{\kappa(C)} - 1)(d(s, s'_A) + R_C),$$

and reordering,

$$c(C) \leq (1 - 2^{1-\kappa(C)})d(s, s'_A) + (2 - 2^{1-\kappa(C)})R_C. \tag{3.10}$$

[5] We are using here that an internal node of an HST is at the same distance from all its leaf descendants.

Due to (3.9), (3.10) and (3.7), we can bound term (3.8) by

$$\begin{aligned}
&(1-2^{\kappa(A)-1})h(B) + 2^{\kappa(A)-1}\frac{R_C h(B)}{R_B+R_C} + \\
&\quad (2^{\kappa(A)-1} - 2^{\kappa(A)-\kappa(C)})\frac{R_B h(B)}{R_B+R_C} + (1-2^{\kappa(A)-\kappa(C)})\frac{R_B R_C}{R_B+R_C} \\
&= h(B) - 2^{\kappa(B)}\frac{R_B h(B)}{R_B+R_C} - (2^{\kappa(B)}-1)\frac{R_B R_C}{R_B+R_C} \\
&< h(B) - \frac{R_B h(B)}{R_B+R_C} - \kappa(B)\frac{R_B R_C}{R_B+R_C} \\
&\leq h(B) - \frac{R_B h(B)}{R_B+R_C} - \frac{h(B)R_C}{R_B+R_C} \\
&= 0
\end{aligned}$$

and the claim follows. □

Invoking the claim for $A = N$, and using $\kappa(N) \leq k$, we obtain (3.6), concluding the proof of the theorem. □

3.2.2 Lower bound against adaptive adversaries

We now show the first lower bound of Theorem 3.2.1, matching the upper bound from the previous section.

Let B_k^α be the binary α-HST of depth k with vertex weights $\alpha^k, \alpha^{k-1}, \ldots, \alpha, 1$ along root-to-leaf paths. For an infinite request sequence σ, we denote by σ_t its prefix consisting of the first t requests. The lower bound for adaptive adversaries in Theorem 3.2.1 follows from the following theorem by letting $\alpha \to \infty$ and $T \to \infty$:

Theorem 3.2.6. *Let $\alpha \geq 3^k$. For each randomized algorithm A for the hard k-taxi problem on B_k^α, any fixed initial configuration, and any leaf ℓ of B_k^α, one can construct online an infinite request sequence σ such that*

(a) for each bounded stopping time T, there exists a deterministic online algorithm ADV (the adversary) such that

$$\mathbb{E}(cost_A(\sigma_T)) \geq \left(2^k - 1 - \frac{3^k}{\alpha}\right)\mathbb{E}(cost_{ADV}(\sigma_T)) - (2\alpha)^k,$$

(b) $cost_A(\sigma_t) \to \infty$ as $t \to \infty$ for all random choices of A, and

(c) *if the initial configuration is extended by adding a* $(k+1)$*st taxi at leaf* ℓ*, then* σ *can be served for free.*

Proof. We call an algorithm with an extra taxi as in (c) *augmented algorithm*.

We prove the theorem by induction on k. For $k = 1$, σ begins with a relocation request from the initial taxi position to the leaf of B_1^α other than ℓ and then places simple requests alternately at the two leaves of B_1^α. The adversary follows the unique strategy to serve the requests.

For the induction step, suppose the theorem holds for k and we want to show it for $k+1$. The infinite request sequence σ consists of several phases. Note that B_{k+1}^α contains two copies of B_k^α as subtrees. We call one of these two subtrees *active* and the other one *passive*, and these roles change after each phase. We also call a taxi *active* or *passive* if it is in the according subtree. All requests of a phase are in the active subtree, except for some relocation requests at the end of a phase. In phase 1, the subtree containing ℓ is active, and we let $\ell_1 = \ell$. We maintain the invariant that at the beginning of phase i, k taxis are active and one is passive, and we denote the leaf in the passive subtree where the latter is located by ℓ_{i+1}. As the active and passive subtree change their roles after each phase, ℓ_i is always in the active subtree of phase i. Clearly the invariant can be ensured for phase 1 by relocation requests in the beginning.

We define now the requests of phase i. As long as A does not activate its passive taxi (i.e. move it from ℓ_{i+1} to the active subtree), we can interpret the behaviour of A as that of an algorithm A_i for the k-taxi problem in the active subtree. To make A_i a full-fledged k-taxi algorithm for B_k^α, we define it arbitrarily from the point when A activates the passive taxi onwards. By the induction hypothesis, we can construct online a request sequence σ^i in the active subtree such that for any bounded stopping time T_i there is an adversary ADV_i with

$$\mathbb{E}(cost_{A_i}(\sigma^i_{T_i}) \mid \mathcal{F}_i) \geq \left(2^k - 1 - \frac{3^k}{\alpha}\right) \mathbb{E}(cost_{ADV_i}(\sigma^i_{T_i}) \mid \mathcal{F}_i) - (2\alpha)^k, \tag{3.11}$$

where $\mathcal{F}_i$ contains the information about A's decisions before phase i. Moreover, $cost_{A_i}(\sigma_t^i) \to \infty$ as $t \to \infty$, and an augmented algorithm with an extra taxi at ℓ_i serves σ^i for free.

Phase i commences with the requests of σ^i until A activates its passive taxi. If A never activates the passive taxi, then phase i never ends. Otherwise, once A activates the passive taxi, we use relocation requests to move k taxis from the active to the passive subtree (i.e. the active subtree of the next phase), which ends phase i and satisfies the aforementioned invariant for phase $i+1$. The k starting points of these relocation requests are the final taxi positions of ADV_i.

We need to show that this sequence satisfies the claimed properties.

We first show that the corresponding statement of (a) for $k+1$ instead of k holds. Let $n(T)$ be the number of phases of the request sequence σ_T. Note that all but possibly the last phase include an activation. For $i \le n(T)$, let T_i be the number of requests in phase i until A activates the passive taxi or (for $i = n(T)$) until time T is reached. For $i > n(T)$ let $T_i = 0$. Note that (3.11) holds even for $i > n(T)$ (with A_i and σ^i defined arbitrarily if σ has less than i phases) and we may multiply the term $(2\alpha)^k$ by $\mathbb{1}_{\{T_i>0\}} = \mathbb{1}_{\{i\le n(T)\}}$. Since activating the passive taxi is at least $\alpha^k(\alpha-1)$ more expensive than using an already active taxi, the cost of A in phase i is at least $cost_{A_i}(\sigma^i_{T_i}) + \mathbb{1}_{\{i<n(T)\}}\alpha^k(\alpha-1)$. Thus,

$$
\begin{aligned}
&\mathbb{E}(cost_A(\sigma_T)) \\
&\quad\ge \mathbb{E}\left(\sum_{i=1}^{\infty} cost_{A_i}(\sigma^i_{T_i}) + \mathbb{1}_{\{i<n(T)\}}\alpha^k(\alpha-1)\right) \\
&\quad= \left(\sum_{i=1}^{\infty} \mathbb{E}(cost_{A_i}(\sigma^i_{T_i}))\right) + (\mathbb{E}(n(T))-1)\alpha^k(\alpha-1) \\
&\quad\ge \left(\sum_{i=1}^{\infty} \left(2^k - 1 - \frac{3^k}{\alpha}\right)\mathbb{E}(cost_{ADV_i}(\sigma^i_{T_i})) - \mathbb{P}(i \le n(T))(2\alpha)^k\right) \\
&\qquad\qquad + \mathbb{E}(n(T))\alpha^k(\alpha-1) - \alpha^{k+1} \\
&\quad\ge \left(2^k - 1 - \frac{3^k}{\alpha}\right)\mathbb{E}\left(\sum_{i=1}^{n(T)} cost_{ADV_i}(\sigma^i_{T_i})\right) + \mathbb{E}(n(T))\alpha^k(\alpha-3^k) - \alpha^{k+1}. \qquad (3.12)
\end{aligned}
$$

The assumption that T is bounded guarantees that all sums have finitely many non-zero summands.

We will consider three different strategies for the adversary: The first strategy is to activate the passive offline taxi by moving it to ℓ_i at the start of each phase i. Since the other k adversary taxis cover the active online taxis at the start of the phase, this allows the adversary to serve the requests of the phase for free (thanks to (c)). Activating the passive taxi costs α^{k+1}, so this strategy incurs a cost of

$$n(T)\alpha^{k+1}. \tag{3.13}$$

In the second strategy we consider, the adversary never moves a taxi from one subtree B_k^α to the other, except when this happens due to a relocation request. In this case, phase i is free only for even i. If i is odd, it copies the behaviour of ADV_i to serve the requests of phase i. The cost of this strategy is

$$\sum_{\substack{i=1 \\ i \text{ odd}}}^{n(T)} cost_{ADV_i}(\sigma^i_{T_i}). \tag{3.14}$$

The third strategy is similar, but the offline algorithm moves the passive taxi from ℓ_2 to ℓ_1 before phase 1, for a cost of α^{k+1}. Analogously to the second strategy, this strategy incurs cost

$$\alpha^{k+1} + \sum_{\substack{i=2 \\ i \text{ even}}}^{n(T)} cost_{ADV_i}(\sigma^i_{T_i}). \tag{3.15}$$

The actual strategy ADV is the one of these three which has the smallest cost in expectation.

Since $\mathbb{E}(cost_{ADV}(\sigma_T))$ is bounded by the expectations of (3.14) and (3.15), we have

$$2\,\mathbb{E}(cost_{ADV}(\sigma_T)) \leq \alpha^{k+1} + \mathbb{E}\left(\sum_{i=1}^{n(T)} cost_{ADV_i}(\sigma^i_{T_i})\right).$$

Therefore, with (3.12) we get

$$\mathbb{E}(cost_A(\sigma_T)) \geq \left(2^{k+1} - 2 - \frac{2 \cdot 3^k}{\alpha}\right)\mathbb{E}(cost_{ADV}(\sigma_T)) + \mathbb{E}(n(T))\alpha^k(\alpha - 3^k) - (2\alpha)^{k+1}. \tag{3.16}$$

Since $\mathbb{E}(cost_{ADV}(\sigma_T))$ is bounded by the expectation of (3.13), for $\alpha \geq 3^k$ we can bound the middle term of (3.16) by

$$\begin{aligned}\mathbb{E}(n(T))\alpha^k(\alpha - 3^k) &\geq \frac{\mathbb{E}(n(T))\alpha^k(\alpha - 3^k)}{\mathbb{E}(n(T))\alpha^{k+1}}\,\mathbb{E}(cost_{ADV}(\sigma_T)) \\ &= \left(1 - \frac{3^k}{\alpha}\right)\mathbb{E}(cost_{ADV}(\sigma_T)).\end{aligned}$$

Therefore,

$$\mathbb{E}(cost_A(\sigma_T)) \geq \left(2^{k+1} - 1 - \frac{3^{k+1}}{\alpha}\right)\mathbb{E}(cost_{ADV}(\sigma_T)) - (2\alpha)^{k+1}.$$

The induction step of (b) is fairly straight-forward: If the number of phases $n(t) \to \infty$ as $t \to \infty$, then it follows from the fact that the cost increases by at least α^{k+1} per phase as this is the cost of activating the passive taxi. Otherwise, the length of the last phase goes to infinity, and so does A's cost during this phase by definition of the phases.

For (c), we show by induction on i that an augmented algorithm with extra taxi starting at ℓ can serve all requests before phase i for free, and at the start of phase i it ends up in the configuration of A with an extra taxi at ℓ_i. For $i = 1$ this is obvious by choice $\ell_1 = \ell$. Suppose now this holds for some i. Since the augmented algorithm has an extra taxi at ℓ_i at the beginning of phase i, this phase is free as well by choice of the request sequence. Thus, all requests before phase $i + 1$ are free. Moreover, since the requests of phase i are free, this means that all requests of phase i are relocation requests or simple requests at points where the augmented algorithm has a taxi at that time. But then it follows that the configuration of the augmented algorithm is always that of A with an extra taxi. Since A served the last simple request of phase i by moving a taxi away from ℓ_{i+1}, the position of the extra taxi is ℓ_{i+1} when phase $i + 1$ starts. □

3.2.3 Lower bound for memoryless algorithms

We now show the other lower bound of Theorem 3.2.1. For a configuration C and a request sequence σ, we denote by $w(C, \sigma)$ the optimal cost of a schedule that, starting from configuration C, serves σ and then returns to configuration C. We will need the following lemma.

Lemma 3.2.7. *Let $C'_0, \ldots, C'_n$ be a $(k+1)$-taxi schedule of cost 0 for a request sequence σ.*

(a) If $C_0, \ldots, C_n$ is a k-taxi schedule for σ with $C_0 \subset C'_0$, then $C_i \subset C'_i$ for all i.

(b) If $C_0, \ldots, C_n$ is a $(k+1)$-taxi schedule for σ with $C_0 \setminus C'_0 = \{\ell\}$ for some $\ell \notin C_n$, then $C_n = C'_n$.

Proof. To prove (a), we proceed by induction on i. If the k-taxi algorithm incurs no cost when passing from C_{i-1} to C_i to serve the ith request r_i, then either $C_{i-1} = C_i$ and r_i is a simple request at a point of C_i, or r_i is a relocation request (s_i, t_i) with $s_i \in C_{i-1}$ and $C_i = C_{i-1} \setminus \{s_i\} \cup \{t_i\}$. In both cases, $C_i \subset C'_i$ follows from $C_{i-1} \subset C'_{i-1}$.

Otherwise, without loss of generality r_i is a simple request at some $t_i \notin C_{i-1}$, and $C_i = C_{i-1} \setminus \{x_i\} \cup \{t_i\}$ for some $x_i \in C_{i-1}$. Since the schedule $C'_0, \ldots, C'_n$ has cost 0, we must have $C'_{i-1} = C'_i = C_{i-1} \cup \{t_i\} = C_i \cup \{x_i\}$.

Part (b) follows from part (a) if we replace C'_i by $C'_i \cup \{\ell\}$ and k by $k+1$. □

The lower bound of Theorem 3.2.1 for memoryless algorithms against oblivious adversaries follows from the following theorem by letting $N \to \infty$. We use again the binary α-HSTs B_k^α from the previous section.

Theorem 3.2.8. *For $N \in \mathbb{N}$ sufficiently large, $\alpha = N^2$, each memoryless algorithm A_k for the hard k-taxi problem on B_k^α and any leaf $\ell \in B_k^\alpha$, there exists a configuration C of k distinct points and a request sequence σ such that*

(a) $\mathbb{E}(cost_{A_k}(C; \sigma)) \geq (2^k - 1)(2\alpha)^k(N - 2^k)$,

(b) $0 < w(C, \sigma) \leq (2\alpha)^k(N + k)$,

(c) $w(C \cup \{\ell\}, \sigma) = 0$.

Proof. Let us first fix some notation. For request sequences σ_1 and σ_2, we denote by $\sigma_1\sigma_2$ their concatenation. For $m \geq 1$, let σ_1^m be the m-fold repetition of σ_1, i.e., σ_1^0 is the empty sequence and $\sigma_1^{m+1} = \sigma_1^m\sigma_1$. For sets of points $X = \{x_1, \ldots, x_n\}$ and

$Y = \{y_1, \ldots, y_n\}$ (with, say, $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ sorted according to some total order on the metric space), we denote by $(X \to Y)$ the sequence of relocation requests $(x_1, y_1) \ldots (x_n, y_n)$. Thus, if an algorithm is in configuration X, then the request sequence $(X \to Y)$ changes the configuration to Y. Abusing notation, we also write X for the sequence of simple requests $x_1 \ldots x_n$. So if X is a configuration of distinct points, then the request sequence X^m forces an algorithm to either move to the configuration X or suffer large cost (if m is large).

We prove the theorem by induction on k. For $k = 1$, we can write $B_1^\alpha = \{\ell, r\}$ for some r. The theorem holds for $C = \{r\}$ and $\sigma = (\ell r)^N$.

For the induction step, suppose the theorem holds for some fixed k and we want to prove it for $k + 1$. Say we are given an algorithm A_{k+1} for the k-taxi problem and a leaf $\ell \in B_{k+1}^\alpha$. We will refer to the subtree B_k^α containing ℓ as the *left subtree* and the other subtree B_k^α as the *right subtree*. For a leaf r in the right subtree, write $A_{k+1}|_r$ for the k-taxi algorithm in the left subtree that behaves like A_{k+1} conditioned on having one taxi at r that does not move.[6] Let $C_{r\ell}$ and $\sigma_{r\ell}$ be the k-taxi configuration and request sequence in the left subtree induced by the induction hypothesis applied to $A_{k+1}|_r$ and ℓ. For a request sequence σ', let $F_r(\sigma')$ denote the final configuration of the schedule $A_{k+1}(C_{r\ell} \cup \{r\}; \sigma')$.

In the following, m will be some large integer. Let

$$p_{r\ell,m} = \mathbb{P}(r \notin F_r(\sigma_{r\ell} C_{r\ell}^m))$$
$$\epsilon_{r\ell,m} = \mathbb{P}(C_{r\ell} \not\subset F_r(\sigma_{r\ell} C_{r\ell}^m)).$$

Note that $p_{r\ell,m}$ is a non-decreasing and $\epsilon_{r\ell,m}$ a non-increasing function of m. Thus, we can define

$$p_{r\ell} = \lim_{m\to\infty} p_{r\ell,m}$$
$$\epsilon_{r\ell} = \lim_{m\to\infty} \epsilon_{r\ell,m}.$$

Define $C_{\ell r}$, $\sigma_{\ell r}$, $p_{\ell r,m}$, $\epsilon_{\ell r,m}$, $p_{\ell r}$, $\epsilon_{\ell r}$ and F_ℓ similarly with the roles of ℓ and r reversed.

[6] If for some configuration $C \cup \{r\}$, A_{k+1} moves the taxi from r with probability 1 for a given request, the move of $A_{k+1}|_r$ from C is defined arbitrarily.

Roughly speaking, the values $p_{r\ell}$ and $p_{\ell r}$ indicate how aggressively the algorithm moves between the two subtrees. We use two different definitions for σ depending on whether these values are large or small. In both cases, we set $C = C_{r\ell} \cup \{r\}$.

Case 1: $p_{\ell r} + p_{r\ell} > \frac{2^k}{N}$ and $p_{r\ell} > 0$ and $p_{\ell r} > 0$.
The central building blocks of σ are the subsequences

$$\sigma_r = \sigma_{r\ell} C_{r\ell}^m (C_{r\ell} \to C_{\ell r})(\sigma_{\ell r} C_{\ell r}^m)^m (C_{\ell r} \to C_{r\ell})$$

and σ_ℓ defined in the same way with ℓ and r reversed. The idea of σ_r is that when starting from configuration $C_{r\ell} \cup \{r\}$, the prefix $\sigma_{r\ell} C_{r\ell}^m$ shall lure the algorithm to move the taxi from r to the left subtree. If it does so, it will be punished during the part $(\sigma_{\ell r} C_{\ell r}^m)^m$, which forces all taxis back to the right subtree. Since $p_{\ell r} + p_{r\ell} > \frac{2^k}{N}$, at least one of σ_r and σ_ℓ will successfully exploit the algorithm's aggressiveness. We define the entire request sequence as

$$\sigma = \sigma_r^\alpha (C_{r\ell} \to C_{\ell r})(C_{\ell r} \cup \{\ell\})^m \sigma_\ell^\alpha (C_{\ell r} \to C_{r\ell}).$$

We first prove the induction step of (b). Clearly, $0 < w(C, \sigma)$. Moreover, by the induction hypothesis, we have $w(C_{r\ell} \cup \{r\}, \sigma_{r\ell}) \le w(C_{r\ell}, \sigma_{r\ell}) \le (2\alpha)^k (N + k)$ and $w(C_{\ell r} \cup \{r\}, \sigma_{\ell r}) = 0$. Therefore, $w(C_{r\ell} \cup \{r\}, \sigma_r) \le (2\alpha)^k (N + k)$. By symmetry, we also have $w(C_{\ell r} \cup \{\ell\}, \sigma_\ell) \le (2\alpha)^k (N + k)$. Thus,

$$\begin{aligned} w(C, \sigma) &\le \alpha w(C_{r\ell} \cup \{r\}, \sigma_r) + \alpha^{k+1} + \alpha w(C_{\ell r} \cup \{\ell\}, \sigma_\ell) + \alpha^{k+1} \\ &\le (2\alpha)^{k+1} (N + k + 1). \end{aligned}$$

To see (c), it follows easily from the induction hypothesis that, starting from $C \cup \{\ell\} = C_{r\ell} \cup \{\ell, r\}$, serving σ incurs no cost and makes the algorithm return to $C \cup \{\ell\}$ in the end.

The most technical part of this proof is the induction step of (a). We can assume that

$$\limsup_{m \to \infty} \mathbb{E}(cost_{A_{k+1}}(C; \sigma)) < \infty, \tag{3.17}$$

since otherwise (a) follows immediately for some large choice of m.

Let us first examine the behaviour of the schedule $A_{k+1}(C_{r\ell} \cup \{r\}; \sigma_{r\ell} C_{r\ell}^m)$. With probability $p_{r\ell,m}$, the algorithm moves the taxi from r to the left subtree for cost α^{k+1}. Otherwise (with probability $1 - p_{r\ell,m}$), the taxi at r stays put; conditioned on this, the expected cost suffered during the prefix $\sigma_{r\ell}$ is at least $(2^k - 1)(2\alpha)^k(N - 2^k)$ by the induction hypothesis. Moreover, if $F_r(\sigma_{r\ell} C_{r\ell}^m) \not\supset C_{r\ell}$, then $F_r(\sigma_{r\ell}\sigma') \not\supset C_{r\ell}$ for any prefix σ of $C_{r\ell}^m$ and the algorithm incurs cost at least α during each of the m subsequences $C_{r\ell}$. Overall, we have

$$\begin{aligned}&\mathbb{E}(cost_{A_{k+1}}(C_{r\ell} \cup \{r\}; \sigma_{r\ell} C_{r\ell}^m)) \\ &\qquad \geq p_{r\ell,m}\alpha^{k+1} + (1 - p_{r\ell,m})(2^k - 1)(2\alpha)^k(N - 2^k) + \epsilon_{r\ell,m}\alpha m. \end{aligned} \tag{3.18}$$

Claim 3.2.9. *$r \notin F_r(\sigma_{r\ell} C_{r\ell}^m)$ if and only if $F_r(\sigma_{r\ell} C_{r\ell}^m) = C_{r\ell} \cup \{\ell\}$.*

Proof. The direction "if" holds since $C_{r\ell} \cup \{\ell\}$ is contained in the left subtree while r is in the right subtree. "Only if" follows from part (c) of the induction hypothesis and Lemma 3.2.7(b). □

From the claim it also follows that the two events defining $\epsilon_{r\ell,m}$ and $p_{r\ell,m}$ are disjoint. So with probability $1 - p_{r\ell,m} - \epsilon_{r\ell,m}$, none of the two events happens. This implies

$$\mathbb{P}(F_r(\sigma_{r\ell} C_{r\ell}^m) = C_{r\ell} \cup \{\ell\}) = p_{r\ell,m} \tag{3.19}$$

$$\mathbb{P}(F_r(\sigma_{r\ell} C_{r\ell}^m) = C_{r\ell} \cup \{r\}) = 1 - p_{r\ell,m} - \epsilon_{r\ell,m}. \tag{3.20}$$

For $i \geq 0$, we have

$$\begin{aligned}&\mathbb{P}(F_r(\sigma_{r\ell} C_{r\ell}^m (C_{r\ell} \to C_{\ell r})(\sigma_{\ell r} C_{\ell r}^m)^i) = C_{\ell r} \cup \{\ell\}) \\ &\geq \mathbb{P}(F_r(\sigma_{r\ell} C_{r\ell}^m) = C_{r\ell} \cup \{\ell\} \wedge \\ &\qquad\qquad \forall j = 1, \ldots, i \colon F_r(\sigma_{r\ell} C_{r\ell}^m (C_{r\ell} \to C_{\ell r})(\sigma_{\ell r} C_{\ell r}^m)^j) = C_{\ell r} \cup \{\ell\}) \\ &= \mathbb{P}(F_r(\sigma_{r\ell} C_{r\ell}^m) = C_{r\ell} \cup \{\ell\})\, \mathbb{P}(F_\ell(\sigma_{\ell r} C_{\ell r}^m) = C_{\ell r} \cup \{\ell\})^i \\ &= p_{r\ell,m}(1 - p_{\ell r,m} - \epsilon_{\ell r,m})^i, \end{aligned} \tag{3.21}$$

where the first equation uses memorylessness of A_{k+1} and the last equation uses (3.19) and the symmetric version of (3.20).

Using (3.21), (3.18) and the symmetric version of (3.18), we can bound the cost during σ_r by

$$\begin{aligned}
&\mathbb{E}(cost_{A_{k+1}}(C_{r\ell} \cup \{r\}; \sigma_r)) \\
&\geq \mathbb{E}(cost_{A_{k+1}}(C_{r\ell} \cup \{r\}; \sigma_{r\ell} C_{r\ell}^m) \\
&\quad + \sum_{i=0}^{m-1} p_{r\ell,m}(1 - p_{\ell r,m} - \epsilon_{\ell r,m})^i \, \mathbb{E}(cost_{A_{k+1}}(C_{\ell r} \cup \{\ell\}; \sigma_{\ell r} C_{\ell r}^m) \\
&\geq p_{r\ell,m}\alpha^{k+1} + (1 - p_{r\ell,m})(2^k - 1)(2\alpha)^k(N - 2^k) + \epsilon_{r\ell,m}\alpha m \\
&\quad + p_{r\ell,m} \frac{1 - (1 - p_{\ell r,m} - \epsilon_{\ell r,m})^m}{p_{\ell r,m} + \epsilon_{\ell r,m}} \Big(p_{\ell r,m}\alpha^{k+1} + \\
&\qquad\qquad (1 - p_{\ell r,m})(2^k - 1)(2\alpha)^k(N - 2^k) + \epsilon_{\ell r,m}\alpha m \Big).
\end{aligned}$$

Due to (3.17), it follows that $\epsilon_{\ell r} = \epsilon_{r\ell} = 0$. Thus, letting $m \to \infty$ and using that $p_{\ell r} > 0$, we get

$$\begin{aligned}
&\limsup_{m\to\infty} \mathbb{E}(cost_{A_{k+1}}(C_{r\ell} \cup \{r\}; \sigma_r) \\
&\geq p_{r\ell}\alpha^{k+1} + (1 - p_{r\ell})(2^k - 1)(2\alpha)^k(N - 2^k) \\
&\quad + \frac{p_{r\ell}}{p_{\ell r}} \left(p_{\ell r}\alpha^{k+1} + (1 - p_{\ell r})(2^k - 1)(2\alpha)^k(N - 2^k) \right) \\
&= 2p_{r\ell}\alpha^{k+1} + \left(1 - 2p_{r\ell} + \frac{p_{r\ell}}{p_{\ell r}} \right) (2^k - 1)(2\alpha)^k(N - 2^k). \qquad (3.22)
\end{aligned}$$

The next claim says that with arbitrarily high probability, the algorithm returns to its initial configuration after each subsequence σ_r.

Claim 3.2.10. *For all $i \leq \alpha$, $\mathbb{P}(F_r(\sigma_r^i) = C_{r\ell} \cup \{r\}) \to 1$ as $m \to \infty$.*

Proof. Since

$$\begin{aligned}
\mathbb{P}(F_r(\sigma_r^i) = C_{r\ell} \cup \{r\}) &\geq \mathbb{P}(\forall j = 1, \ldots, i \colon F_r(\sigma_r^j) = C_{r\ell} \cup \{r\}) \\
&= \mathbb{P}(F_r(\sigma_r) = C_{r\ell} \cup \{r\})^i,
\end{aligned}$$

we only need to show the claim for $i = 1$. Thanks to (3.19), (3.20) and $\epsilon_{r\ell} = 0$, it suffices to show

$$\mathbb{P}(F_r(\sigma_r) = C_{r\ell} \cup \{r\} \mid F_r(\sigma_{r\ell} C_{r\ell}^m) = C_{r\ell} \cup \{\ell\}) \to 1 \text{ as } m \to \infty \tag{3.23}$$

$$\mathbb{P}(F_r(\sigma_r) = C_{r\ell} \cup \{r\} \mid F_r(\sigma_{r\ell} C_{r\ell}^m) = C_{r\ell} \cup \{r\}) = 1. \tag{3.24}$$

We first show (3.24). When arriving in $C_{r\ell} \cup \{r\}$ after $\sigma_{r\ell} C_{r\ell}^m$, then the relocation sequence $(C_{r\ell} \to C_{\ell r})$ changes the configuration to $C_{\ell r} \cup \{r\}$. Thanks to part (c) of the induction hypothesis, the following $\sigma_{\ell r}$ is then served for free and A_{k+1} returns to configuration $C_{\ell r} \cup \{r\}$ at the end of it. The subsequent subsequence $C_{\ell r}^m$ does not cause any movement. Thus, at the end of the whole subsequence $(\sigma_{\ell r} C_{\ell r}^m)^m$, the configuration is $C_{\ell r} \cup \{r\}$. The last set of relocation requests changes the configuration to $C_{r\ell} \cup \{r\}$.

For (3.23), if the configuration is $C_{r\ell} \cup \{\ell\}$ after $\sigma_{r\ell} C_{r\ell}^m$, then the relocation sequence $(C_{r\ell} \to C_{\ell r})$ changes it to $C_{\ell r} \cup \{\ell\}$. Thus,

$$\begin{aligned}
&\mathbb{P}(F_r(\sigma_r) = C_{r\ell} \cup \{r\} \mid F_r(\sigma_{r\ell} C_{r\ell}^m) = C_{r\ell} \cup \{\ell\}) \\
&= \mathbb{P}(F_\ell((\sigma_{\ell r} C_{\ell r}^m)^m (C_{\ell r} \to C_{r\ell})) = C_{r\ell} \cup \{r\}) \\
&\geq \mathbb{P}(F_\ell((\sigma_{\ell r} C_{\ell r}^m)^m) = C_{\ell r} \cup \{r\}) \\
&= 1 - \mathbb{P}(\forall i \leq m \colon F_\ell((\sigma_{\ell r} C_{\ell r}^m)^i) \neq C_{\ell r} \cup \{r\}) \\
&= 1 - \mathbb{P}(\forall i \leq m \colon F_\ell((\sigma_{\ell r} C_{\ell r}^m)^i) = C_{\ell r} \cup \{\ell\}) \\
&\qquad - \mathbb{P}(\exists i \leq m \colon F_\ell((\sigma_{\ell r} C_{\ell r}^m)^i) \notin \{C_{\ell r} \cup \{\ell\}, C_{\ell r} \cup \{r\}\}) \\
&= 1 - (1 - p_{\ell r,m} - \epsilon_{\ell r,m})^m - \mathbb{P}(\exists i \leq m \colon C_{\ell r} \notin F_\ell((\sigma_{\ell r} C_{\ell r}^m)^i)),
\end{aligned}$$

where the second equation uses part (c) of the induction hypothesis in the same way as it was used in the proof of (3.24). Due to our assumptions $p_{\ell r} > 0$ and (3.17), both subtrahends in the last term tend to 0 as $m \to \infty$. □

Let $q_m = \mathbb{P}\left(F_r(\sigma_r^\alpha (C_{r\ell} \to C_{\ell r})(C_{\ell r} \cup \{\ell\})^m) = C_{\ell r} \cup \{\ell\}\right)$. This is the probability of successfully forcing configuration $C_{\ell r} \cup \{\ell\}$, bringing A_{k+1} in the symmetric

situation of the initial configuration, before the "second half" of σ. Again by (3.17), we have $q_m \to 1$ as $m \to \infty$. We are now ready to put the parts together:

$$\begin{aligned}
&\mathbb{E}(cost_{A_{k+1}}(C;\sigma)) \\
&\geq \sum_{i=0}^{\alpha-1} \mathbb{P}(F_r(\sigma_r^i) = C_{r\ell} \cup \{r\})\, \mathbb{E}(cost_{A_{k+1}}(C_{r\ell} \cup \{r\}; \sigma_r)) \\
&\quad + q_m \sum_{i=0}^{\alpha-1} \mathbb{P}(F_\ell(\sigma_\ell^i) = C_{\ell r} \cup \{\ell\})\, \mathbb{E}(cost_{A_{k+1}}(C_{\ell r} \cup \{\ell\}; \sigma_\ell))
\end{aligned}$$

and therefore

$$\begin{aligned}
&\limsup_{m\to\infty} \mathbb{E}(cost_{A_{k+1}}(C;\sigma)) \\
&\geq \alpha \limsup_{m\to\infty} \left[\mathbb{E}(cost_{A_{k+1}}(C_{r\ell} \cup \{r\}; \sigma_r)) + \mathbb{E}(cost_{A_{k+1}}(C_{\ell r} \cup \{\ell\}; \sigma_\ell))\right] \\
&\geq \alpha \left[2p_{r\ell}\alpha^{k+1} + \left(1 - 2p_{r\ell} + \frac{p_{r\ell}}{p_{\ell r}}\right)(2^k - 1)(2\alpha)^k(N - 2^k)\right. \\
&\qquad \left. + 2p_{\ell r}\alpha^{k+1} + \left(1 - 2p_{\ell r} + \frac{p_{\ell r}}{p_{r\ell}}\right)(2^k - 1)(2\alpha)^k(N - 2^k)\right] \\
&\geq 2\alpha \left[(p_{r\ell} + p_{\ell r})\alpha^{k+1} + \left(2 - (p_{r\ell} + p_{\ell r})\right)(2^k - 1)(2\alpha)^k(N - 2^k)\right] \\
&\geq (2\alpha)^{k+1} \left[N - 2^k(2^k - 1)(1 - \frac{2^k}{N}) + (2^{k+1} - 2)(N - 2^k)\right] \\
&> (2\alpha)^{k+1}(2^{k+1} - 1)(N - 2^{k+1}),
\end{aligned}$$

where the first inequality uses $q_m \to 1$ and Claim 3.2.10 and its symmetric case, the second inequality uses 3.22 and its symmetric case, the third inequality uses that $\frac{x}{y} + \frac{y}{x} \geq 2$ for $x, y > 0$, and the fourth inequality holds for large enough $\alpha = N^2$ and uses $p_{r\ell} + p_{\ell r} > \frac{2^k}{N}$. Since the last inequality is strict, the induction step follows for large enough m.

Case 2: $p_{\ell r} + p_{r\ell} \leq \frac{2^k}{N}$ or $p_{r\ell} = 0$ or $p_{\ell r} = 0$.

In this case, we exploit the reluctance of $A_{k+1}(C_{r\ell} \cup \{r\}; \sigma_{r\ell} C_{r\ell}^m)$ to move the taxi from r to the left subtree (or likewise with r and ℓ reversed) by requesting $\sigma_{r\ell} C_{r\ell}^m$ many times in a row. Eventually, the algorithm will have to bring the taxi from r

or it will suffer unbounded cost. Concretely, we define

$$\sigma_\ell = (\sigma_{r\ell} C_{r\ell}^m)^m (C_{r\ell} \to C_{\ell r})$$
$$\sigma_r = (\sigma_{\ell r} C_{\ell r}^m)^m (C_{\ell r} \to C_{r\ell})$$
$$\sigma = (\sigma_\ell \sigma_r)^{2^k N}.$$

We begin with the induction step of (b). From initial configuration $C = C_{r\ell} \cup \{r\}$, the offline algorithm can move the taxi from r to ℓ for cost α^{k+1} at the start. Now, from configuration $C_{r\ell} \cup \{\ell\}$, the sequence $(\sigma_{r\ell} C_{r\ell}^m)^m (C_{r\ell} \to C_{\ell r})$ is served for free thanks to part (c) of the induction hypothesis, leading to configuration $C_{\ell r} \cup \{\ell\}$. Then, the taxi from ℓ moves back to r for another cost α^{k+1}, so that no more cost is incurred during $(\sigma_{\ell r} C_{\ell r}^m)^m (C_{\ell r} \to C_{r\ell})$, which makes the algorithm returns to C. Repeating this $2^k N$ times, we get $w(C, \sigma) \leq 2^k N(\alpha^{k+1} + \alpha^{k+1}) \leq (2\alpha)^{k+1}(N + k)$, as desired.

The proof of $0 < w(C, \sigma)$ and (c) is again straightforward.

For part (a), if $p_{r\ell} = 0$ or $p_{\ell r} = 0$, then the cost is unbounded as $m \to \infty$ during the first subsequence $\sigma_\ell \sigma_r$ already. So we can assume $p_{r\ell} > 0$ and $p_{\ell r} > 0$. The same techniques as in the aggressive case yield

$$\begin{aligned}
&\mathbb{E}(cost_{A_{k+1}}(C_{r\ell} \cup \{r\}; \sigma_\ell)) \\
&\quad \geq \sum_{i=0}^{m-1} (1 - p_{r\ell,m} - \epsilon_{r\ell,m})^i \, \mathbb{E}(cost_{A_{k+1}}(C_{r\ell} \cup \{r\}; \sigma_{r\ell} C_{r\ell}^m)) \\
&\quad \geq \frac{1 - (1 - p_{r\ell,m} - \epsilon_{r\ell,m})^m}{p_{r\ell,m} + \epsilon_{r\ell,m}} \Big(p_{r\ell,m} \alpha^{k+1} + (1 - p_{r\ell,m})(2^k - 1)(2\alpha)^k (N - 2^k) \\
&\qquad\qquad + \epsilon_{r\ell,m} \alpha m \Big)
\end{aligned}$$

and, with assumption (3.17),

$$\mathbb{P}(F_r(\sigma_\ell) = C_{\ell r} \cup \{\ell\}) \to 1 \text{ as } m \to \infty.$$

Together with the symmetric equivalents of these statements, this gives

$$\begin{aligned}
&\limsup_{m\to\infty} \mathbb{E}(cost_{A_{k+1}}(C;\sigma)) \\
&\quad \geq 2^k N \left[2\alpha^{k+1} + \left(\frac{1}{p_{r\ell}} + \frac{1}{p_{\ell r}} - 2 \right) (2^k - 1)(2\alpha)^k (N - 2^k) \right] \\
&\quad \geq (2\alpha)^{k+1}(N - (2^k - 1)2^k) + 2^k N \frac{4}{p_{\ell r} + p_{r\ell}} (2^k - 1)(2\alpha)^k (N - 2^k) \\
&\quad \geq (2\alpha)^{k+1} \left(N - (2^k - 1)2^k + (2^{k+1} - 2)(N - 2^k) \right) \\
&\quad > (2\alpha)^{k+1}(2^{k+1} - 1)(N - 2^{k+1}),
\end{aligned}$$

where the second inequality uses $\frac{1}{x} + \frac{1}{y} \geq \frac{4}{x+y}$ for $x, y > 0$ and the third inequality holds for large enough N and uses $p_{\ell r} + p_{r\ell} \leq \frac{2^k}{N}$. This completes the proof. □

3.2.4 Reduction from layered graph traversal

The *layered width-k graph traversal problem (k-LGT)* is defined as follows: A searcher starts at a node s of a graph with non-negative edge weights and whose nodes can be partitioned into layers $L_0 = \{s\}, L_1, L_2, \ldots$ such that all edges run between consecutive layers. Each layer contains at most k nodes. The goal is to move the searcher along the edges to some vertex t while minimizing the distance traveled by the searcher. However, the nodes in L_ℓ and the edges between $L_{\ell-1}$ and L_ℓ are only revealed when the searcher reaches a node in $L_{\ell-1}$.

The following reduction yields an interesting connection between k-LGT (and the equivalent problem of chasing sets of cardinality k [FFK+98]) and the k-taxi problem.

Theorem 3.2.11. *If there exists a ρ-competitive deterministic algorithm for the hard k-taxi problem, then there exists a ρ-competitive deterministic algorithm for k-LGT.*

Proof. Fiat et al. [FFK+98] showed that k-LGT has the same competitive ratio as its restricted case where the graph is a tree and all edges have weight 0 or 1. Let s_ℓ be the first node visited by the online algorithm in the ℓth layer; in particular, s_0 is the starting position of the searcher. We can assume that any node $v \in L_\ell \setminus \{s_\ell\}$ has at most one adjacent node in layer $\ell + 1$, and the connecting edge would be of

weight 0. This is because any other edges leaving v can be delayed to a later layer, once the searcher moves to that branch. We design a ρ-competitive algorithm for the traversal of this type of 0-1-weighted trees. The movement of the searcher is determined by the decisions of a ρ-competitive k-taxi algorithm.

Let T be the layered tree of width at most k for the traversal problem with the aforementioned properties. As metric space for the k-taxi algorithm we use an infinite tree where each node has infinitely many children and each edge has weight 1. Note that T can be isometrically embedded into this infinite tree by contracting any nodes connected by an edge of weight 0 to a single node. Moreover, such an embedding can be constructed online while T is being revealed. We will only make taxi requests at nodes corresponding to the revealed part of T, so we can pretend that the requests and taxis are located on T itself.

We will maintain the following invariant: Right after a layer L_ℓ gets revealed because the searcher moved to $s_{\ell-1}$, the configuration C_ℓ of the k taxis is a multiset over L_ℓ where each node of L_ℓ has multiplicity at least 1. Initially this situation can be achieved by relocation requests. Then, a simple request at $s_{\ell-1}$ is issued. The k-taxi algorithm will serve the request by moving from some $x \in C_\ell$ to $s_{\ell-1}$. We will move the same distance in the traversal problem by moving the searcher from $s_{\ell-1}$ to x, which reveals the $(\ell+1)$st layer and sets $s_\ell := x$. Some of the taxis may be able to move to layer $\ell+1$ for free along edges of weight 0. By making relocation requests it will be ensured that all taxis occupy all of the at most k nodes in layer $\ell+1$, so the invariant holds again. This defines a procedure for traversing the tree.

Note that the online cost for traversing the tree is the same as the online cost for the k-taxi problem. It only remains to show that the optimal cost for the traversal problem is at least the optimal cost for the k-taxi problem.

Let σ_ℓ be the request sequence up to the point where the online taxis are in configuration C_ℓ. For $y \in C_\ell$ we denote by $C_\ell - y + s_{\ell-1}$ the configuration obtained from C_ℓ by replacing one copy of y by $s_{\ell-1}$. Let $w_\ell(C_\ell - y + s_{\ell-1})$ be the optimal cost to serve σ_ℓ and subsequently end up in configuration $C_\ell - y + s_{\ell-1}$. We claim

that $w_\ell(C_\ell - y + s_{\ell-1}) \leq d(s_0, y)$ for all $y \in C_\ell$, where d denotes the distance function on T. We prove this by induction on ℓ.

After the initial relocation requests, the offline configuration is C_1 and configuration $C_1 - y + s_0$ can be reached for cost $d(s_0, y)$ by moving a taxi from y to s_0. Thus, the claim holds for $\ell = 1$.

Suppose the claim holds for some ℓ. The next requests after σ_ℓ are a simple request at $s_{\ell-1}$ (which changes the online configuration from C_ℓ to $C_\ell - s_\ell + s_{\ell-1}$) and some relocation requests that, together with moves along weight-0 edges, change the configuration to $C_{\ell+1}$. Let $y' \in C_{\ell+1}$ and let $y \in C_\ell$ be its parent in layer ℓ. One (offline) way to serve $\sigma_{\ell+1}$ and end up in configuration $C_{\ell+1} - y' + s_\ell$ is as follows: First serve σ_ℓ and reach configuration $C_\ell - y + s_{\ell-1}$ for cost $w_\ell(C_\ell - y + s_{\ell-1})$. The simple request at $s_{\ell-1}$ is then served for free without moving. Recall that the following relocation requests and moves along weight-0 edges change the online configuration from $C_\ell - s_\ell + s_{\ell-1}$ to $C_{\ell+1}$. If the edge (y, y') has weight 0, then the same relocations and moves along weight-0 edges, except for the move from y to y', change the offline configuration from $C_\ell - y + s_{\ell-1}$ to $C_{\ell+1} - y' + s_\ell$. So in this case,

$$w_{\ell+1}(C_{\ell+1} - y' + s_\ell) \leq w_\ell(C_\ell - y + s_{\ell-1}) \leq d(s_0, y) = d(s_0, y')$$

as claimed. In the other case, $y = s_\ell$ by assumption on the layered tree, so before the relocation requests, the offline configuration is the same as the online configuration. Thus, online and offline configuration are the same also after the relocation moves, which is configuration $C_{\ell+1}$. Finally, $C_{\ell+1} - y' + s_\ell$ can be reached by moving a taxi from y' to $y = s_\ell$. Thus,

$$w_{\ell+1}(C_{\ell+1} - y' + s_\ell) \leq w_\ell(C_\ell - y + s_{\ell-1}) + d(y, y') \leq d(s_0, y) + d(y, y') = d(s_0, y').$$

From the claim it follows that the optimal cost for the taxi request sequence is at most the length of the path from s_0 to any node y in the last layer. If y is the target vertex of the traversal problem, the latter is precisely the offline cost for the traversal problem. □

The theorem easily extends to randomized algorithms against adaptive adversaries. However, we do not know whether it also holds for randomized algorithms against oblivious adversaries.

3.2.5 Tight bounds for two taxis

Let us consider now the special case $k = 2$ of the hard k-taxi problem. The goal of this section is to give tight bounds of 9 for deterministic algorithms in general metrics.

Theorem 3.2.12. *The deterministic competitive ratio of the hard 2-taxi problem is exactly 9.*

The lower bound follows from the reduction from k-LGT from the previous section and the fact that the deterministic competitive ratio of 2-LGT is exactly 9 [PY91].

For the upper bound, we present an algorithm BiasedDC for the hard 2-taxi problem on general metrics.

Note that there is always a pair of an online algorithm taxi and an offline algorithm taxi occupying the same location, namely the taxis that served the last request (or, in the initial configuration, any online taxi and the corresponding offline taxi starting at the same point). We call these taxis *active* and denote them by A (online) and a (offline). The other two taxis are *passive*, denoted by P (online) and p (offline).

BiasedDC is a speed-adjusted variant of the Double Coverage algorithm. Upon a simple request at s, BiasedDC moves both taxis towards s, but P moves at twice the speed of A. As soon as either taxi reaches s, both taxis stop moving.

This definition assumes that all points along shortest paths from the old taxi positions to s belong to the metric space, which does not have to be true in general. However, we can assume that this is the case by adding virtual points to the metric space: If a taxi moves from its old position ℓ towards the request s but stops after a fraction q of the movement, we augment the metric space by adding a new point at distance $qd(\ell, s)$ from ℓ and $(1-q)d(\ell, s)$ from s, and other distances as induced by the shortest path through s or ℓ. When a taxi wants to stop at a virtual point

before reaching the request, we actually leave this taxi at its old position, but when computing future moves we pretend it is located at the virtual point. By the triangle inequality, this does not increase the overall distance traveled.

The intuition is that BiasedDC seeks to be in a configuration similar to the offline algorithm. Before the request, A was already at the position of the offline taxi a, whereas P may have been placed suboptimally away from any offline taxi. Therefore, we prefer to move P away from its old location as opposed to A. Accordingly, BiasedDC moves P faster towards the request (= the new position of some offline taxi).

By the following theorem, BiasedDC achieves the optimal competitive ratio of 9, matching the aforementioned lower bound and together yielding Theorem 3.2.12.

Theorem 3.2.13. BiasedDC *is* 9*-competitive for the hard* 2*-taxi problem.*

Proof. We use the potential $\Phi = 3M$, where M is the minimum matching of the two online taxis with the two offline taxis. After serving a request, when A and a are both located at the same point, M is simply the distance $d(p, P)$ between the two passive taxis.

Let *cost* and *opt* denote the cost of BiasedDC and the offline algorithm respectively for a given request, and let $\Delta\Phi$ denote the change in potential due to serving this request. It suffices to show that

$$cost + \Delta\Phi \leq 9\,opt. \tag{3.25}$$

Summing this inequality over all request yields the result because Φ is initially 0 and remains non-negative.

For relocation requests, no cost is incurred and the potential remains unchanged, hence (3.25) is satisfied.

Consider now some simple request. We can assume without loss of generality that serving the request lasts exactly one time unit, so A moves distance 1 and P moves distance 2. Thus, $cost = 3$. We distinguish two cases depending on whether a or p serves the request.

If a serves the request, then $opt \geq 1$ because a starts its movement from the same location as A and moves at least as far. In the old minimum matching, a was matched to A and p to P. The distance between a and A increased by $opt - 1$ and the distance between p and P increased by at most 2. Thus, the minimum matching increased by at most $opt + 1$. Putting it all together, we get

$$cost + \Delta\Phi \leq 3 + 3(opt + 1) \leq 9opt,$$

so (3.25) is shown.

If p serves the request, we divide the analysis into two steps, where first the offline algorithm moves and then BiasedDC moves. The matching may increase by at most opt in the first step due to the movement of p. During the second step, A moves a distance 1 away from its matching partner a, but P moves a distance 2 towards its matching partner p. If the matching partners change afterwards, then this would only further reduce the matching, so in the second step, the matching decreases by at least 1. Overall, the matching increases by at most $opt - 1$ for this request. Hence,

$$cost + \Delta\Phi \leq 3 + 3(opt - 1) = 3opt$$

and (3.25) follows again. □

3.2.6 A competitive algorithm for three taxis on the line

With a significant extension of BiasedDC, it is possible to obtain a competitive algorithm for three taxis when the metric space is the real line:

Theorem 3.2.14. *There exists a competitive algorithm for the hard 3-taxi problem on the line.*

The algorithm, which we call RegionTracker, moves the taxis at different speeds towards the request, similarly to BiasedDC. However, besides the location of the active taxi, the algorithm also maintains an interval around each taxi. Intuitively, the intervals are supposed to indicate regions that the taxis should explore more aggressively. For instance, if a taxi moves some distance towards a request, but

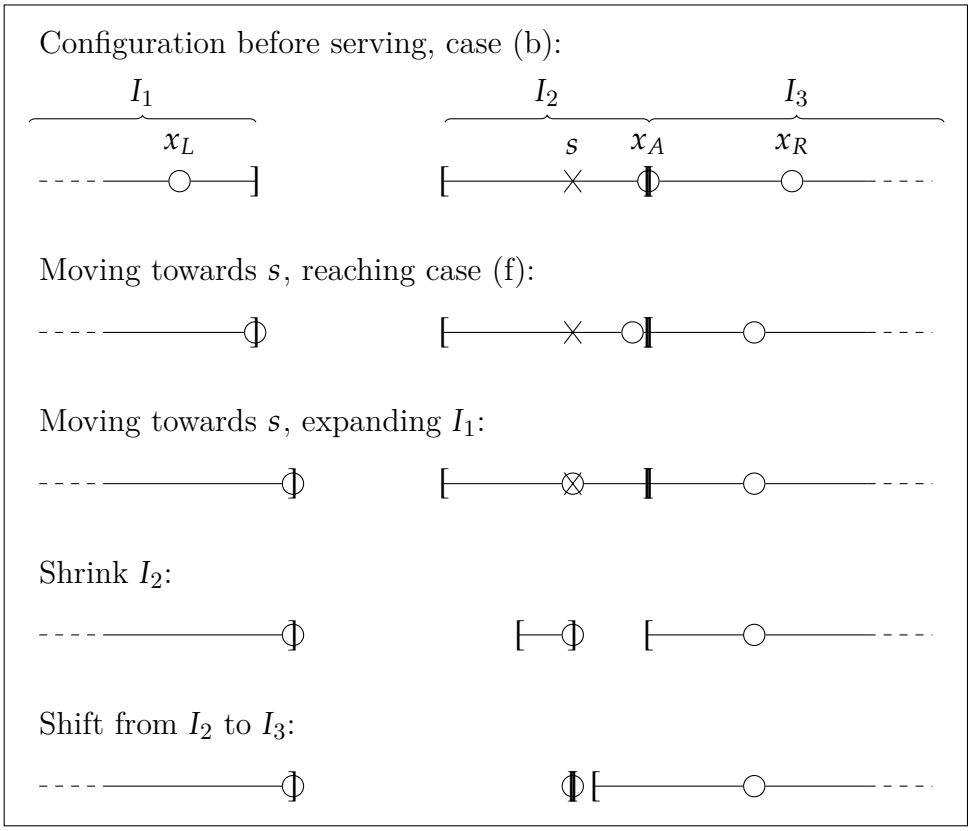

Figure 3.3: Example of RegionTracker serving a simple request at s.

stops before reaching the request, this may have been a bad move. This taxi shall therefore remember where it came from, by keeping track of the region it passed. If the taxi is moved to the opposite direction in the future, it shall move there at a faster speed so as to correct for its earlier mistake. During the run of RegionTracker, it is also possible that a taxi inherits (possibly a shifted) part of a region previously owned by another taxi. This is to enable a taxi to correct for an earlier bad move by *another* taxi.

We use in this section the notation $x \wedge y = \min\{x, y\}$ and $x \vee y = \max\{x, y\}$. Algorithm 1 contains the pseudocode of RegionTracker. An example of the steps involved in serving a simple request is depicted in Figure 3.3.

At any point in time, we denote by $x_1 \leq x_2 \leq x_3$ the locations of the algorithm's taxis. The index $A \in \{1, 2, 3\}$ indicates the *active taxi* that served the last request (or $A = 1$ initially). We use variables $r_1 \leq \ell_2 \leq r_2 \leq \ell_3$ to represent the intervals $I_1 = (-\infty, r_1]$, $I_2 = [\ell_2, r_2]$ and $I_3 = [\ell_3, \infty)$, and we will ensure at all times that $x_i \in$

Algorithm 1 RegionTracker

Require: Initial taxi locations $x_1 \leq x_2 \leq x_3$

$A \leftarrow 1$
$(r_0, \ell_1, r_1, \ell_2, r_2, \ell_3, r_3, \ell_4) \leftarrow (-\infty, -\infty, x_1, x_1, x_2, x_3, \infty, \infty)$
for each request (s, t) **do**
    if $s < x_2$ **then**
        while $s \notin \{x_1, x_2\}$ **do**
            Change x_1, x_2, x_3 at rates specified in Table 3.1
            $r_1 \leftarrow (x_1 \vee r_1) \wedge x_2$
            $\ell_2 \leftarrow r_1 \vee (\ell_2 \wedge x_2)$
        end while
        $A \leftarrow \min\{i \mid x_i = s\}$
        while $\ell_A < x_A < r_A$ **do**
            Increase ℓ_A and decrease r_A at the same rate
        end while
        while $\ell_A < x_A < \ell_{A+1}$ **do**
            Increase ℓ_A and decrease ℓ_{A+1} at the same rate
        end while
        while $r_{A-1} < x_A < r_A$ **do**
            Decrease r_A and increase r_{A-1} at the same rate
        end while
    else if $s > x_2$ **then**
        Act symmetrically to case "$s < x_2$"
    end if
    $(e_1, e_2) \leftarrow (r_1, \ell_2, r_2, \ell_3) \setminus (x_A, x_A)$
    $x_A \leftarrow t$
    $(x_1, x_2, x_3) \leftarrow \text{sort}(x_1, x_2, x_3)$
    $A \leftarrow \min\{i \mid x_i = t\}$
    $(r_1, \ell_2, r_2, \ell_3) \leftarrow \text{sort}(e_1, e_2, x_A, x_A)$
end for

I_i for $i = 1, 2, 3$. For technical reasons, we also define $r_0 = \ell_1 = -\infty$ and $r_3 = \ell_4 = \infty$.

Before and after serving a request, it will always be the case that two of the four finite interval endpoints are equal to the location x_A of the active taxi. We sometimes denote the other two interval endpoints by $e_1 \leq e_2$, and the two passive taxis by $L = \min\{1, 2, 3\} \setminus \{A\}$ and $R = \max\{1, 2, 3\} \setminus \{A\}$. Let *sort* be the operator that maps a sequence of numbers to the same sequence sorted in non-decreasing order.

Observation 3.2.15. *If* $(r_1, \ell_2, r_2, \ell_3) = \text{sort}(e_1, e_2, x_A, x_A)$, *then* $x_L \in (-\infty, e_1]$ *and* $x_R \in [e_2, \infty)$.

Given a taxi request (s, t), RegionTracker moves a taxi to s as follows. For

the sake of this description, let us assume that $s < x_2$; the other case is symmetric. If $s \leq x_1$, then we simply move the leftmost taxi to s. Otherwise, in most cases (see Table 3.1) we move the two adjacent taxis continuously towards s until one of them reaches s. If one of them has reached the frontier of its interval and the other one has not, then the one which is still in the interior of its interval moves by a factor $b + 1$ or $c + 1$ faster than the one that is already at the frontier, for constants $c > b > 0$. However, there is one exception: If none of the three taxis has reached the interval frontier between itself and s, then all three taxis move towards the request at speeds $b + 1$, 1 and b. Simultaneously to moving the taxis, we will also update the interval frontiers so as to ensure that $x_i \in I_i$ continues to hold and the interiors of I_1, I_2 and I_3 are disjoint (lines 7–8 of Algorithm 1). In other words, if a taxi reaches its interval frontier, then it pushes this frontier further as it is moving; if it reaches the interval frontier of an adjacent taxi, it pushes the frontier back. Once s is reached, the active taxi index A is updated.

Since pushing the frontiers and updating A may have violated the property that the list r_1, ℓ_2, r_2, ℓ_3 contains two copies of x_A, we need to do some post-processing. In lines 11–13, we shrink interval I_A until one of the endpoints reaches x_A. Thereafter (lines 14–19), we "shift" any remaining part of I_A to the side. More precisely, if $\ell_A < x_A = r_A$ (which can only be the case for $A = 1, 2$), then we push the frontier ℓ_A towards x_A (further shrinking I_A) while pulling ℓ_{A+1} away from x_{A+1} towards x_A (enlarging I_{A+1}). This is done until either ℓ_A or ℓ_{A+1} reaches x_A. We act similarly if instead we had $\ell_A = x_A < r_A$ after line 13. After this, it is indeed true that (at least) two of the interval endpoints r_1, ℓ_2, r_2, ℓ_3 are equal to x_A.

In line 23, we define $e_1 \leq e_2$ as the other two interval endpoints, as mentioned above. (In the pseudocode, we use the set difference notation to remove elements from a list.) To serve the relocation part of the request, we simply change the location of x_A, make sure that $x_1 \leq x_2 \leq x_3$ are again in the right order, and update A accordingly. Finally, we update the interval endpoints to react to the relocation.

Table 3.1: Rates of movement if $s \in (-\infty, x_1) \cup (x_1, x_2)$, where $c > b > 0$ are constants

	Conditions	x_1'	x_2'	x_3'
(a)	$s < x_1$	-1	0	0
(b)	$s > x_1, x_1 < r_1, \ell_2 < x_2, \ell_3 < x_3$	$b+1$	-1	$-b$
(c)	$s > x_1, x_1 < r_1, \ell_2 < x_2, \ell_3 = x_3$	1	-1	0
(d)	$s > x_1, x_1 = r_1, \ell_2 = x_2$	1	-1	0
(e)	$s > x_1, x_1 < r_1, \ell_2 = x_2$	$b+1$	-1	0
(f)	$s > x_1, x_1 = r_1, \ell_2 < x_2, A \geq 2$	1	$-(b+1)$	0
(g)	$s > x_1, x_1 = r_1, \ell_2 < x_2, A = 1$	1	$-(c+1)$	0

Analysis

We use the notation $\lfloor x, y \rceil = [x \wedge y, x \vee y]$ for the interval between x and y. For the locations of the offline taxis, we write $y_1 \leq y_2 \leq y_3$, and $a \in \{1, 2, 3\}$ for the index of the active offline taxi.

To prove that RegionTracker is competitive, we use a potential consisting of two parts. One of them is

$$\Psi = \int_{-\infty}^{\infty} (w_1(z) + w_2(z) + w_3(z))dz$$

where

$$w_i(z) = \begin{cases} 0, & \text{if } z \notin \lfloor x_i, y_i \rceil, \\ \gamma - \psi, & \text{if } z \in \lfloor x_i, y_i \rceil \cap I_i, \\ \gamma + \psi, & \text{if } z \in \lfloor x_i, y_i \rceil \setminus [r_{i-1}, \ell_{i+1}], \\ \gamma, & \text{otherwise.} \end{cases}$$

for some constants $\psi > 0$ and $\gamma = \frac{2b+1}{2b}\psi$. We can think of Ψ as a special weighted matching of the online and offline configurations: The ith online taxi is matched to the ith offline taxi. The interval between them is partitioned into (up to) three segments whose contribution to Ψ is their length weighted by some factor. The segment that is in I_i has weight $\gamma - \psi$, the (possible) segment between the frontiers of I_i and the adjacent taxi's I_j has weight γ and a possibly remaining segment from the boundary of the adjacent taxi's I_j to the offline taxi has weight $\gamma + \psi$.

The other part of the potential is

$$\Sigma = \begin{cases} (r_1 - x_1) \wedge (x_2 - \ell_2), & \text{if } \ell_3 = x_3, \\ (r_2 - x_2) \wedge (x_3 - \ell_3), & \text{if } x_1 = r_1, \\ (r_1 - x_1 + r_2 - x_2) \wedge (x_2 - \ell_2 + x_3 - \ell_3) & \text{otherwise.} \end{cases}$$

Note that Σ is well-defined, since if $x_1 = r_1$ and $\ell_3 = x_3$, then Σ equates to 0 by both the first or the second case of the definition. This part has a purpose somewhat similar to the "sum of pairwise server distances" part in the proof by potential for the Double Coverage algorithm for the k-server problem [CKPV91]. For the hard k-taxi problem, a plain pairwise server distances potential does not make sense since these distances can be changed arbitrarily by relocation requests; instead, Σ is a variant of this that represents the distance between the two passive online taxis, truncated at the closest e_i:

Claim 3.2.16. *If* $(r_1, \ell_2, r_2, \ell_3) = sort(e_1, e_2, x_A, x_A)$, *then* $\Sigma = (e_1 - x_L) \wedge (x_R - e_2)$.

Proof. By Observation 3.2.15, we have $x_L \leq e_1 \leq e_2 \leq x_R$.

If $A = 1$, then $(r_1, \ell_2, r_2, \ell_3) = (x_1, x_1, e_1, e_2)$. Therefore $\Sigma = (r_2 - x_2) \wedge (x_3 - \ell_3) = (e_1 - x_L) \wedge (x_R - e_2)$. The case $A = 3$ is similar.

For $A = 2$, we consider several subcases. If $x_1 = e_1$, then $r_1 = x_1$ and $r_2 = x_2$. Therefore $\Sigma = (r_2 - x_2) \wedge (x_3 - \ell_3) = 0 = (e_1 - x_L) \wedge (x_R - e_2)$. The same argument handles the case $x_3 = e_2$, so let us assume $x_1 < e_1$ and $e_2 < x_3$. If $e_1 \leq x_2 \leq e_2$, then $(r_1, \ell_2, r_2, \ell_3) = (e_1, x_2, x_2, e_2)$ and $\Sigma = (r_1 - x_1 + r_2 - x_2) \wedge (x_2 - \ell_2 + x_3 - \ell_3) = (e_1 - x_L) \wedge (x_R - e_2)$. If $x_2 < e_1$, then $(r_1, \ell_2, r_2, \ell_3) = (x_2, x_2, e_1, e_2)$. If $x_1 = x_2$, then $\Sigma = (r_2 - x_2) \wedge (x_3 - \ell_3) = (e_1 - x_L) \wedge (x_R - e_2)$. If $x_1 < x_2$, then $\Sigma = (r_1 - x_1 + r_2 - x_2) \wedge (x_2 - \ell_2 + x_3 - \ell_3) = (e_1 - x_L) \wedge (x_R - e_2)$. The case $x_2 > e_2$ is symmetric to $x_2 < e_1$. □

As the overall potential, we use $\Phi = \alpha\Sigma + \Psi$ for some constant $\alpha > 0$.

Claim 3.2.17. Φ *remains constant during the relocation in lines 24–27 of Algorithm 1.*

Proof. By Claim 3.2.16, Σ depends only on x_L, e_1, e_2, x_R, which do not change under relocation.

To show that also Ψ remains unchanged, we show that the value of $w_1(z) + w_2(z) + w_3(z)$ is independent of the location $y_a = x_A$ of the active taxi pair for almost all z. To do so, we determine the value of $w_1(z) + w_2(z) + w_3(z)$ for

$z \notin \{x_1, x_2, x_3\}$. Let $\delta_z = |\{i\colon x_i < z\}| - |\{i\colon y_i < z\}|$ be the number of taxis that the online algorithm has to the left of z more than the offline algorithm. Then $\delta_z \in \{-2, -1, 0, 1, 2\}$, and δ_z is invariant under relocation of the active taxi pair.

If $\delta_z = 0$, then $w_1(z) + w_2(z) + w_3(z) = 0$ since $z \notin \lfloor x_i, y_i \rceil$ for all i.

Otherwise, let $m = \max\{i\colon x_i < z\}$. If $\delta_z = 2$, then $z \in \lfloor x_i, y_i \rceil$ if and only if $i \in \{m-1, m\}$. Hence, $w_1(z) + w_2(z) + w_3(z) = w_{m-1}(z) + w_m(z)$. Since $r_{m-2} \le x_{m-1} \le \ell_m \le x_m < z$, we have $w_{m-1}(z) = \gamma + \psi$. Moreover, if $A > m$ then $r_m = x_A > z$ and otherwise $r_m = \infty$. In either case, $z \in I_m$ and hence $w_m(z) = \gamma - \psi$. Thus, $w_1(z) + w_2(z) + w_3(z) = 2\gamma$ independent of the location of the active taxis.

If $\delta_z = 1$, then $w_1(z) + w_2(z) + w_3(z) = w_m(z)$. We consider several sub-cases. If $x_R < z$, then $w_m(z) = \gamma - \psi$ as in the previous case. Otherwise, $x_L < z < x_R$. If $z \le e_1$, then either $x_A \in (z, e_1]$ and $r_m = x_A$ or $x_A \notin (z, e_1]$ and $r_m = e_1$. In both cases, $z \le r_m$ and therefore $w_m(z) = \gamma - \psi$.

If $e_2 < z$, then either $x_A \in [e_2, z)$ and $\ell_{m+1} = x_A < z$ or $x_A \notin [e_2, z)$ and $\ell_{m+1} = e_2 < z$. In both cases, $w_m(z) = \gamma + \psi$.

If $z \in (e_1, e_2]$, then either $x_A \in [e_1, z)$ and $r_m = x_A < z$ or $x_A \notin [e_1, z)$ and $r_m = e_1 < z$. In both cases, $z \notin I_m$. Moreover, either $x_A \in (z, e_2]$ and $\ell_{m+1} = x_A > z$ or $x_A \notin (z, e_2]$ and $\ell_{m+1} = e_2 \ge z$. In both cases, $z \in [x_m, \ell_{m+1}] \subseteq [r_{m-1}, \ell_{m+1}]$. Thus, $w_m(z) = \gamma$ in this case, independent of the location of the active taxis.

The cases $\delta_z \in \{-2, -1\}$ are symmetric to $\delta_z \in \{1, 2\}$. □

We need to show that when serving simple requests (lines 4–22), the cost of RegionTracker plus the change of Φ is bounded by a constant times the offline cost. We first observe that Φ is non-increasing during the shrink and shift steps of the algorithm.

Claim 3.2.18. *During the shrink step (lines 11–13), Φ does not increase.*

Proof. It is easy to see that Σ does not increase.

Regarding Ψ, note the offline algorithm must have a taxi y_a at x_A. If $A = a$, then clearly Ψ can only decrease. If $a < A$, then $\int w_A(z)dz$ may increase at rate

at most ψ, but at the same time $\int w_{A-1}(z)dz$ decreases at rate ψ. Similarly for $A > a$. □

Claim 3.2.19. *During the shift step (lines 14–19),* Φ *does not increase.*

Proof. Similar to the proof of Claim 3.2.18. □

When the offline algorithm moves a taxi, Φ can only increase by at most $\gamma + \psi = O(1)$ times the distance moved by the offline algorithm. Moreover, if the offline algorithm serves the new (simple) request by moving the active taxi from y_a that also served the last request, then also the cost of RegionTracker for this request is at most a constant times the the offline cost: This is because RegionTracker also has a taxi starting at $y_a = x_A$, and clearly the cost of RegionTracker to serve a request is at most a constant (depending on b and c) times the distance from x_A to s. So if the offline algorithm moves the same active taxi twice in a row, then the increase in potential plus the online cost is at most a constant times the offline cost for this request. Thus, it only remains to show now that if the offline algorithm has already moved a taxi to the new request, but this was *not* the previously active offline taxi, then the cost of RegionTracker is cancelled by a decrease in potential. This is established in the following last Claim of this section.

Claim 3.2.20. *If* $x_A = y_a$ *and* $s = y_i$ *for some* $i \neq a$ *before* RegionTracker *serves a simple request at* s*, then the movement cost of* RegionTracker *to serve the request is at most the amount by which* Φ *decreases at the same time.*

Proof. We show for all cases (a)–(g) of Table 3.1 that $cost' + \Phi' \leq 0$ almost always, where $cost' = |x_1'| + |x_2'| + |x_3'|$ is the instantaneous movement cost of RegionTracker and Φ' the rate of change of Φ. Technically, the values x_i' and Φ' are only well-defined when x_i and Φ are differentiable as a function of time, which they are not e.g. when the condition in Table 3.1 changes. However, they are differentiable almost everywhere and it suffices to show it for these times.

In case (a), we have $cost' = 1$. Moreover, x_1 moves towards y_1 and therefore $\Psi' = -(\gamma - \psi)$. Even though Σ may increase when x_1 decreases, in the subsequent

shrink step r_1 will be reduced to the new value of x_1, which cancels any previous increase. So overall, Σ does not increase. The claim follows for $\gamma - \psi$ large enough.

In all other cases, we have $x_1 < s < x_2$. Denote by $\hat{x}_i$, $\hat{\ell}_i$ and $\hat{r}_i$ the values that x_i, ℓ_i and r_i had at the beginning of the while-loop. Then $y_a = \hat{x_A}$, and $(\hat{r_1}, \hat{\ell_2}, \hat{r_2}, \hat{\ell_3}) = \text{sort}(e_1, e_2, y_a, y_a)$. Since taxis only move towards s, the current interval endpoints are $r_1 = (\hat{r_1} \vee x_1) \wedge x_2$, $\ell_2 = r_1 \vee (\hat{\ell_2} \wedge x_2)$, $r_2 = \hat{r_2}$ and $\ell_3 = \hat{\ell_3}$.

Observe that if one of the inequalities $x_1 \leq r_1$, $\ell_2 \leq x_2$ or $\ell_3 \leq x_3$ becomes tight, then it remains tight throughout the while-loop. In particular, the case in the definition of Σ changes at most once during each run of the while-loop, and Σ can only decrease if this happens.

We will show for the cases (b) and (c) that Σ decreases at some constant rate and Ψ does not increase. Choosing α large enough, this will be enough to handle these cases. For the remaining cases, observe that Σ increases at an at most constant rate (for fixed b and c), which is immediate from the definition of Σ and the fact that the x_i change at an at most constant rate. Thus, to handle cases (d)–(g), it suffices to show that Ψ decreases at an at least constant rate. Choosing γ and ψ large enough, the decrease of Ψ cancels the increase of Σ and the cost of the algorithm.

Case (b): We must have $\hat{r_1} = r_1$ and $\hat{\ell_2} = \ell_2$. Moreover, it must be that $y_a = \ell_3 = r_2$ since otherwise the active online taxi would have moved away from s. The interval endpoints ℓ_i and r_i remain constant during this case, and therefore $\Sigma' \leq (-(b+1)+1) \vee (-1-b) = -b$.

For the change in Ψ, let us consider first the case $x_1 < y_1$. Then x_1 moves towards y_1, and for each z that x_1 moves past, $w_1(z)$ changes from $\gamma - \psi$ to 0. So the movement of x_1 decreases Ψ at rate $(b+1)(\gamma - \psi)$. For $i = 2, 3$, the movement of x_i is in the worst case away from y_i, but it remains in the interior of I_i. So for any z passed by x_i, $w_i(z)$ changes from 0 to $\gamma - \psi$ in the worst case. So the movements of x_2 and x_3 increase Ψ at rates at most $(\gamma - \psi)$ and $b(\gamma - \psi)$, respectively. Overall, $\Psi' \leq 0$.

The case $y_1 = x_1$ can be ignored because this will be the case only for a time interval of length 0.

If $y_1 < x_1$, then $y_2 = s$ and $y_3 = y_a = \ell_3$. In this case, the movement of x_1 increases Ψ at rate $(b+1)(\gamma - \psi)$, but x_2 and x_3 move towards y_2 and y_3, respectively, decreasing Ψ at rates $b(\gamma - \psi)$ and $\gamma - \psi$, respectively. Again, $\Psi' \leq 0$.

Case (c): As in case (b), we have $y_a = \ell_3$. Thus, $y_a = x_3$.

Again, the interval endpoints remain constant during case (c). Therefore, $\Sigma' = -1$.

If $y_1 = s$, then x_1 moves towards y_1, decreasing Ψ at rate $\gamma - \psi$, while the movement of x_2 increases Ψ at most at rate $\gamma - \psi$. Otherwise, $y_2 = s$, the movement of x_2 decreases Ψ at rate $\gamma - \psi$ and the movement of x_1 increases Ψ at most at rate $\gamma - \psi$. In both cases, $\Psi' \leq 0$.

As mentioned before, we show in the remaining cases only that Ψ decreases at a constant rate.

Case (d): We have $\hat{r_1} < s < \hat{\ell_2} < y_a \leq y_3$, so $s = y_1$ or $s = y_2$. If $s = y_1$, then x_1 moves towards y_1, contributing a decrease at rate γ to Ψ. Even if x_2 moves away from y_2, it contributes an increase at a rate of at most $\gamma - \psi$ to Ψ. So in total, $\Psi' \leq -\psi$. The case $s = y_2$ is similar.

Case (e): If $s = y_1$, then the movement of x_1 contributes a decrease at rate $(b+1)(\gamma - \psi)$ and the movement of x_2 contributes an increase at rate at most $\gamma - \psi$. Together, $\Psi' \leq -b(\gamma - \psi)$. If $y_1 < s$, then $y_2 \leq s$ and the movement of x_1 contributes an increase at rate at most $(b+1)(\gamma - \psi)$ while the movement of x_2 contributes a decrease at rate at least γ. Together, $\Psi' \leq (b+1)(\gamma - \psi) - \gamma = -b(\gamma - \psi)$, since $\gamma = (2b+1)(\gamma - \psi)$.

Case (f): The calculations are essentially the same as in case (e).

Case (g): Since $A = 1$, $y_a = \hat{r_1} < s$, so $s = y_2$ or $s = y_3$. For $s = y_2$ we get $\Psi' \leq -c(\gamma - \psi)$ similar to cases (e) and (f). However, for $s = y_3$ it could be that $y_2 \leq \ell_2 = r_1$, so that x_1's movement is pushing ℓ_2 towards x_2, leading to an additional contribution of $+2\psi$ to the change of Ψ. But we still have $\Psi' \leq 2\psi - c(\gamma - \psi)$, which is negative for c large enough. □

We conclude that RegionTracker achieves a constant competitive ratio, proving Theorem 3.2.14.

3.3 The easy k-taxi problem

We now turn to the easy k-taxi problem. We prove that it is exactly equivalent to the k-server problem, tightening the previous result which showed that the difference between their competitive ratios is at most 2 [Kos96].

Theorem 3.3.1. *The easy k-taxi problem has the same deterministic/randomized competitive ratio as the k-server problem.*

Proof. Clearly, since the k-taxi problem is a generalization of the k-server problem, its competitive ratio is at least that of the k-server problem. Thus, it suffices to show that given a ρ-competitive algorithm A for the k-server problem, we can construct a $(\rho + \frac{1}{N})$-competitive algorithm A_N for the easy k-taxi problem, for any $N \in \mathbb{N}$. The following proof is for deterministic algorithms. The only change that would need to be made for randomized algorithms is to replace $cost_A$ and $cost_{A_N}$ by their expectation.

The idea of algorithm A_N is to simulate the behavior of A on the request sequence obtained by replacing a k-taxi request (s, t) by many k-server requests along a shortest path from s to t. In general, the underlying metric space (M, d) may not contain any points on a shortest path from s to t; we can easily fix this by embedding M into a larger metric space $\tilde{M}$ that contains some additional virtual points. More precisely, $\tilde{M}$ is the metric space obtained by from M by adding, for each $x, y \in M$, a line segment L_{xy} with Euclidean metric and length $d(x, y)$ to M by gluing its endpoints to x and y respectively. We transform a k-taxi request sequence σ_{taxi} on M into a k-server request sequence σ_{server} on $\tilde{M}$ by replacing a k-taxi request (s, t) by a subsequence $r_0, \ldots, r_{2kN}$ of k-server requests placed along L_{st}, with $r_0 = s$, $r_{2kN} = t$ and distance $\frac{d(s,t)}{2kN}$ between two successive requests.

Clearly,

$$opt(\sigma_{server}) \leq opt(\sigma_{taxi})$$

because an optimal schedule for σ_{taxi} can be turned into a valid schedule for σ_{server} of the same cost by using the server that would serve a taxi request (s, t) to serve all

the associated k-server requests $r_0, \ldots, r_{2kN}$. Therefore, since A is ρ-competitive on $\tilde{M}$,

$$\begin{aligned} cost_A(\sigma_{server}) &\leq \rho \cdot opt(\sigma_{server}) + c \\ &\leq \rho \cdot opt(\sigma_{taxi}) + c \end{aligned} \tag{3.26}$$

for some constant c.

The idea of algorithm A_N is to transform A's schedule for σ_{server} into a valid schedule for σ_{taxi} while incurring an additional cost of at most $opt(\sigma_{taxi})/N$. To define A_N, we will pretend that taxis of A_N can be located at virtual points in $\tilde{M} \setminus M$ even though this is not possible in the original metric space M. However, this will only ever happen when A_N makes a move that does not serve a request, so A_N does not actually have to carry out such a move and can keep the taxi in its old position until it is used to serve a request. Due to the triangle inequality, this will not increase the overall cost.

We can make the following two assumptions about A when it serves the subsequence $r_0, \ldots, r_{2kN}$ of equidistant requests on L_{st} associated to the taxi request (s, t): First, A is lazy, so to serve r_i it moves one server to r_i and moves no other server. Second, for $i \geq 2$, A never serves r_i with a server located at r_j for some $j \leq i - 2$; this is because A could instead move the last used server from r_{i-1} to r_i and (non-lazily) move the server from r_j to r_{i-1} to end up in the same configuration for the same cost, but then A may as well delay the non-lazy move until later when/if this server is actually used to serve a request. These two assumptions mean that the requests $r_0, \ldots, r_{2kN}$ can be partitioned into at most k blocks of adjacent requests such that all requests within the same block are served by the same server and requests in different blocks are served by different servers. Formally, if $\ell \leq k$ is the number of servers used to serve $r_0, \ldots, r_{2kN}$, then there are indices $i_0 = -1 < i_1 < \cdots < i_\ell = 2kN$ such that A uses the jth of these servers to serve all the requests $r_{i_{j-1}+1}, r_{i_{j-1}+2}, \ldots, r_{i_j}$. To turn this into a valid way to serve the taxi request (s, t), we have to ensure that the same server/taxi that serves $r_0 = s$ will also end up at $r_{2kN} = t$. For this, we will let the same server serve all the

requests $r_0, \ldots, r_{2kN}$, which can be done at a small additional cost: Namely, at the transition between blocks where A uses a new server to serve r_{i_j+1} instead of reusing the old server from r_{i_j}, algorithm A_N will carry out the same server movement as A, followed by swapping the two servers at r_{i_j} and r_{i_j+1}. It remains to analyze the cost of A_N.

Since the distance between the two adjacent requests involved in a swap is $\frac{d(s,t)}{2kN}$, swapping the servers yields an additional cost of $\frac{d(s,t)}{kN}$. Therefore, total cost of all $\ell - 1 < k$ swaps associated with the request (s, t) is at most $\frac{d(s,t)}{N}$. Over the entire request sequence, the total cost of swaps is at most a $\frac{1}{N}$ fraction of the sum of the distances of all start-destination pairs in σ_{taxi}. Since the optimal algorithm must pay at least all of these distances, we have

$$\begin{aligned} cost_{A_N}(\sigma_{taxi}) &\leq cost_A(\sigma_{server}) + \frac{1}{N} opt(\sigma_{taxi}) \\ &\leq \left(\rho + \frac{1}{N}\right) opt(\sigma_{taxi}) + c, \end{aligned}$$

where the last inequality follows from (3.26). □

3.4 Conclusion and open problems

The most important open problem is whether there exists an algorithm for the hard k-taxi problem on general metric spaces with competitive ratio based only on k, i.e., avoiding the dependency on n in Corollary 3.2.2. One can show that the work function algorithm, which achieves the best known upper bound of $2k - 1$ for the k-server problem, has unbounded competitive ratio for the hard k-taxi problem, even for $k = 2$. However, the *generalized work function algorithm*, a less greedy variant, may be competitive. This algorithm is $O(k2^k)$-competitive for k-LGT [Bur96], but we do not see any direct way to adapt the proof to yield a similar competitive ratio for the hard k-taxi problem. In any case, the connection between the k-taxi and k-LGT problems is intriguing. Another way to obtain an $f(k)$-competitive algorithm for general metrics may be via dynamically updating the HST embedding, similarly to [Lee18a].

We believe that our algorithm for three taxis on the line can be the foundation to solve the problem more generally, i.e., for general k and/or more general metrics such as trees or arbitrary metrics. One interesting metric space — due to its obvious application to the k-taxi problem — is the 2-dimensional ℓ_1-norm (also known as taxicab metric and Manhattan metric). The lower bounds of $2^k - 1$ hold even for the line and the ℓ_1-norm: This is because the *binary* α-HSTs from Theorems 3.2.6 and 3.2.8 can be embedded into the line, with a distortion tending to 1 as $\alpha \to \infty$.

For HSTs, the main open question is whether with memory and against oblivious adversaries one can break the exponential barrier. We conjecture that the competitive ratio of $2^k - 1$ on HSTs can also be achieved by a deterministic algorithm, namely the Double Coverage algorithm [CL91]. For $k = 2$ this can be shown using the same potential as for Flow (and the fact that root-leaf-paths have the same length), however it is easy to see that this potential fails for $k > 2$. For weighted star metrics one can show that Double Coverage achieves the optimal competitive ratio for the hard k-taxi problem, and this is $2k - 1$.[7] If our conjecture holds, then this would mean that the deterministic competitive ratio is identical to the randomized memoryless competitive ratio against oblivious adversaries on HSTs. Notice that the same is known to be true for the k-server problem at least on some metric spaces, where tight bounds of k are known for both deterministic as well memoryless randomized algorithms (cf. [Kou09]; see also [RS94]). It would be interesting to prove this as a generic result for a broad class of online problems.

[7] For the upper bound, one can use a weighted matching as potential, where distances between online taxis and the root are scaled by a factor $2k - 1$.

4

Pure Entropic Regularization for Metrical Task Systems

4.1 Introduction

We consider randomized algorithms for the metrical task systems (MTS) problem: Let (X, d_X) be a metric space with $|X| = n > 1$. A randomized algorithm A starts in some initial state $\rho_0 \in X$. A sequence $\boldsymbol{c} = (c_1, c_2, \ldots, c_m)$ of nonnegative cost functions $c_t \colon X \to \mathbb{R}_+$ is revealed one by one. At each time t, algorithm A selects a random state $\rho_t \in X$, incurring service cost $c_t(\rho_t)$ and movement cost $d_X(\rho_{t-1}, \rho_t)$. Denote by

$$\mathsf{S}_A(\boldsymbol{c}) := \mathbb{E} \sum_{t=1}^{m} c_t(\rho_t) \quad \text{and} \quad \mathsf{M}_A(\boldsymbol{c}) := \mathbb{E} \sum_{t=1}^{m} d_X(\rho_{t-1}, \rho_t)$$

the total expected service and movement cost of A, respectively. Recall the definition of the refined guarantees: Algorithm A is called *α-competitive for service costs* and *α'-competitive for movement costs* if for every cost $\boldsymbol{c}$, it holds that

$$\mathsf{S}_A(\boldsymbol{c}) \leq \alpha \cdot \mathit{opt}(\boldsymbol{c}) + \beta$$
$$\mathsf{M}_A(\boldsymbol{c}) \leq \alpha' \cdot \mathit{opt}(\boldsymbol{c}) + \beta',$$

where $\beta, \beta' \geq 0$ are constants independent of $\boldsymbol{c}$. We obtain optimal refined guarantees for HSTs (see Section 4.1.1 for the formal definition of HSTs used in this chapter).

Theorem 4.1.1. *On any n-point HST metric X, there is a randomized online algorithm that is 1-competitive for service costs and $O(\log n)$-competitive for movement costs.*

Remark 4.1.2 (Optimality of the refined guarantees)**.** Any finitely competitive algorithm for MTS on an n-point uniform metric cannot be better than $\Omega(\log n)$-competitive for movement costs, regardless of its competitive ratio for service costs. This is because this lower bound holds even if the cost functions only take values 0 and ∞. Moreover, it cannot be better than 1-competitive for service costs, regardless of its competitive ratio for movement costs. To see this, consider the case where each cost function is the constant function 1.

We also define the following even more refined guarantees. Suppose that for some numbers $\alpha_0, \alpha_1, \gamma, \beta, \beta' > 0$, a randomized online algorithm A satisfies, for every cost $\boldsymbol{c}$ and every offline algorithm ADV:

$$\mathsf{S}_A(\boldsymbol{c}) \leq \alpha_0 \mathsf{S}_{ADV}(\boldsymbol{c}) + \alpha_1 \mathsf{M}_{ADV}(\boldsymbol{c}) + \beta \tag{4.1}$$

$$\mathsf{M}_A(\boldsymbol{c}) \leq \gamma \mathsf{S}_A(\boldsymbol{c}) + \beta' . \tag{4.2}$$

In this case, we say that A is $(\alpha_0, \alpha_1, \gamma)$-*finely competitive*. We establish the following.

Theorem 4.1.3. *On any n-point HST metric X, for every $\kappa \geq 1$, there is an online randomized algorithm that is $(1, 1/\kappa, O(\kappa \log n))$-finely competitive. In fact, one can take $\beta = 0$ and $\beta' \leq O(\kappa \mathrm{diam}(X))$.*

Combined with the random embedding from [FRT04], this yields the following consequence for general n-point metric spaces.

Corollary 4.1.4. *On any n-point metric space, there is an online randomized algorithm that is 1-competitive for service costs and $O((\log n)^2)$-competitive for movement costs.*

Proof. Consider an n-point metric space (X, d_X). It is known [FRT04] that there exists a random function $d_T \colon X \times X \to \mathbb{R}_+$ so that (X, d_T) is a random HST metric and for all $x, y \in X$:

(a) $\mathbb{P}[d_T(x,y) \geq d_X(x,y)] = 1$,

(b) $\mathbb{E}[d_T(x,y)] \leq D \cdot d_X(x,y)$,

and $D \leq O(\log n)$.

Let A_T be the randomized algorithm for (X, d_T) guaranteed by Theorem 4.1.3 with $\kappa = D$. Let A denote the algorithm that results from sampling d_T and then using A_T on (X, d_T). We use M^T to denote movement cost measured in d_T and M^X for movement cost measured in d_X.

Then for any cost $\boldsymbol{c}$ and any offline algorithm ADV, we have

$$\begin{aligned}\mathsf{S}_A(\boldsymbol{c}) = \mathbb{E}[\mathsf{S}_{A_T}(\boldsymbol{c})] &\leq \mathsf{S}_{ADV}(\boldsymbol{c}) + \kappa^{-1}\,\mathbb{E}[\mathsf{M}^T_{ADV}(\boldsymbol{c})] + O(1)\\ &\leq \mathsf{S}_{ADV}(\boldsymbol{c}) + \kappa^{-1} D \mathsf{M}^X_{ADV}(\boldsymbol{c}) + O(1)\\ &= \mathsf{S}_{ADV}(\boldsymbol{c}) + \mathsf{M}^X_{ADV}(\boldsymbol{c}) + O(1)\,,\end{aligned}$$

and

$$\mathsf{M}^X_A(\boldsymbol{c}) = \mathbb{E}[\mathsf{M}^X_{A_T}(\boldsymbol{c})] \leq \mathbb{E}[\mathsf{M}^T_{A_T}(\boldsymbol{c})] \leq O(\kappa \log n)\,\mathbb{E}[\mathsf{S}_{A_T}(\boldsymbol{c})] + O(1),$$

completing the proof. □

4.1.1 HST metrics

We use a slightly different definition of HSTs than we did in Chapter 3, which still has the property that any n-point metric is $O(\log n)$-approximated by an HST in the sense as employed in Corollary 4.1.4.

Let $T = (V, E)$ be a finite tree with root $\mathbb{r}$ and vertex weights $\{w_u > 0 : u \in V\}$, let $\mathcal{L} \subseteq V$ denote the leaves of T, and suppose that the vertex weights on T are non-increasing along root-leaf paths. Consider the metric space $(\mathcal{L}, d_T)$, where $d_T(\ell, \ell')$ is the weighted length of the path connecting ℓ and ℓ' when the edge from a node u to its parent is w_u. We will use $\mathfrak{D}_T$ for the combinatorial (i.e., unweighted)

depth of T. We write $\chi(u)$ for the set of children of u and $\mathcal{L}_u$ for the set of leaves in the subtree rooted at u. For $u \neq \mathbb{r}$, we denote by $\mathsf{p}(u)$ the parent of u.

$(\mathcal{L}, d_T)$ is called an *HST metric* (or, equivalently for finite metric spaces, an *ultrametric*). If, for some $\tau > 1$, the weights on T satisfy the stronger inequality $w_v \leq w_u/\tau$ whenever v is a child of u, the space $(\mathcal{L}, d_T)$ is said to be a τ*-HST metric.*

4.1.2 The fractional model on trees

We will work in the following deterministic fractional setting, which is equivalent to the setting of randomized (integral) algorithms described earlier (see [BCLL19, §2]). The state of a fractional algorithm is given by a point in the polytope

$$\mathsf{K}_T := \left\{ x \in \mathbb{R}_+^V : x_{\mathbb{r}} = 1,\ x_u = \sum_{v \in \chi(u)} x_v \quad \forall u \in V \setminus \mathcal{L} \right\}. \tag{4.3}$$

A state $x \in \mathsf{K}_T$ corresponds to the situation that the state of a randomized integral algorithm is in $\mathcal{L}_u$ with probability x_u, for each $u \in V$. Note that K_T is simply an affine encoding of the probability simplex on $\mathcal{L}$. In the fractional setting, changing from state x to x' incurs movement cost $\|x - x'\|_{\ell_1(w)}$, where

$$\|z\|_{\ell_1(w)} := \sum_{u \in V} w_u |z_u|$$

denotes the weighted ℓ_1-norm on $\mathbb{R}^V$.

A cost function can be viewed as a vector $c \in \mathbb{R}_+^{\mathcal{L}}$. The associated service cost of the fractional state x can be written as $\langle c, x \rangle_{\mathcal{L}}$, where

$$\langle y, z \rangle_{\mathcal{L}} := \sum_{\ell \in \mathcal{L}} y_\ell z_\ell,$$

denotes the inner product of the projection of vectors $y, z \in \mathbb{R}^{\mathcal{L}} \cup \mathbb{R}^V$ onto $\mathbb{R}^{\mathcal{L}}$.

4.1.3 Mirror descent, metric filtrations, and regularization

Following [BCLL19], our algorithm is based on the mirror descent framework as established in [BCL+18]. This is a method for regularized online convex optimization, an approach that was previously explored for competitive analysis in [ABBS10, BCN14].

A central component of mirror descent is choosing the appropriate mirror map (which we will often refer to as the "regularizer"). This is a strictly convex function $\Phi : \mathsf{K}_T \to \mathbb{R}$ that endows K_T with a geometric (Riemannian) structure, specifying how to perform constrained vector flow. In other words, it specifies how one can move in a preferred direction while remaining inside K_T.

The paper [BCLL19] employs the following regularizer:

$$\Phi_0(x) := \frac{1}{\eta} \sum_{u \in V \setminus \{\mathbb{r}\}} w_u \left(x_u + \delta_u\right) \log \left(x_u + \delta_u\right) , \tag{4.4}$$

with $\eta \asymp \log |\mathcal{L}|$ and $\delta_u = |\mathcal{L}_u|/|\mathcal{L}|$. Using this regularizer, they prove for weighted trees (not just HSTs) of depth $\mathfrak{D}_T$ that a competitive ratio of $O(\mathfrak{D}_T \log n)$ can be achieved.

Metric filtrations

It is straightforward that one can think of Φ_0 as a type of multiscale entropy (this is the *negative* of the associated Shannon entropy, since we use the analyst's convention that the entropy is convex). To understand this notion, let us forget momentarily the weights on T. Then the structure of T gives a natural filtration over probability measures on the leaves $\mathcal{L}$. Suppose that $\boldsymbol{X}$ is a random variable taking values in $\mathcal{L}$ and, for $u \in V$, denote by $\mathcal{E}_u$ the event $\{\boldsymbol{X} \in \mathcal{L}_u\}$. Then the chain rule for Shannon entropy yields

$$\sum_{\ell \in \mathcal{L}} \mathbb{P}[\mathcal{E}_\ell] \log \frac{1}{\mathbb{P}[\mathcal{E}_\ell]} = \sum_{u \in V \setminus \{\mathbb{r}\}} \mathbb{P}[\mathcal{E}_u] \log \frac{\mathbb{P}[\mathcal{E}_{\mathsf{p}(u)}]}{\mathbb{P}[\mathcal{E}_u]}.$$

If we now imagine that uncertainty at higher scales is more costly than uncertainty at lower scales, then we might define an analogous *weighted* entropy by

$$\sum_{u \in V \setminus \{\mathbb{r}\}} w_u \, \mathbb{P}[\mathcal{E}_u] \log \frac{\mathbb{P}[\mathcal{E}_{\mathsf{p}(u)}]}{\mathbb{P}[\mathcal{E}_u]}. \tag{4.5}$$

Such a notion is natural in the context of "metric learning" problems.

Ignoring the $\{\delta_u\}$ values for a moment, consider that (4.4) is not analogous to (4.5). Indeed, it corresponds to the quantity

$$\sum_{u \in V \setminus \{\mathbb{r}\}} w_u \, \mathbb{P}[\mathcal{E}_u] \log \frac{1}{\mathbb{P}[\mathcal{E}_u]}, \tag{4.6}$$

and now one can see a fundamental reason why the algorithm associated to (4.4) only achieves a competitive ratio $O(\mathfrak{D}_T \log n)$ rather than $O(\log n)$: The quantity (4.6) *overmeasures* the metric uncertainty.

Suppose that $\boldsymbol{X}$ is a uniformly random leaf. Then $\sum_{\ell \in \mathcal{L}} \mathbb{P}[\mathcal{E}_\ell] \log \frac{1}{\mathbb{P}[\mathcal{E}_\ell]} = \log n$, where $n = |\mathcal{L}|$. But, in general, one could have $\sum_{u \in V} \mathbb{P}[\mathcal{E}_u] \log \frac{1}{\mathbb{P}[\mathcal{E}_u]} \geq \Omega(\mathfrak{D}_T \log n)$. Since the vertex weights are decreasing geometrically down root-leaf paths, the quantity (4.6) is actually within an $O(1)$ factor of (4.5), but given the manner in which the regularizer distorts the geometry, the overlap effect occurs as for the unweighted entropy. This fact was not lost on the authors of [BCLL19], but they bypass the problem and still obtain an $O(\log n)$-competitive algorithm for HSTs (without the refined guarantee of being 1-competitive for service costs) by combining mirror descent on stars with a recursive composition method called "unfair gluing".

Multiscale conditional entropy

We employ a regularizer that is a more faithful analog of (4.5):

$$\Phi(x) := \sum_{u \in V \setminus \{\mathbb{r}\}} \frac{w_u}{\eta_u} \left(x_u + \delta_u x_{\mathsf{p}(u)}\right) \log \left(\frac{x_u}{x_{\mathsf{p}(u)}} + \delta_u\right) \tag{4.7}$$

If one ignores the additional parameters $\{\eta_u \geq 1, \delta_u > 0\}$, this is precisely the negative weighted Shannon entropy written according to the chain rule. Here, we set

$$\theta_u := \frac{|\mathcal{L}_u|}{|\mathcal{L}_{\mathsf{p}(u)}|} \tag{4.8}$$

$$\eta_u := 1 + \log(1/\theta_u) \tag{4.9}$$

$$\delta_u := \theta_u / \eta_u \,. \tag{4.10}$$

The numbers $\{\theta_u\}$ are the conditional probabilites of the uniform distribution on leaves. The $\{\delta_u\}$ values are employed as "noise" added to the entropy calculation. Such noise is a fundamental aspect for competitive analysis, and distinguishes it from the application of mirror descent to regret minimization problems (see, e.g., [BC12]). The effect of these noise parameters appears ubiquitously in applications of the primal-dual method to competitive analysis (see [BN09]), and manifests itself as

an additive term in the update rules (see (4.11) below). Intuitively, it ensures that the conditional probability $\frac{x_u}{x_{\mathsf{p}(u)}}$ is updated fast enough even when it is close to 0.

Finally, the numbers $\{\eta_u : u \in V\}$ are commonly referred to as "learning rates" in the study of online learning. They represent the rate at which information is discounted in the resulting algorithm; for MTS, this corresponds to the relative importance of costs arriving now vs. costs that arrived in the past.

The dynamics

We will derive in Section 4.3 the following *continuous time* evolution of the resulting mirror descent algorithm ($x(t) \in \mathsf{K}_T : t \in [0,\infty)$) for a cost path $c\colon [0,\infty) \to \mathbb{R}_+^{\mathcal{L}}$:

$$\partial_t \left(\frac{x_u(t)}{x_{\mathsf{p}(u)}(t)}\right) = \frac{\eta_u}{w_u}\left(\frac{x_u(t)}{x_{\mathsf{p}(u)}(t)} + \delta_u\right)\left(\beta_{\mathsf{p}(u)}(t) - \sum_{\ell \in \mathcal{L}_u} \frac{x_\ell(t)}{x_u(t)} c_\ell(t)\right) \tag{4.11}$$

Here, $\beta_{\mathsf{p}(u)}(t)$ is a Lagrangian multiplier that ensures conservation of conditional probability:

$$\sum_{v \in \chi(\mathsf{p}(u))} \partial_t \left(\frac{x_v(t)}{x_{\mathsf{p}(u)}(t)}\right) = 0\,.$$

One can see that the evolution of $\frac{x_u}{x_{\mathsf{p}(u)}}$ is being driven by the expected instantaneous cost incurred conditioned on the current state being in $\mathcal{L}_u$.

One should interpret (4.11) only when $x(t)$ lies in the relative interior of K_T. Otherwise, the conditional probabilities are ill-defined. One way to rectify this is to prevent $x(t)$ from hitting the relative boundary of K_T at all. It is possible to adaptively modify the cost functions by a suitably small perturbation so as to guarantee this property and, at the same time, ensure that the total discrepancy between the modified and true service cost is a small additive constant.

Instead, we will follow a different approach, by extending the dynamics to an analogous system of conditional probabilities $\{q_u(t) : u \in V \setminus \{\mathbb{r}\}\}$:

$$\partial_t q_u(t) = \frac{\eta_u}{w_u}\left(q_u(t) + \delta_u\right)\left(\beta_{\mathsf{p}(u)}(t) - \hat{c}_u(t) + \alpha_u(t)\right), \tag{4.12}$$

where $q_u(t) = \frac{x_u(t)}{x_{\mathsf{p}(u)}(t)}$ whenever $x_{\mathsf{p}(u)}(t) > 0$, $\alpha_u(t)$ is a Lagrangian multiplier for the constraint $q_u(t) \geq 0$, and $\hat{c}_u(t)$ is the "derived" cost in the subtree rooted at u:

$$\hat{c}_u(t) := \sum_{\ell \in \mathcal{L}_u} q_{\ell|u}(t) c_\ell(t)$$
$$q_{\ell|u}(t) := \prod_{v \in \gamma_{u,\ell} \setminus \{u\}} q_v(t)\,,$$

where $\gamma_{u,\ell}$ is the unique simple u-ℓ path in T.

Stated this way, the mirror descent algorithm can be envisioned as running a "weighted star" algorithm on the conditional probabilities at every internal node of T, with the derived costs at an internal node u given by the expected service cost of the current strategy conditioned on the state being in $\mathcal{L}_u$.

In the next section, we will implement and analyze a discretization of (4.12) using Bregman projections. The major benefit of the continuous formulations (4.11) and (4.12) is in motivating such an algorithm and prescribing the derived costs. In Section 4.3.3, we show that taking the discretization parameter in our algorithm to zero, the discrete algorithm converges to the continuous dynamics.

4.2 The MTS algorithm

Consider a convex polytope $\mathsf{K}_0 \subseteq \mathbb{R}^n$, define $\mathsf{K} := \mathsf{K}_0 \cap \mathbb{R}^n_+$, and assume that K is compact. Suppose additionally that $\Phi : \mathcal{D} \to \mathbb{R}$ is differentiable and strictly convex in an open neighborhood $\mathcal{D} \supseteq \mathsf{K}$.

Let us write D_Φ for the corresponding Bregman divergence

$$\mathsf{D}_\Phi(y \,\|\, x) := \Phi(y) - \Phi(x) - \left\langle \nabla\Phi(x), y - x \right\rangle,$$

which is non-negative due to convexity of Φ. Then for $x, y, z \in \mathsf{K}$, we have:

$$\mathsf{D}_\Phi(z \,\|\, y) - \mathsf{D}_\Phi(z \,\|\, x) = -\Phi(y) + \Phi(x) - \langle \nabla\Phi(y), z - y\rangle + \langle \nabla\Phi(x), z - x\rangle. \quad (4.13)$$

For a vector $c \in \mathbb{R}^n$ and $x \in \mathsf{K}$, define the projection

$$\Pi^c_{\mathsf{K}}(x) := \operatorname{argmin}\left\{\mathsf{D}_\Phi(y \,\|\, x) + \langle c, y\rangle : y \in \mathsf{K}\right\}.$$

Since K is compact and Φ is strictly convex, there is a unique minimizer $y^* \in \mathsf{K}$.

For $x \in \mathsf{K}$, recall the definition of the normal cone at x:

$$\mathsf{N}_{\mathsf{K}}(x) = \left\{p \in \mathbb{R}^n : \langle p, y - x\rangle \leq 0 \text{ for all } y \in \mathsf{K}\right\}.$$

Given a representation of K by inequality constraints, $\mathsf{K} = \{x \in \mathbb{R}^n : Ax \leq b\}$ for $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^n$, it holds

$$\mathsf{N}_{\mathsf{K}}(x) = \{A^T y : y \geq 0 \text{ and } y^T(Ax - b) = 0\}.$$

The KKT conditions yield

$$\nabla\Phi(y^*) = \nabla\Phi(x) - c - \lambda^*, \tag{4.14}$$

where $\lambda^* \in \mathsf{N}_{\mathsf{K}}(y^*)$. Since $\mathsf{N}_{\mathsf{K}}(y^*) = \mathsf{N}_{\mathsf{K}_0}(y^*) + \mathsf{N}_{\mathbb{R}^n_+}(y^*)$, we can can decompose $\lambda^* = \beta - \alpha$ with $\beta \in \mathsf{N}_{\mathsf{K}_0}(y^*)$ and $-\alpha \in \mathsf{N}_{\mathbb{R}^n_+}(y^*)$. In particular, we have $\alpha \geq 0$ and $\alpha_i > 0 \implies y^*_i = 0$ for every $i = 1, \ldots, n$.

Substituting this into (4.13) gives

$$\begin{aligned}\mathsf{D}_\Phi(z \,\|\, y^*) - \mathsf{D}_\Phi(z \,\|\, x) &= -\Phi(y^*) + \Phi(x) + \langle \nabla\Phi(x), y^* - x\rangle + \langle c - \alpha + \beta, z - y^*\rangle \\ &\leq -\mathsf{D}_\Phi(y^* \,\|\, x) + \langle c - \alpha, z - y^*\rangle,\end{aligned}$$

where the inequality comes from $\langle \beta, z - y^*\rangle \leq 0$ since $z \in \mathsf{K}$ and $\beta \in \mathsf{N}_{\mathsf{K}}(y^*)$. We have proved the following.

Lemma 4.2.1. *For any $x, z \in \mathsf{K}$, and $c \in \mathbb{R}^n$, let $y^* = \Pi^c_{\mathsf{K}}(x)$ and λ^* be as in (4.14). Then for any $\alpha \in -\mathsf{N}_{\mathbb{R}^+_n}(y^*)$ such that $\lambda^* + \alpha \in \mathsf{N}_{\mathsf{K}_0}(y^*)$, it holds that*

$$\mathsf{D}_\Phi(z \,\|\, y^*) - \mathsf{D}_\Phi(z \,\|\, x) \leq \langle c - \alpha, z - y^*\rangle.$$

4.2.1 Iterative Bregman projections

We describe now a discretization of the algorithm from the introduction. Fix a tree T and recall the definition of K_T from (4.3). Let Q_T denote the collection of vectors $q \in \mathbb{R}_+^{V\setminus\{\mathfrak{r}\}}$ such that for all $u \in V \setminus \mathcal{L}$,

$$\sum_{v\in\chi(u)} q_v = 1.$$

For $q \in Q_T$ and $u \in V \setminus \mathcal{L}$, we use $q^{(u)} \in \mathbb{R}_+^{\chi(u)}$ to denote the vector defined by $q_v^{(u)} := q_v$ for $v \in \chi(u)$, and define the corresponding probability simplex $Q_T^{(u)} := \{q^{(u)} : q \in Q_T\}$. We will use $\Delta : Q_T \to \mathsf{K}_T$ for the map which sends $q \in Q_T$ to the (unique) $x = \Delta(q) \in \mathsf{K}_T$ such that

$$x_v = x_u q_v \qquad \forall u \in V \setminus \mathcal{L}, v \in \chi(u).$$

Note that q contains more information than x; the map Δ fails to be invertible whenever there is some $u \in V \setminus \mathcal{L}$ with $x_u = 0$.

Fix $\kappa \geq 1$. On the open domain $\mathcal{D}^{(u)} = (-\min_{v \in \chi(u)} \delta_v, \infty)^{\chi(u)}$, for δ_v as given in (4.10), define the strictly convex function $\Phi^{(u)} : \mathcal{D}^{(u)} \to \mathbb{R}$ by

$$\Phi^{(u)}(p) := \frac{1}{\kappa} \sum_{v \in \chi(u)} \frac{w_v}{\eta_v} \left(p_v + \delta_v\right) \log \left(p_v + \delta_v\right).$$

Denote the corresponding Bregman divergence on $Q_T^{(u)}$ by

$$\mathsf{D}^{(u)}\big(p \,\|\, p'\big) = \frac{1}{\kappa} \sum_{v \in \chi(u)} \frac{w_v}{\eta_v} \left[\left(p_v + \delta_v\right) \log \frac{p_v + \delta_v}{p'_v + \delta_v} + p'_v - p_v \right].$$

We now define an algorithm that takes a point $q \in Q_T$ and a cost vector $c \in \mathbb{R}_+^{\mathcal{L}}$ and outputs a point $p = \mathcal{A}(q, c) \in Q_T$. Fix a topological ordering $u_1, u_2, \ldots, u_N$ of $V \setminus \mathcal{L}$ such that every child in T occurs before its parent. We define p inductively as follows. Let $\hat{c}_\ell := c_\ell$ for $\ell \in \mathcal{L}$. For every $j = 1, 2, \ldots, N$:

$$\hat{c}_v^{(u_j)} := \hat{c}_v \qquad \forall v \in \chi(u_j) \tag{4.15}$$

$$p^{(u_j)} := \operatorname{argmin} \left\{ \mathsf{D}^{(u_j)}\Big(p \,\|\, q^{(u_j)}\Big) + \left\langle p, \hat{c}^{(u_j)} \right\rangle \;\Big|\; p \in Q_T^{(u_j)} \right\} \tag{4.16}$$

$$\hat{c}_{u_j} := \sum_{v \in \chi(u_j)} p_v^{(u_j)} \hat{c}_v \tag{4.17}$$

Let $\alpha^{(u_j)}$ be the vector of Lagrange multipliers corresponding to the nonnegativity constraints in (4.16) (recall Lemma 4.2.1). One should note that in this setting (a probability simplex), the nonnegativity multipliers are unique and thus well-defined.

We denote $\alpha = \alpha^{q,c} \in \mathbb{R}_+^V$ as the vector given by $\alpha_v := \alpha_v^{(\mathsf{p}(v))}$ for $v \neq \mathbb{r}$ and $\alpha_{\mathbb{r}} := 0$. Recall the complementary slackness conditions:

$$\alpha_v > 0 \implies p_v = 0. \tag{4.18}$$

For $v \in \chi(u)$, calculate

$$\left(\nabla\Phi^{(u)}(p)\right)_v = \frac{1}{\kappa}\frac{w_v}{\eta_v}\left(1+\log(p_v+\delta_v)\right).$$

Then using (4.14), we can write the algorithm as follows:

For $j = 1, 2, \ldots, N$:
  For $v \in \chi(u_j)$:
    $p_v^{(u_j)} := (q_v^{(u_j)} + \delta_v)\exp\left(\kappa\frac{\eta_v}{w_v}\left(\beta_{u_j} - (\hat{c}_v - \alpha_v)\right)\right) - \delta_v,$
  $\hat{c}_{u_j} := \sum_{v\in\chi(u_j)} p_v^{(u_j)}\hat{c}_v.$

where $\beta_{u_j} \geq 0$ is the multiplier for the constraint $\sum_{v\in\chi(u_j)} q_v^{(u_j)} \geq 1$. There is no multiplier for the constraint $\sum_{v\in\chi(u_j)} q_v^{(u_j)} \leq 1$ because this constraint will be satisfied automatically and is therefore not needed in (4.16): If it were violated, decreasing some p_v with $p_v > q_v^{(u_j)}$ would yield a strictly better solution to the minimization problem (4.16).

4.2.2 The global divergence

For $z \in \mathsf{K}_T$ and $q \in Q_T$, we define the global divergence function

$$\tilde{\mathsf{D}}(z \,\|\, q) := \frac{1}{\kappa}\sum_{u\notin\mathcal{L}}\sum_{v\in\chi(u)}\frac{w_v}{\eta_v}\left[(z_v+\delta_v z_u)\log\left(\frac{\frac{z_v}{z_u}+\delta_v}{q_v+\delta_v}\right) + z_u q_v - z_v\right],$$

with the convention that $0\log\left(\frac{0}{0}+\delta_v\right) = \lim_{\epsilon\to 0}\epsilon\log\left(\frac{0}{\epsilon}+\delta_v\right) = 0$. This is the Bregman divergence associated to (4.7) (divided by κ) with $\frac{x_v}{x_u}$ replaced by q_v. We will use $\tilde{\mathsf{D}}$ as a potential function to prove (4.1). The next lemma shows that when the offline algorithm moves, the change in potential is bounded by $O(1/\kappa)$ times the offline movement cost.

Lemma 4.2.2. *It holds that for any $q \in Q_T$ and $z, z' \in \mathsf{K}_T$,*

$$\left|\tilde{\mathsf{D}}(z \,\|\, q) - \tilde{\mathsf{D}}(z' \,\|\, q)\right| \leq \frac{1}{\kappa}\left(2+\frac{4}{\tau}\right)\|z - z'\|_{\ell_1(w)}.$$

Proof. Consider a differentiable map $z\colon [0,1] \to \mathbb{R}^V_{++}$ such that $\sum_{v\in\chi(u)} z_v(t) \leq z_u(t)$ for each t and $u \notin \mathcal{L}$. It suffices to show that for each t and every fixed $q \in Q_T$,

$$\kappa\left|\partial_t\tilde{\mathsf{D}}(z(t) \,\|\, q)\right| \leq \left(2+\frac{4}{\tau}\right)\|z'(t)\|_{\ell_1(w)}.$$

Moreover, it suffices to address the case when there is at most one $u \in V$ with $z'_u(t) \neq 0$.

A direct calculation gives

$$\kappa \partial_t \tilde{\mathsf{D}}(z(t) \,\|\, q) = \frac{w_u}{\eta_u} z'_u(t) \log\left(\frac{z_u(t)/z_{\mathsf{p}(u)}(t) + \delta_u}{q_u + \delta_u}\right) + \sum_{v \in \chi(u)} \frac{w_v}{\eta_v}\left[\delta_v z'_u(t) \log\left(\frac{z_v(t)/z_u(t) + \delta_v}{q_v + \delta_v}\right) + z'_u(t)\left(q_v - \frac{z_v(t)}{z_u(t)}\right)\right]. \tag{4.19}$$

Let us now use definitions (4.9) and (4.10) to observe that

$$\frac{1}{\eta_v}\left|\log\frac{p_v + \delta_v}{q_v + \delta_v}\right| \leq \frac{1}{\eta_v}\log\frac{1 + \delta_v}{\delta_v} \leq 2.$$

Using this in (4.19) yields

$$\begin{aligned}\kappa\left|\partial_t \tilde{\mathsf{D}}(z(t) \,\|\, q)\right| &\leq w_u |z'_u(t)|\left(2 + \frac{1}{\tau}\sum_{v \in \chi(u)}\left(2\delta_v + \left|q_v - \frac{z_v(t)}{z_u(t)}\right|\right)\right) \\ &\leq w_u |z'_u(t)|\left(2 + \frac{4}{\tau}\right),\end{aligned}$$

where the last inequality uses $\sum_{v \in \chi(u)} \delta_v \leq \sum_{v \in \chi(u)} \theta_v \leq 1$ and $\sum_{v \in \chi(u)} z_v(t) \leq z_u(t)$. □

According to the following lemma, the change in potential due to movement of the online algorithm is bounded by the difference in service cost between the offline and online algorithm.

Lemma 4.2.3. *For any cost vector $c \in \mathbb{R}_+^{\mathcal{L}}$, $z \in \mathsf{K}_T$, and $q \in Q_T$, it holds that if $p = \mathcal{A}(q, c)$, then*

$$\tilde{\mathsf{D}}(z \,\|\, p) - \tilde{\mathsf{D}}(z \,\|\, q) \leq \left\langle c, z - \Delta(p)\right\rangle_{\mathcal{L}}.$$

Proof. Fix $q \in Q_T$ and $c \in \mathbb{R}_+^{\mathcal{L}}$. Let $\alpha = \alpha^{q,c}$ denote the vector of multipliers defined in Section 4.2.1. For $u \in V \setminus \mathcal{L}$ with $z_u > 0$, define $z^{(u)} \in Q_T^{(u)}$ by

$$z_v^{(u)} := \frac{z_v}{z_u}.$$

Then Lemma 4.2.1 gives

$$\mathsf{D}^{(u)}\left(z^{(u)} \,\|\, p^{(u)}\right) - \mathsf{D}^{(u)}\left(z^{(u)} \,\|\, q^{(u)}\right) \leq \left\langle \hat{c}^{(u)} - \alpha^{(u)}, z^{(u)} - p^{(u)}\right\rangle_{\chi(u)},$$

where we use $\langle \cdot, \cdot \rangle_{\chi(u)}$ for the standard inner product on $\mathbb{R}^{\chi(u)}$. Multiplying by z_u and summing yields

$$\begin{aligned}\tilde{\mathsf{D}}(z \,\|\, p) - \tilde{\mathsf{D}}(z \,\|\, q) &\le \sum_{u \notin \mathcal{L}} z_u \left\langle \hat{c}^{(u)} - \alpha^{(u)}, z^{(u)} - p^{(u)} \right\rangle_{\chi(u)} \\ &= \sum_{u \notin \mathcal{L}} \sum_{v \in \chi(u)} (\hat{c}_v^{(u)} - \alpha_v^{(u)}) z_v - \sum_{u \notin \mathcal{L}} z_u \sum_{v \in \chi(u)} (\hat{c}_v^{(u)} - \alpha_v^{(u)}) p_v \,.\end{aligned}$$

Note that from (4.18), the latter expression is

$$\sum_{u \notin \mathcal{L}} z_u \sum_{v \in \chi(u)} \hat{c}_v^{(u)} p_v \overset{(4.17)}{=} \sum_{u \notin \mathcal{L}} z_u \hat{c}_u .$$

Noting that $\hat{c}_{\mathbb{r}} = \sum_{\ell \in \mathcal{L}} \Delta(p)_\ell c_\ell$, this gives

$$\tilde{\mathsf{D}}(z \,\|\, p) - \tilde{\mathsf{D}}(z \,\|\, q) \le \sum_{u \neq \mathbb{r}} (\hat{c}_u - \alpha_u) z_u - \sum_{u \notin \mathcal{L}} z_u \hat{c}_u \le \left\langle c, z - \Delta(p) \right\rangle_{\mathcal{L}} . \qquad \square$$

4.2.3 Algorithm and competitive analysis

For the proof of bound (4.2), we employ two potential functions ψ and Ψ, defined as follows. For $x \in \mathsf{K}_T$, let $\psi(x) := \sum_{u \neq \mathbb{r}} w_u x_u$. For $q \in Q_T$, let

$$\begin{aligned}\Psi_u(q) &:= -\Delta(q)_u \, \mathsf{D}^{(u)}\left(\theta^{(u)} \,\|\, q^{(u)}\right) \\ \Psi(q) &:= \sum_{u \notin \mathcal{L}} \Psi_u(q).\end{aligned}$$

The next lemma justifies that when the algorithm moves from x to y, it suffices to bound the positive movement cost $\left\|(x-y)_+\right\|_{\ell_1(w)}$ rather than the actual movement cost $\|x - y\|_{\ell_1(w)}$. Its proof is straightforward.

Lemma 4.2.4. *For $x, y \in \mathsf{K}_T$ it holds that*

$$\|x - y\|_{\ell_1(w)} = 2\left\|(x-y)_+\right\|_{\ell_1(w)} + [\psi(y) - \psi(x)].$$

In the next section, we will prove the following.

Lemma 4.2.5 (Movement analysis)**.** *It holds that*

$$\frac{\tau - 3}{\kappa \tau} \left\|\left(x - y\right)_+\right\|_{\ell_1(w)} \le (2\mathfrak{D}_T + \log n)\langle c, x \rangle_{\mathcal{L}} + \left[\Psi(q) - \Psi(p)\right].$$

Define $w_{\min} := \min\{w_\ell : \ell \in \mathcal{L}\}$ and

$$\epsilon_T := \frac{w_{\min}}{2(2\mathfrak{D}_T + \log n)} \frac{\tau - 3}{\tau \kappa}.$$

Theorem 4.2.6. *Consider any $q \in Q_T$ and $c \in \mathbb{R}_+^{\mathcal{L}}$. If we define $p = \mathcal{A}(q, c)$, $x = \Delta(q)$, $y = \Delta(p)$, then for any $z \in \mathsf{K}_T$:*

$$\langle c, y\rangle_{\mathcal{L}} \le \langle c, z\rangle_{\mathcal{L}} + \left[\tilde{\mathsf{D}}(z \,\|\, q) - \tilde{\mathsf{D}}(z \,\|\, p)\right] \tag{4.20}$$

$$\kappa^{-1} \left\|x - y\right\|_{\ell_1(w)} \le [\psi(y) - \psi(x)] + \frac{2\tau}{\tau - 3} \left([\Psi(q) - \Psi(p)] + (2\mathfrak{D}_T + \log n)\langle c, x\rangle_{\mathcal{L}}\right) \tag{4.21}$$

Moreover, if $\|c\|_\infty \le \epsilon_T$, then

$$\kappa^{-1} \left\|x - y\right\|_{\ell_1(w)} \le [\psi(y) - \psi(x)] + \frac{4\tau}{\tau - 3} \left([\Psi(q) - \Psi(p)] + (2\mathfrak{D}_T + \log n)\langle c, y\rangle_{\mathcal{L}}\right). \tag{4.22}$$

Proof. The bound (4.20) follows from Lemma 4.2.3, and (4.21) follows from Lemma 4.2.5 and Lemma 4.2.4. To see that (4.22) follows from (4.21) and Lemma 4.2.5, use the fact that

$$\langle c, x\rangle_{\mathcal{L}} \le \langle c, y\rangle_{\mathcal{L}} + \frac{\|c\|_\infty}{w_{\min}} \left\|(x - y)_+\right\|_{\ell_1(w)}. \qquad \square$$

In light of Theorem 4.2.6, we can respond to a cost function $c \in \mathbb{R}_+^{\mathcal{L}}$ by splitting it into M pieces $c_1, c_2, \ldots, c_M$ where $M = \lceil \|c\|_\infty / \epsilon_T \rceil$. Now define $q_i := \mathcal{A}(q_{i-1}, c/M)$, $q_0 := q$ and $\bar{\mathcal{A}}(q, c) := q_M$.

Theorem 4.2.7. *Fix $\tau \ge 4$. Consider the algorithm that begins in some configuration $q_0 \in Q_T$. If $c_t \in \mathbb{R}_+^{\mathcal{L}}$ is the cost function that arrives at time t, denote $q_t := \bar{\mathcal{A}}(q_{t-1}, c_t)$. Then the online algorithm that moves to $\Delta(q_t)$ at time t is $(1, O(1/\kappa), O(\kappa(\mathfrak{D}_T + \log n)))$-finely competitive.*

We prove this momentarily. The following fact is well-known and, in conjunction with the preceding theorem, yields the validity of Theorem 4.1.1 and Theorem 4.1.3.

Lemma 4.2.8. *If $(\mathcal{L}, d_T)$ is an HST metric, then there is another weighted tree T' with leaf set $\mathcal{L}$ such that*

(a) $(\mathcal{L}, d_{T'})$ *is a 4-HST metric.*

(b) $\mathfrak{D}_{T'} \le \log_2 |\mathcal{L}|$

(c) All the leaves of T' *have depth* $\mathfrak{D}_{T'}$*.*

(d) $d_T(\ell, \ell') \le d_{T'}(\ell, \ell') \le O(d_T(\ell, \ell'))$ *for all* $\ell, \ell' \in \mathcal{L}$*.*

Proof sketch. Replace every weight w_v in T with $\hat{w}_v := 4^{\lceil \log_4 w_v \rceil}$ and iteratively contract every edge $(\mathsf{p}(u), u)$ with $\hat{w}_{\mathsf{p}(u)} = \hat{w}_u$ and $u \notin \mathcal{L}$. The resulting weighted tree T_1 is a 4-HST by construction.

Now iteratively contract every edge $(\mathsf{p}(u), u)$ in T_1 for which $|\mathcal{L}_u^{T_1}| > \frac{1}{2}|\mathcal{L}_{\mathsf{p}(u)}^{T_1}|$. The resulting tree T' has depth $\mathfrak{D}_{T'} \le \log_2 |\mathcal{L}|$. Finally, one can achieve property (3) by increasing the depth of every root-leaf path to $\mathfrak{D}_{T'}$ using vertex weights that decrease by a factor of 4 along the path. □

Proof of Theorem 4.2.7. Consider a sequence $(c_1, \ldots, c_m)$ of cost functions. By splitting the costs into smaller pieces, we may assume that $\|c_t\|_\infty \le \epsilon_T$ for all t.

Let $z_0^*, \ldots, z_m^*$ denote the states of some offline algorithm with $z_0^* = \Delta(q_0)$, and let $x_0, \ldots, x_m$ denote the states of our online algorithm, with $x_t = \Delta(q_t)$. Then using $\tilde{\mathsf{D}}(z_0^* \,\|\, x_0) = 0$ along with (4.20) and Lemma 4.2.2 yields

$$\begin{aligned}\sum_{t=1}^m \langle c_t, x_t \rangle_{\mathcal{L}} &\le \sum_{t=1}^m \langle c_t, z_t^* \rangle_{\mathcal{L}} - \tilde{\mathsf{D}}(z_m^* \,\|\, q_m) + O(1/\kappa) \sum_{t=1}^m \|z_t^* - z_{t-1}^*\|_{\ell_1(w)} \\ &\le \sum_{t=1}^m \langle c_t, z_t^* \rangle_{\mathcal{L}} + O(1/\kappa) \sum_{t=1}^m \|z_t^* - z_{t-1}^*\|_{\ell_1(w)},\end{aligned}$$

where we have used $\tilde{\mathsf{D}}(z \,\|\, q) \ge 0$ for all $z \in \mathsf{K}_T$ and $q \in Q_T$. This verifies (4.1) with $\alpha_0 = 1$, $\alpha_1 = O(1/\kappa)$, and $\beta = 0$. Moreover, (4.22) gives

$$\frac{1}{\kappa} \sum_{t=1}^m \|x_t - x_{t-1}\|_{\ell_1(w)} \le$$

$$\big[\psi(x_m) - \psi(x_0)\big] + \frac{4\tau}{\tau - 3} \big[\Psi(q_0) - \Psi(q_m)\big] + (2\mathfrak{D}_T + \log n) \sum_{t=1}^m \langle c_t, x_t \rangle_{\mathcal{L}},$$

verifying (4.2) with $\alpha_1 \le O(\kappa(\mathfrak{D}_T + \log n))$ and $\beta' \le O(\kappa \max_{v \ne \mathbb{r}} w_v)$ (see Lemma 4.2.10 below). □

4.2.4 Movement analysis

It remains to prove Lemma 4.2.5. The KKT conditions (cf. (4.14)) give: For every $v \in \chi(u)$,

$$\frac{1}{\kappa}\frac{w_v}{\eta_v}\log\left(\frac{p_v+\delta_v}{q_v+\delta_v}\right) = \beta_u - \hat{c}_v + \alpha_v\,, \tag{4.23}$$

where $\beta_u \geq 0$ is the multiplier corresponding to the constraint $\sum_{v\in\chi(u)} q_v \geq 1$.

Lemma 4.2.9. *It holds that $\alpha_v \leq \hat{c}_v$ for all $v \in V \setminus \{\mathbb{r}\}$.*

Proof. Note that $\hat{c}_v \geq 0$ by construction. Thus if $\alpha_v = 0$, we are done. Otherwise, by complementary slackness, it must be that $p_v = 0$, and therefore $\log\left(\frac{p_v+\delta_v}{q_v+\delta_v}\right) \leq 0$. Since $\beta_{\mathsf{p}(v)} \geq 0$, (4.23) implies that $\alpha_v \leq \hat{c}_v$. □

Define $\sigma_v := \log\left(\frac{p_v+\delta_v}{q_v+\delta_v}\right)$ so that

$$q_v - p_v = (q_v+\delta_v)(1-e^{\sigma_v}). \tag{4.24}$$

Recall that for $v \in \chi(u)$, we have $x_v = q_v x_u$ and $y_v = p_v y_u$, thus

$$x_v - y_v = x_u(q_v - p_v) + p_v(x_u - y_u) = (x_v + \delta_v x_u)(1-e^{\sigma_v}) + p_v(x_u - y_u).$$

In particular,

$$\begin{aligned} w_v\left(x_v - y_v\right)_+ &\leq w_v(x_v+\delta_v x_u)(1-e^{\sigma_v})_+ + w_v p_v\left(x_u - y_u\right)_+ \\ &\leq w_v(x_v+\delta_v x_u)(1-e^{\sigma_v})_+ + \frac{w_u}{\tau}p_v\left(x_u - y_u\right)_+ . \end{aligned}$$

Using $\sum_{v\in\chi(u)} p_v = 1$ and summing over all vertices yields

$$\sum_{v\neq\mathbb{r}} w_v\left(x_v - y_v\right)_+ \leq \sum_{v\neq\mathbb{r}} w_v(x_v+\delta_v x_{\mathsf{p}(v)})(1-e^{\sigma_v})_+ + \frac{1}{\tau}\sum_{v\neq\mathbb{r}} w_v\left(x_v - y_v\right)_+ ,$$

hence

$$\begin{aligned} \sum_{v\neq\mathbb{r}} w_v\left(x_v - y_v\right)_+ &\leq \frac{\tau}{\tau-1}\sum_{v\neq\mathbb{r}} w_v(x_v+\delta_v x_{\mathsf{p}(v)})(1-e^{\sigma_v})_+ \\ &\leq \frac{\tau}{\tau-1}\sum_{v\neq\mathbb{r}} w_v(x_v+\delta_v x_{\mathsf{p}(v)})\,(\sigma_v)_- \\ &\leq \frac{\kappa\tau}{\tau-1}\left(\sum_{v\neq\mathbb{r}}\eta_v x_v\hat{c}_v + \sum_{u\notin\mathcal{L}} x_u \sum_{v\in\chi(u)}\theta_v(\hat{c}_v - \alpha_v)\right), \end{aligned} \tag{4.25}$$

where the last line uses Lemma 4.2.9 and (4.23), to bound $w_v(\sigma_v)_- \leq \kappa\eta_v(\hat{c}_v - \alpha_v)$.

Note that

$$\sum_{v\neq \mathbb{r}} \eta_v x_v \hat{c}_v \leq \sum_{\ell\in\mathcal{L}} c_\ell x_\ell \sum_{v\in\gamma_{\mathbb{r},\ell}\setminus\{\mathbb{r}\}} \eta_v \leq (\mathfrak{D}_T + \log n)\langle c, x\rangle, \tag{4.26}$$

since for any $\ell \in \mathcal{L}$, it holds that

$$\sum_{v\in\gamma_{\mathbb{r},\ell}\setminus\{\mathbb{r}\}} \eta_v = \mathfrak{D}_T(\ell) + \sum_{v\in\gamma_{\mathbb{r},\ell}\setminus\{\mathbb{r}\}} \log\frac{|\mathcal{L}_{\mathsf{p}(v)}|}{|\mathcal{L}_v|} = \mathfrak{D}_T(\ell) + \log n,$$

where $\mathfrak{D}_T(\ell)$ is the combinatorial depth of ℓ.

The second sum in (4.25) can be interpreted as the service cost of hybrid configurations of q and θ: While $\sum_{v\in\chi(u)} x_v\hat{c}_v$ is the service cost of x in $\mathcal{L}_u$, the term $x_u\sum_{v\in\chi(u)}\theta_v\hat{c}_v$ is the service cost in $\mathcal{L}_u$ of the modification of x whose conditional probabilities at the children of u are given by $\theta^{(u)}$ rather than $q^{(u)}$. To bound this hybrid service cost, we will employ the auxiliary potential Ψ.

The hybrid cost

We require the following elementary estimate.

Lemma 4.2.10. *For $u \notin \mathcal{L}$ it holds that*

$$\max\left\{\mathsf{D}^{(u)}(r\,\|\,p) : r, p \in Q_T^{(u)}\right\} \leq \frac{2}{\kappa}\frac{w_u}{\tau}.$$

Proof. Define $\phi_v : (-\delta_v, \infty) \to \mathbb{R}$ by

$$\phi_v(p) := \frac{1}{\eta_v}(p_v + \delta_v)\log(p_v + \delta_v),$$

and let

$$\mathsf{D}_{\phi_v}(q_v\,\|\,p_v) = \frac{1}{\eta_v}\left[(q_v+\delta_v)\log\frac{q_v+\delta_v}{p_v+\delta_v} + (p_v - q_v)\right]$$

denote the corresponding Bregman divergence. Then for $q_v, p_v \geq 0$, it holds that $\mathsf{D}_{\phi_v}(q_v\,\|\,p_v) \geq 0$ since ϕ_v is convex on $\mathbb{R}_+$. Employing the τ-HST property of T, this implies that

$$\mathsf{D}^{(u)}(r\,\|\,p) = \frac{1}{\kappa}\sum_{v\in\chi(u)} w_v \mathsf{D}_{\phi_v}(r_v\,\|\,p_v) \leq \frac{w_u}{\kappa\tau}\sum_{v\in\chi(u)}\mathsf{D}_{\phi_v}(r_v\,\|\,p_v).$$

Define $F : Q_T^{(u)} \times Q_T^{(u)} \to \mathbb{R}_+$ by $F(r,p) := \sum_{v\in\chi(u)} \mathsf{D}_{\phi_v}(r_v \,\|\, p_v)$. The map $r \mapsto F(r,p)$ is convex in general (for any Bregman divergence). The map $p \mapsto F(r,p)$ is convex as well, as this holds for each map $p_v \mapsto \mathsf{D}_{\phi_v}(q_v \,\|\, p_v)$ since $-\log(x)$ is convex on $\mathbb{R}_{++}$. Since the maximum of a convex function on the a polytope is achieved at an extreme point, we have

$$\begin{aligned}
&\max\left\{F(r,p) : r,p \in Q_T^{(u)}\right\} \\
&\leq \max_{\substack{v,v'\in\chi(u)\\ v\neq v'}} \left[\frac{1}{\eta_v}\left((1+\delta_v)\log\frac{1+\delta_v}{\delta_v} - 1\right) + \frac{1}{\eta_{v'}}\left(\delta_{v'}\log\frac{\delta_{v'}}{1+\delta_{v'}} + 1\right)\right] \\
&\leq 2. \qquad \square
\end{aligned}$$

The next lemma is crucial: It relates the service cost (with respect to the reduced cost $\hat{c} - \alpha$) of the hybrid configurations to the service cost of the actual configuration and the movement cost.

Lemma 4.2.11. *For any $u \notin \mathcal{L}$, it holds that*

$$\Psi_u(p) - \Psi_u(q) \leq \frac{2}{\kappa}\frac{w_u}{\tau}\left(x_u - y_u\right)_+ + \sum_{v\in\chi(u)} (\hat{c}_v - \alpha_v)\left[x_v - \theta_v x_u\right]. \tag{4.27}$$

Proof. Write

$$\begin{aligned}
&\Psi_u(p) - \Psi_u(q) \\
&\quad= x_u \mathsf{D}^{(u)}\left(\theta^{(u)} \,\|\, q^{(u)}\right) - y_u \mathsf{D}^{(u)}\left(\theta^{(u)} \,\|\, p^{(u)}\right) \\
&\quad= (x_u - y_u)\mathsf{D}^{(u)}(\theta^{(u)} \,\|\, p^{(u)}) + x_u\left[\mathsf{D}^{(u)}(\theta^{(u)} \,\|\, q^{(u)}) - \mathsf{D}^{(u)}(\theta^{(u)} \,\|\, p^{(u)})\right].
\end{aligned}$$

Using Lemma 4.2.10, the first term is bounded by $\frac{2}{\kappa}\frac{w_u}{\tau}(x_u - y_u)_+$.

Let us now bound the second term. Using $1 + t \leq e^t$, we have

$$\begin{aligned}
&\kappa x_u\left[\mathsf{D}^{(u)}(\theta^{(u)} \,\|\, q^{(u)}) - \mathsf{D}^{(u)}(\theta^{(u)} \,\|\, p^{(u)})\right] \\
&\quad= x_u \sum_{v\in\chi(u)} \frac{w_v}{\eta_v}\left[(\theta_v + \delta_v)\log\frac{p_v + \delta_v}{q_v + \delta_v} + q_v - p_v\right] \\
&\quad\overset{(4.24)}{=} x_u \sum_{v\in\chi(u)} \frac{w_v}{\eta_v}\left[(\theta_v + \delta_v)\sigma_v + (q_v + \delta_v)(1 - e^{\sigma_v})\right] \\
&\quad\leq x_u \sum_{v\in\chi(u)} \frac{w_v}{\eta_v}\sigma_v(\theta_v - q_v) \\
&\quad= \sum_{v\in\chi(u)} \frac{w_v}{\eta_v}\sigma_v\left[\theta_v x_u - x_v\right].
\end{aligned}$$

To finish the proof, observe that from (4.23),

$$\sum_{v\in\chi(u)} \frac{w_v}{\eta_v}\sigma_v\left[\theta_v x_u - x_v\right] = \kappa \sum_{v\in\chi(u)} (\beta_u - \hat{c}_v + \alpha_v)\left[\theta_v x_u - x_v\right]$$
$$= \kappa \sum_{v\in\chi(u)} (\alpha_v - \hat{c}_v)\left[\theta_v x_u - x_v\right],$$

where the last equality uses $\sum_{v\in\chi(u)} x_v = x_u$ and $\sum_{v\in\chi(u)} \theta_v = 1$ (from (4.8)). □

Using the lemma gives

$$\sum_{u\notin\mathcal{L}} x_u \sum_{v\in\chi(u)} \theta_v(\hat{c}_v - \alpha_v)$$
$$\overset{(4.27)}{\leq} [\Psi(q) - \Psi(p)] + \frac{2}{\kappa\tau}\left\|\left(\Delta(q) - \Delta(p)\right)_+\right\|_{\ell_1(w)} + \sum_{v\neq\mathbb{r}} \hat{c}_v x_v$$
$$\leq [\Psi(q) - \Psi(p)] + \frac{2}{\kappa\tau}\left\|\left(\Delta(q) - \Delta(p)\right)_+\right\|_{\ell_1(w)} + \mathfrak{D}_T\langle c, x\rangle_{\mathcal{L}}.$$

Combining this inequality with (4.25) and (4.26) gives

$$\kappa^{-1}\left\|(x-y)_+\right\|_{\ell_1(w)}$$
$$\leq \frac{\tau}{\tau-1}\left[(2\mathfrak{D}_T + \log n)\langle c, x\rangle_{\mathcal{L}} + \left(\Psi(q) - \Psi(p)\right) + \frac{2}{\kappa\tau}\left\|(x-y)_+\right\|_{\ell_1(w)}\right],$$

completing the verification of Lemma 4.2.5.

4.3 Derivation of the dynamics and derived costs

For the sake of motivating the dynamics (4.11), we review the continuous-time mirror descent framework of [BCL+18]. Suppose that $\mathsf{K} \subseteq \mathbb{R}^N$ is a convex set. We recall again the definition of the *normal cone to* K *at* $x \in \mathsf{K}$, which is given by

$$N_{\mathsf{K}}(x) := (\mathsf{K} - x)^\circ = \left\{p \in \mathbb{R}^N : \left\langle p, y - x\right\rangle \leq 0 \text{ for all } y \in \mathsf{K}\right\}.$$

Suppose additionally that $\Phi : \mathcal{D} \to \mathbb{R}$ is C^2 and strictly convex on an open neighborhood $\mathcal{D} \supseteq \mathsf{K}$ so that the Hessian $\nabla^2\Phi(x)$ is well-defined and positive definite on $\mathcal{D}$. Given a control function $F : [0,\infty) \times \mathsf{K} \to \mathbb{R}^N$ and an initial point

$x_0 \in \mathsf{K}$, we will be concerned with absolutely continuous solutions $x : [0,\infty) \to \mathsf{K}$ to the differential inclusion

$$x(0) = x_0,$$
$$\nabla^2\Phi(x(t))x'(t) \in F(t,x(t)) - N_{\mathsf{K}}(x(t))\,.$$

In other words, a trajectory that satisfies $x(0) = x_0$ and for almost every $t \geq 0$,

$$x'(t) = \nabla^2\Phi(x(t))^{-1}\left(F(t,x(t)) - \gamma(t)\right) \tag{4.28}$$

with $\gamma(t) \in N_{\mathsf{K}}(x(t))$.

Under suitably strong conditions on Φ and F, there is a unique absolutely continuous solution to (4.28) [BCL+18]. In our setup, these conditions are actually *not satisfied* unless we prevent the path x from hitting the relative boundary of K. Nevertheless, the formal calculation is elucidating and motivates the algorithm of Section 4.2. For simplicity, we assume $\kappa := 1$ in this section.

4.3.1 Hessian computation

Let us take Φ as in (4.7) and calculate $\nabla^2\Phi(x)$ for $x \in \mathbb{R}^V_{++}$. Fix $u \neq \mathbb{r}$. Then we have

$$\partial_u\Phi(x) = \frac{w_u}{\eta_u}\left(\log\left(\frac{x_u}{x_{\mathsf{p}(u)}} + \delta_u\right) + 1\right) + \sum_{v\in\chi(u)}\frac{w_v}{\eta_v}\left(\delta_v \log\left(\frac{x_v}{x_u} + \delta_v\right) - \frac{x_v}{x_u}\right). \tag{4.29}$$

Moreover, $\partial_{uv}\Phi(x) = 0$ unless $u = v$, $u \in \chi(v)$, or $v \in \chi(u)$, and in this case,

$$\partial_{uu}\Phi(x) = \frac{w_u}{\eta_u(x_u + \delta_u x_{\mathsf{p}(u)})} + \sum_{v\in\chi(u)}\left(\frac{x_v}{x_u}\right)^2\frac{w_v}{\eta_v(x_v + \delta_v x_u)}$$
$$\partial_{u,\mathsf{p}(u)}\Phi(x) = \partial_{\mathsf{p}(u),u}\Phi(x) = -\frac{x_u}{x_{\mathsf{p}(u)}}\frac{w_u}{\eta_u(x_u + \delta_u x_{\mathsf{p}(u)})}.$$

4.3.2 Explicit dynamics

We are now in a position to calculate the formal dynamics. Let us define the control by $F(\cdot,t) := -c(t)$. We claim that for $u \neq \mathbb{r}$,

$$\partial_t\left(\frac{x_u(t)}{x_{\mathsf{p}(u)}(t)}\right) = \frac{\eta_u}{w_u}\left(\frac{x_u(t)}{x_{\mathsf{p}(u)}(t)} + \delta_u\right)\left(\beta_{\mathsf{p}(u)}(t) - \sum_{\ell\in\mathcal{L}_u}\frac{x_\ell(t)}{x_u(t)}c_\ell\right), \tag{4.30}$$

where $\beta_u(t) \geq 0$ denotes the Lagrange multiplier corresponding to the constraint $x_u = \sum_{v \in \chi(u)} x_v$.

To verify (4.30), let us define, for $u \neq \mathbb{r}$,

$$\mathcal{E}(u) := \frac{w_u}{\eta_u} \frac{x_{\mathsf{p}(u)}(t)}{x_u(t) + \delta_u x_{\mathsf{p}(u)}(t)} \partial_t \left(\frac{x_u(t)}{x_{\mathsf{p}(u)}(t)} \right).$$

Then (4.30) is equivalent to the assertion that

$$\mathcal{E}(u) = \beta_{\mathsf{p}(u)}(t) - \sum_{\ell \in \mathcal{L}_u} \frac{x_\ell(t)}{x_u(t)} c_\ell(t). \tag{4.31}$$

Recalling (4.28), the equality $\left(\nabla^2 \Phi(x(t)) x'(t)\right)_u = \left(F(t, x(t)) - \gamma(t)\right)_u$ is equivalent to

$$\mathcal{E}(\ell) = \beta_{\mathsf{p}(\ell)}(t) - c_\ell(t), \qquad \ell \in \mathcal{L}, \tag{4.32}$$

$$\mathcal{E}(u) - \sum_{v \in \chi(u)} \frac{x_v(t)}{x_u(t)} \mathcal{E}(v) = \beta_{\mathsf{p}(u)}(t) - \beta_u(t), \qquad u \in V \setminus (\mathcal{L} \cup \{\mathbb{r}\}). \tag{4.33}$$

Clearly (4.32) already confirms (4.31) for $\ell \in \mathcal{L}$.

Let us conclude by verifying (4.31) for all $u \notin \mathbb{r}$ by (reverse) induction on the depth. Employing (4.33) along with the validity of (4.31) for $\{\mathcal{E}(v) : v \in \chi(u)\}$ yields

$$\begin{aligned} \mathcal{E}(u) &= \beta_{\mathsf{p}(u)}(t) - \beta_u(t) + \sum_{v \in \chi(u)} \frac{x_v(t)}{x_u(t)} \left(\beta_u(t) - \sum_{\ell \in \mathcal{L}_u} \frac{x_\ell(t)}{x_u(t)} c_\ell(t) \right) \\ &= \beta_{\mathsf{p}(u)}(t) - \sum_{\ell \in \mathcal{L}_u} \frac{x_\ell(t)}{x_u(t)} c_\ell(t), \end{aligned}$$

where we used the fact that $x_u = \sum_{v \in \chi(u)} x_v$ for $x \in \mathsf{K}_T$.

4.3.3 Relationship between discrete and continuous dynamics

Recall the setup from Section 4.1.3. We consider a system of variables $\{q_u(t) : u \in V \setminus \{\mathbb{r}\}\}$ satisfying the differential equations

$$\partial_t q_u(t) = \frac{\eta_u}{w_u} \left(q_u(t) + \delta_u \right) \left(\beta_{\mathsf{p}(u)}(t) - \hat{c}_u(t) + \alpha_u(t) \right), \tag{4.34}$$

where $\alpha_u(t)$ is a Lagrangian multiplier for the constraint $q_u(t) \geq 0$, and $\hat{c}_u(t)$ is the "derived" cost in the subtree rooted at u:

$$\hat{c}_u(t) := \sum_{\ell \in \mathcal{L}_u} q_{\ell|u}(t) c_\ell(t)$$
$$q_{\ell|u}(t) := \prod_{v \in \gamma_{u,\ell} \setminus \{u\}} q_v(t)\,,$$

where $\gamma_{u,\ell}$ is the unique simple u-ℓ path in T. Now the values $q_{\ell|\mathbb{r}}$ give a probability distribution on the leaves.

Let us argue that when the discretization parameter of the algorithm presented in Section 4.2 goes to zero, one arrives at a solution to (4.34). Recall that in Section 4.2.3, we split each cost function $c \in \mathbb{R}_+^{\mathcal{L}}$ into M pieces $M^{-1}c$ and computed a sequence of configurations $q_0, \ldots, q_M \in Q_T$. Define the piecewise-linear function $q_{(M)} : [0,1] \to Q_T$ by

$$q_{(M)}\left(\frac{j+\delta}{M}\right) := (1-\delta)q_j + \delta q_{j+1}, \qquad \delta \in [0,1], j \in \{0, \ldots, M-1\}.$$

Recalling Section 4.2.1, we have

$$q_j^{(u)} := \operatorname{argmin}\left\{ \mathsf{D}^{(u)}\left(p \,\|\, q_{j-1}^{(u)}\right) + \left\langle p, M^{-1}\hat{c}_j^{(u)} \right\rangle \;\middle|\; p \in Q_T^{(u)} \right\}, \tag{4.35}$$

where

$$\hat{c}_j^{(u)} = \sum_{\ell \in \mathcal{L}_v} (q_j)_{\ell|u} c_\ell.$$

Thus for $v \in \chi(u)$ and $j \geq 1$,

$$\left(q_j^{(u)}\right)_v = \left[\left(q_{j-1}^{(u)}\right)_v + \delta_v\right] \exp\left(\frac{\eta_v}{w_v}\left(\beta_u - (M^{-1}(\hat{c}_j^{(u)})_v - \alpha_v)\right)\right) - \delta_v.$$

One can now verify that there is a constant $L = L(c,T)$ such that

$$\left|(q_j)_v - (q_{j-1})_v\right| \leq \frac{L}{M}, \quad j \in \{1, \ldots, M\}, v \in V \setminus \{\mathbb{r}\}.$$

In particular, we see that $q'_{(M)} \in L^\infty([0,1], \mathbb{R}^{V \setminus \{\mathbb{r}\}})$ for every $M \geq 1$ and, moreover,

$$\sup_{M \geq 1} \left\| q'_{(M)} \right\|_{L^\infty} < \infty. \tag{4.36}$$

Therefore by Arzelà-Ascoli, there is a subsequence $\{M_k\}$ such that $q_{(M_k)}$ converges uniformly to a function $q : [0,1] \to Q_T$.

Since the unit ball of $L^\infty([0,1], \mathbb{R}^{V\setminus\{\mathfrak{r}\}})$ is weakly compact (by the sequential Banach-Alaoglu Theorem), we can pass to a further subsequence $\{M'_k\}$ along which $q'_{(M'_k)}$ converges weakly to some $h \in L^\infty([0,1], \mathbb{R}^{V\setminus\{\mathfrak{r}\}})$. Moreover, since $q_{(M)}(b) - q_{(M)}(a) = \int_a^b q'_{(M)}(t)\,dt$ for all $0 \le a < b \le 1$, it follows that $q(b) - q(a) = \int_a^b h(t)\,dt$ as well, and therefore for almost all $t \in [0,1]$, we have $q'(t) = h(t)$.

If we similarly linearly interpolate the cost function to $\hat{c}_{(M)} : [0,1] \to \mathbb{R}_+^{V\setminus\{\mathfrak{r}\}}$, then $\hat{c}_{(M_k)} \to \hat{c}$ along this sequence as well, and

$$\hat{c}^{(u)}(t) = \sum_{\ell \in \mathcal{L}_v} q_{\ell|u}(t) c_\ell.$$

Now the KKT conditions for optimality in (4.35) give

$$\nabla\Phi^{(u)}\left(q_j^{(u)}\right) - \nabla\Phi^{(u)}\left(q_{j-1}^{(u)}\right) + M^{-1}\hat{c}_j^{(u)} \in -\mathsf{N}_{Q_T^{(u)}}\left(q_j^{(u)}\right),$$

or equivalently,

$$\frac{\nabla\Phi^{(u)}\left(q_j^{(u)}\right) - \nabla\Phi^{(u)}\left(q_{j-1}^{(u)}\right)}{M^{-1}} \in -\hat{c}_j^{(u)} - \mathsf{N}_{Q_T^{(u)}}\left(q_j^{(u)}\right).$$

By standard results in differential inclusion theory (e.g., the Convergence Theorem [AC84, Thm. 1.4.1]), we conclude that $q : [0,1] \to Q_T$ solves the differential inclusion

$$\nabla^2\Phi^{(u)}\left(q^{(u)}(t)\right)\partial_t q^{(u)}(t) \in -\hat{c}^{(u)}(t) - \mathsf{N}_{Q_T^{(u)}}(q^{(u)}(t)).$$

Calculating the Hessian $\nabla^2\Phi^{(u)}$ reveals that $q(t)$ is a solution to (4.34).

5
Final Remarks

We studied the ∞-server problem, the k-taxi problem and metrical task systems in the framework of competitive analysis.

For the ∞-server problem, our main result was a tight connection with the k-server problem against weak adversaries. This connection seems particularly promising for improving lower bounds for the latter problem. Potentially, our ideas can be extended to yield an infinite lower bound, which would refute the hypothesis that having sufficiently many more online than offline servers would allow for a constant competitive ratio.

Our main result for the k-taxi problem is a competitive algorithm for any finite metric. It remains open to find an algorithm whose competitive ratio is independent of the size of the metric space. Our algorithm for three taxis on the line might be the basis for such an algorithm. The mirror descent technique used in Chapter 4 may be a way to obtain subexponentially competitive randomized algorithms for the k-taxi problem. Indeed, it is possible to adapt the mirror descent k-server algorithm from [BCL$^+$18] to obtain an $O(\log k)$-competitive algorithm for the k-taxi problem on weighted stars. However, the relocation requests for the k-taxi problem pose a significant challenge in extending this approach to HSTs. It would also seem surprising if the randomized competitive ratio on HSTs were (poly)logarithmic rather than polynomial: At least for deterministic algorithms,

HSTs are exponentially more difficult for the k-taxi problem than weighted stars. Moreover, the reduction from layered graph traversal to the k-taxi problem for *deterministic* algorithms and the polynomial lower bound for *randomized* layered graph traversal algorithms are some indication that a similar polynomial lower bound may also hold for randomized k-taxi algorithms on general metrics.

For metrical task systems, we established a pure regularization approach that yields optimal refined guarantees on HSTs. This poses the question of how to measure uncertainty correctly on general metrics, and whether the regularization approach can be extended to obtain the optimal competitive ratio on general metrics. Another key open problem is whether one can replace the entropy regularizer for the k-server problem from [BCL$^+$18] with a conditional variant in order to improve the competitive ratio on HSTs from $O((\log k)^2)$ to $O(\log k)$, similarly to how replacing the unconditional entropy in [BCLL19] with the conditional version of Chapter 4 improves the competitive ratio of the associated MTS-algorithm from $O((\log n)^2)$ to $O(\log n)$.

Bibliography

[ABBS10] Jacob Abernethy, Peter Bartlett, Niv Buchbinder, and Isabelle Stanton. A regularization approach to metrical task systems. In *Algorithmic Learning Theory*, ALT 2010. Springer, 2010.

[AC84] Jean Pierre Aubin and A. Cellina. *Differential Inclusions: Set-Valued Maps and Viability Theory*. Springer-Verlag New York, Inc., 1984.

[AGGT19] CJ Argue, Anupam Gupta, Guru Guruganesh, and Ziye Tang. Chasing convex bodies with linear competitive ratio. *arXiv:1905.11877*, 2019.

[Bar96] Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96*, pages 184–193, 1996.

[BBBT97] Yair Bartal, Avrim Blum, Carl Burch, and Andrew Tomkins. A polylog(n)-competitive algorithm for metrical task systems. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 711–719, 1997.

[BBK+94] 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.

[BBM06] 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.

[BBMN15] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. *J. ACM*, 62(5):40:1–40:49, 2015.

[BBN10] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. Metrical task systems and the k-server problem on HSTs. In *Proceedings of the 37th International Colloquium Conference on Automata, Languages and Programming*, ICALP'10, pages 287–298, 2010.

[BBN12] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. *J. ACM*, 59(4):Art. 19, 24, 2012.

[BC12] Sébastien Bubeck and Nicolò Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learning*, 5(1):1–122, 2012.

[BCL02] Wolfgang W. Bein, Marek Chrobak, and Lawrence L. Larmore. The 3-server problem in the plane. *Theor. Comput. Sci.*, 289(1):335–354, 2002.

[BCL+18] Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. k-server via multiscale entropic regularization. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pages 3–16, 2018.

[BCLL19] Sébastien Bubeck, Michael B. Cohen, James R. Lee, and Yin Tat Lee. Metrical task systems on trees via mirror descent and unfair gluing. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 89–97, 2019.

[BCN14] Niv Buchbinder, Shahar Chen, and Joseph (Seffi) Naor. Competitive analysis via regularization. In *Proceedings of the Twenty-fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 436–444. Society for Industrial and Applied Mathematics, 2014.

[BE98] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[BEJ+15] Nikhil Bansal, Marek Eliáš, Łukasz Jeż, Grigorios Koumoutsos, and Kirk Pruhs. Tight bounds for double coverage against weak adversaries. In *International Workshop on Approximation and Online Algorithms*, pages 47–58. Springer, 2015.

[BEJK19] Nikhil Bansal, Marek Eliés, Lukasz Jez, and Grigorios Koumoutsos. The (h,k)-server problem on bounded depth trees. *ACM Trans. Algorithms*, 15(2):28:1–28:26, 2019.

[BEK17] Nikhil Bansal, Marek Eliás, and Grigorios Koumoutsos. Weighted k-server bounds via combinatorial dichotomies. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 493–504, 2017.

[BEKN18] Nikhil Bansal, Marek Eliás, 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 2018*, pages 992–1001, 2018.

[BGMN19] Niv Buchbinder, Anupam Gupta, Marco Molinaro, and Joseph (Seffi) Naor. k-servers with a smile: Online algorithms via projections. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 98–116, 2019.

[BK04] Yair Bartal and Elias Koutsoupias. On the competitive ratio of the work function algorithm for the k-server problem. *Theoretical Computer Science*, 324(2-3):337–345, September 2004.

[BKRS00] Avrim Blum, Howard Karloff, Yuval Rabani, and Michael Saks. A decomposition theorem for task systems and bounds for randomized server problems. *SIAM J. Comput.*, 30(5):1624–1661, 2000.

[BLLS19] Sébastien Bubeck, Yin Tat Lee, Yuanzhi Li, and Mark Sellke. Competitively chasing convex bodies. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 861–868, 2019.

[BLMN05] Yair Bartal, Nathan Linial, Manor Mendel, and Assaf Naor. On metric Ramsey-type phenomena. *Ann. of Math. (2)*, 162(2):643–709, 2005.

[BLS92] Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, October 1992.

[BN09] 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.

[Bur96] William R. Burley. Traversing layered graphs using the work function algorithm. *J. Algorithms*, 20(3):479–511, 1996.

[CDRS93] 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.

[CEFJ14] Ilan R. Cohen, Alon Eden, Amos Fiat, and Łukasz Jeż. Pricing online decisions: Beyond auctions. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on discrete algorithms*, pages 73–91. SIAM, 2014.

[CIN+01] János Csirik, Csanád Imreh, John Noga, Steve S. Seiden, and Gerhard J. Woeginger. Buying a constant competitive ratio for paging. In *European Symposium on Algorithms*, pages 98–108. Springer, 2001.

[CK19] Christian Coester and Elias Koutsoupias. The online k-taxi problem. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 1136–1147, 2019.

[CKL17] Christian Coester, Elias Koutsoupias, and Philip Lazos. The infinite server problem. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, pages 14:1–14:14, 2017.

[CKPV91] Marek Chrobak, Howard Karloff, Tom Payne, and Sundar Vishwanathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, 1991.

[CL91] Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k servers on trees. *SIAM Journal on Computing*, 20(1):144–148, 1991.

[CL92] 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*. Citeseer, 1992.

[CL96] Marek Chrobak and Lawrence L. Larmore. Metrical task systems, the server problem and the work function algorithm. In *Online Algorithms, The State of the Art (Proc. Dagstuhl Seminar, June 1996)*, pages 74–96, 1996.

[CL19] Christian Coester and James R. Lee. Pure entropic regularization for metrical task systems. In *Conference on Learning Theory, COLT 2019*, pages 835–848, 2019.

[CS04] Marek Chrobak and Jirí Sgall. The weighted 2-server problem. *Theor. Comput. Sci.*, 324(2-3):289–312, 2004.

[CV13] Ashish Chiplunkar and Sundar Vishwanathan. On randomized memoryless algorithms for the weighted k-server problem. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013*, pages 11–19, 2013.

[DEH+17] Sina Dehghani, Soheil Ehsani, MohammadTaghi Hajiaghayi, Vahid Liaghat, and Saeed Seddighin. Stochastic k-Server: How Should Uber Work? In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, pages 126:1–126:14, 2017.

[FFK+98] Amos Fiat, Dean P. Foster, Howard J. Karloff, Yuval Rabani, Yiftach Ravid, and Sundar Vishwanathan. Competitive algorithms for layered graph traversal. *SIAM J. Comput.*, 28(2):447–462, 1998.

[FKL+91] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel D. Sleator, and Neal E. Young. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991.

[FL93] Joel Friedman and Nathan Linial. On convex body chasing. *Discrete & Computational Geometry*, 9:293–321, 1993.

[FM03] Amos Fiat and Manor Mendel. Better algorithms for unfair metrical task systems and applications. *SIAM Journal on Computing*, 32(6):1403–1422, 2003.

[FR94] Amos Fiat and Moty Ricklin. Competitive algorithms for the weighted server problem. *Theor. Comput. Sci.*, 130(1):85–99, 1994.

[FRR90] Amos Fiat, Yuval Rabani, and Yiftach Ravid. Competitive k-server algorithms (extended abstract). In *31st Annual Symposium on Foundations of Computer Science*, pages 454–463, 1990.

[FRT04] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, 2004.

[FW98] Amos Fiat and Gerhard J. Woeginger, editors. *Online Algorithms, The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*. Springer, 1998.

[Gra66] Ronald L Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966.

[Gro91] Edward F. Grove. The harmonic online k-server algorithm is competitive. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 260–266, 1991.

[Haz16] Elad Hazan. Introduction to online convex optimization. *Foundations and Trends in Optimization*, 2(3-4):157–325, 2016.

[IR91] Sandy Irani and Ronitt Rubinfeld. A competitive 2-server algorithm. *Information Processing Letters*, 39(2):85–91, 1991.

[KMRS88] Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. Competitive snoopy caching. *Algorithmica*, 3:77–119, 1988.

[Kos96] Andrew P. Kosoresow. *Design and analysis of online algorithms for mobile server applications*. PhD thesis, Stanford University, 1996.

[Kou99] Elias Koutsoupias. Weak adversaries for the k-server problem. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99*, pages 444–449, 1999.

[Kou09] Elias Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009.

[KP95] Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *Journal of the ACM (JACM)*, 42(5):971–983, 1995.

[KP96] Elias Koutsoupias and Christos Papadimitriou. The 2-evader problem. *Information Processing Letters*, 57(5):249–252, 1996.

[KP00] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.

[KT04] Elias Koutsoupias and David Scot Taylor. The CNN problem and other k-server variants. *Theor. Comput. Sci.*, 324(2-3):347–359, 2004.

[Lee18a] James R. Lee. Fusible HSTs and the randomized k-server conjecture. In *Proceedings of the 59th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '18, pages 438–449, 2018.

[Lee18b] James R. Lee. Fusible HSTs and the randomized k-server conjecture. arXiv:1711.01789v2, February 2018.

[Lee19] James R. Lee. Personal Communication, 2019.

[MMS88] Mark Manasse, Lyle McGeoch, and Daniel Sleator. Competitive algorithms for on-line problems. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 322–333. ACM, 1988.

[PSTW02] Cynthia A. Phillips, Clifford Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32(2):163–200, 2002.

[PY91] Christos H. Papadimitriou and Mihalis Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84(1):127–150, 1991.

[Ram95] H. Ramesh. On traversing layered graphs on-line. *J. Algorithms*, 18(3):480–512, 1995.

[RS94] Prabhakar Raghavan and Marc Snir. Memory versus randomization in on-line algorithms. *IBM Journal of Research and Development*, 38(6):683–708, 1994.

[Sei99] Steve Seiden. Unfair problems and randomized algorithms for metrical task systems. *Inf. Comput.*, 148(2):219–240, 1999.

[Sel19] Mark Sellke. Chasing convex bodies optimally. *arXiv:1905.11968*, 2019.

[SS06] René A. Sitters and Leen Stougie. The generalized two-server problem. *J. ACM*, 53(3):437–458, 2006.

[ST85a] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[ST85b] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985.

[You91] Neal E. Young. On-line caching as cache size varies. In *Proceedings of the Second Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms*, pages 241–250, 1991.

[You94] Neal Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, 1994.