The Online k-Taxi Problem*

Christian Coester and Elias Koutsoupias

Department of Computer Science, University of Oxford
{christian.coester,elias.koutsoupias}@cs.ox.ac.uk

Abstract

We consider the online k-taxi problem, a generalization of the k-server problem, in which k taxis serve a sequence of requests in a metric space. A request consists of two points s and t, representing a passenger that wants to be carried by a taxi from s to t. The goal is to serve all requests while minimizing the total distance traveled by all taxis. 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 only the distance of empty runs.

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 hierarchically separated trees (HSTs), 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 well-known HST embedding techniques, 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).

*Partially supported by the ERC Advanced Grant 321171 (ALGAME) and by EPSRC.

1 Introduction

The k-taxi problem, originally proposed by Karloff and introduced by Fiat et al. [16], is a natural generalization of the fundamental k-server problem. In this problem, k taxis are located in a metric space and need to serve a sequence σ 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 goal is to serve all requests in order while minimizing the total distance traveled by all taxis. We consider the online version of this problem where requests appear one by one, i.e., a new request is revealed only after the previous request has been served.

Since the distance from s to t needs to be traveled anyway — independently of the algorithm's decisions — it makes sense to exclude it from the cost and minimize only the overhead travel that actually depends on the algorithm choices. This is precisely what the hard k-taxi problem does: In the hard version, the cost is defined as the distance traveled while not carrying a passenger, i.e., the overhead distance traveled on top of the distances between the start-destination pairs. In contrast, for the easy k-taxi problem, the cost is the total distance traveled by the taxis. 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. [16] pointed out the two versions of this problem, and they were called easy taxicab problem and hard taxicab problem by Kosoresow [17].

The problem was recently reintroduced as the Uber problem in [13], which studied the easy version of the problem with the input being produced in a stochastic manner. Here we consider the adversarial case. This worst-case analysis is arguably useful for developing algorithms and improving our understanding of a problem, even when there is sufficiently large collected data to allow us to treat the problem as a stochastic one, which arguably is the case with Uber, Lyft etc.

Besides scheduling taxis, the k-taxi problem also models other tasks such as scheduling elevators (in which case the metric space is the line; see Section 2.6) or transport vehicles in a factory, and other applications where people or objects need to be transported between locations.

1.1 Previous results and related work

The first competitive algorithm for the easy k-taxi problem was given by Fiat et al. [16] when they introduced the problem, with a competitive ratio exponential in k. Following the finding of a $(2k-1)$-competitive algorithm for the k-server problem [19], the competitive ratio of the easy k-taxi problem was improved to $2k+1$: Kosoresow [17] 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 [13] with a similar reduction.

For the hard k-taxi problem, Fiat et al. [16] mentioned a competitive algorithm by Karloff for $k = 2$. Based on Karloff's algorithm, Kosoresow [17] gave a 15-competitive algorithm for $k = 2$. No competitive algorithm is known for $k > 2$.

The k-server problem [21], one of the most studied online problems, is the special case of the (easy and hard) k-taxi problem where for each request, start and destination are identical. Thus, the lower bound of k on the competitive ratio of the k-server problem [21] immediately implies the same lower bound for the k-taxi problem. According to the famous *k-server conjecture*, this bound is tight for the k-server problem, yet the best known upper bound remains $2k - 1$ [19]. The *randomized k-server conjecture* states that a competitive ratio of $O(\log k)$ can be achieved by randomized algorithms on any metric space, and there has been tremendous progress on this question recently [1, 7, 20]. More information on the k-server problem is presented in [18]. Besides the k-taxi/Uber problem, recent work on other variants of the k-server problem include the (h, k)-server problem [3, 2], the infinite server problem [11] and the weighted k-server problem [4].

The layered graph traversal problem, first introduced in [22], is another deep problem in online computation and known to be equivalent to the metrical service systems problem [15]. As we will see, the k-taxi problem generalizes not only the k-server problem but also the (deterministic) layered graph traversal problem. The best known bounds on the competitive ratio of layered graph traversal (for graphs of width k) are $\Omega(2^k)$ [15] and $O(k2^k)$ [8] in the deterministic case, and $O(k^{13})$ and $\Omega\left(\frac{k^2}{\log^{1+\epsilon}(k)}\right)$ in the randomized case [24].

1.2 Our results

For the hard k-taxi problem, we show tight bounds on HSTs (defined in Section 1.3) in two settings: for randomized algorithms against adaptive online adversaries, and for memoryless randomized algorithms against oblivious adveraries:

Theorem 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 [6], this also implies a 4^k-competitive deterministic algorithm for HSTs. More importantly, thanks to known probabilistic approximation of general n-point metrics by HSTs with distortion $O(\log n)$ [14], we obtain the first competitive algorithm for the hard k-taxi problem on general finite metrics:

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

For deterministic algorithms, we show a reduction from the layered width-k graph traversal problem (k-LGT) to the hard k-taxi problem.

Theorem 3. *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. In particular, the deterministic competitive ratio of the hard k-taxi problem is $\Omega(2^k)$.*

This also shows that the hard k-taxi problem is substantially more difficult than the k-server problem.

Although the lower bound of $\Omega(2^k)$ also follows already from the first lower bound in Theorem 1, the relation between the hard k-taxi problem and k-LGT is of independent interest, as it reveals an interesting connection between these two problems; a connection which can be potentially exploited to extend to lower bounds for randomized algorithms.

For the special case $k = 2$, we improve the previous upper bound of 15 [17] and give tight bounds of 9 on general metrics:

Theorem 4. *The deterministic competitive ratio of the hard 2-taxi problem is exactly 9.*

For the upper bound, we present a simple modification of the well-known Double Coverage algorithm with biased taxi speeds. The lower bound follows from the reduction from k-LGT and the fact that the deterministic competitive ratio of 2-LGT is exactly 9 [22].

With a significant extension of the algorithm, we obtain the following result for three taxis when the metric space is the real line:

Theorem 5. *There is an $O(1)$-competitive algorithm for the hard 3-taxi problem on the line.*

Our algorithm achieving this upper bound keeps track of regions around the taxis where they move more aggressively. These regions are continuously expanded, shrunk and shifted while the algorithm serves requests.

Lastly, we show the easy k-taxi problem to be exactly equivalent to the k-server problem, tightening the old result by removing the "+2" term:

Theorem 6. *The easy k-taxi problem has the same deterministic/randomized competitive ratio as the k-server problem. In particular, the deterministic competitive ratio of the easy k-taxi problem is between k and $2k-1$, and it is k if and only if the k-server conjecture holds.*

1.3 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. An online algorithm has to make this decision without knowledge of the future requests. 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.

The performance of an online algorithm is measured by comparing its cost to that of an adversary who generates a request sequence and also serves it. An algorithm A is *ρ-competitive* if $E(cost_A(\sigma)) \leq \rho E(cost_{ADV}(\sigma)) + c$ for every possible request sequence σ generated by the adversary. Here, $cost_A(\sigma)$ and $cost_{ADV}(\sigma)$ denote the cost of the algorithm and the adversary respectively to serve σ, and c is a constant independent of σ. We consider two types of adversaries: The *oblivious adversary* generates the request sequence in advance, independently of the outcome of random choices by the algorithm. Since this adversary knows in advance the sequence σ it will generate, it can serve it *offline* with optimal cost $OPT(\sigma)$, i.e., $cost_{ADV}(\sigma) = OPT(\sigma)$ for the oblivious adversary. The *adaptive online adversary* makes the next request based on the algorithm's answers to previous requests, but serves it immediately. Thus, σ is a random variable and $cost_{ADV}(\sigma)$ is not necessarily optimal. If no adversary type is stated explicitly, then we mean the oblivious adversary. For deterministic algorithms, the two notions are identical. The deterministic/randomized *competitive ratio* of an online problem (and a given adversary type) is the infimum of all ρ such that a deterministic/randomized ρ-competitive algorithm exists.

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 a 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). The *k-server problem* is the special case of the (easy and hard) k-taxi problem where all requests are simple, and in this case the taxis are called *servers*. 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) [5]. For $\alpha > 1$, an α-*HST* is a tree where all leaves have the same combinatorial distance from the root 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 $O(\log n)$ [14].[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.

1.4 Organization

The hard k-taxi problem is studied in Section 2. To prove Theorem 1, we present the $(2^k - 1)$-competitive algorithm for HSTs in Subsection 2.1, and the matching lower bounds in Subsections 2.2 and 2.3. In Subsection 2.4 we give the reduction from layered graph traversal that yields Theorem 3 and the lower bound of Theorem 4. The upper bound of Theorem 4 is shown in Subsection 2.5. The algorithm for three taxis on the line (Theorem 5) is presented in Subsection 2.6. The equivalence of the easy k-taxi problem and the k-server problem (Theorem 6) is shown in Section 3.

2 The hard k-taxi problem

2.1 An optimally competitive 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 $t_1, \dots, t_k$. For each taxi we need to specify its probability to serve s. Let N be the Steiner tree of $s, t_1, \dots, t_k$, 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 $t_1, \dots, t_k$, 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 $s_A, t_1, \dots, t_k$. Formally, we view A as the set of all nodes and edges of this subtree. We denote by $\kappa(A) = |A \cap \{t_1, \dots, t_k\}|$ the number of leaves of A where a taxi is located. Note that $N \in \mathcal{N}$ and $\kappa(N) \le k$, with equality if and only if all taxis are located at different leaves.

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

[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. [12]: 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.

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. The resistance of A is defined as

$$R_A = d(s_A, s'_A) + \frac{R_B R_C}{R_B + R_C}. \tag{1}$$

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 from the request location to the taxi locations, and whichever taxi we end up at is the one that will 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 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 7. 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 only need to show for a simple request at a leaf s where the adversary already has a taxi that

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

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 $P_{t_i s} \cap P_{t_s s}$. 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

$$\Delta M \leq \ell(P_{t_i s} \setminus P_{t_s s}) - \ell(P_{t_i s} \cap P_{t_s s}). \tag{2}$$

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.

[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.

For $A \in \mathcal{N}$ let

$$m(A) = E(\ell(A \cap P_{t_is} \setminus P_{t_ss}) - \ell(A \cap P_{t_is} \cap P_{t_ss}) \mid t_i \in A)$$

be the expected contribution of edges from A to the matching bound (2), conditioned on FLOW using a taxi that starts in A. Moreover, let

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

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

$$c(A) = d(s_A, s'_A) + \frac{R_C c(B)}{R_B + R_C} + \frac{R_B c(C)}{R_B + R_C}. \tag{3}$$

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

$$c(N) + (2^k - 1)m(N) \leq 0. \tag{4}$$

A key insight is provided by the following claim, relating $m(A)$ to $c(A)$ and R_A, which will allow us to reformulate (4) purely in terms of the expected cost $c(N)$ and resistance R_N.

Claim 8. *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 $A = P_{s_At_s}$; 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 $m(A) = -\ell(P_{s_At_s})$ and $c(A) = \ell(P_{s_At_s})$. Since $R_A = d(s_A, t_s) = \ell(P_{s_At_s})$, the claim follows.

If $\kappa(A) \geq 2$, then A can be split into the path $P_{s_As'_A}$ and subtrees $B, C \in \mathcal{N}$ as above. Conditional on $t_i \in A$, the path $P_{s_As'_A}$ is contained in both P_{t_is} and P_{t_ss}, so it contributes negatively to $m(A)$. Moreover, conditional on $t_i \in A$, the remaining part of $A \cap P_{t_is}$ 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_ss} do not intersect with C, and therefore $m(C) = c(C)$. This gives us

$$m(A) = -d(s_A, s'_A) + \frac{R_C(c(B) - 2R_B)}{R_B + R_C} + \frac{R_B c(C)}{R_B + R_C}. \tag{5}$$

Combining (3) and (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 (4) as

$$2^{k-1}c(N) \leq (2^k - 1)R_N. \tag{6}$$

Observe that unlike (4), the reformulation (6) no longer depends on the adversary configuration and the matching.

To show (6), we will need the following property: For $A \in \mathcal{N}$,

$$\kappa(A)R_A \geq h(A), \tag{7}$$

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 (6):

Claim 9. *For $A \in \mathcal{N}$ with $s \in A$, we have $2^{\kappa(A)-1}c(A) \le (2^{\kappa(A)}-1)R_A$.*

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

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

From (1) and (3), we get

$$\begin{aligned} &2^{\kappa(A)-1}c(A) - (2^{\kappa(A)}-1)R_A \\ &\quad = (1-2^{\kappa(A)-1})d(s,s'_A) + 2^{\kappa(A)-1}\left(\frac{R_C c(B)}{R_B+R_C} + \frac{R_B c(C)}{R_B+R_C}\right) - (2^{\kappa(A)}-1)\frac{R_B R_C}{R_B+R_C}. \end{aligned} \tag{8}$$

We will show that this term is negative. Thus, for $\kappa(A) \ge 2$ 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 (7) to $B \cup C \in \mathcal{N}$, we get that term (8) is less than

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

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

$$c(B) = h(B) = d(s, s'_A), \tag{9}$$

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

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

and reordering,

$$c(C) \le (1-2^{1-\kappa(C)})d(s,s'_A) + (2-2^{1-\kappa(C)})R_C. \tag{10}$$

Due to (9), (10) and (7), we can bound term (8) by

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

and the claim follows. □

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

Invoking the claim for $A = N$, and using $\kappa(N) \leq k$, we obtain (6), concluding the proof of the theorem. $\square$

2.2 Lower bound against adaptive adversaries

We now show the first lower bound of Theorem 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 1 follows from the following theorem by letting $\alpha \to \infty$ and $T \to \infty$:

Theorem 10. *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

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

(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

$$E(cost_{A_i}(\sigma^i_{T_i}) \mid \mathcal{F}_i) \geq \left(2^k - 1 - \frac{3^k}{\alpha}\right) E(cost_{ADV_i}(\sigma^i_{T_i}) \mid \mathcal{F}_i) - (2\alpha)^k, \tag{11}$$

where $\mathcal{F}_i$ contains the information about A's decisions before phase i. Moreover, $cost_{A_i}(\sigma^i_t) \to \infty$ 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 (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,

$$
\begin{aligned}
&E(cost_A(\sigma_T)) \\
&\quad \ge E\left(\sum_{i=1}^{\infty} cost_{A_i}(\sigma^i_{T_i}) + \mathbb{1}_{\{i<n(T)\}}\alpha^k(\alpha-1)\right) \\
&\quad = \left(\sum_{i=1}^{\infty} E(cost_{A_i}(\sigma^i_{T_i}))\right) + (E(n(T))-1)\alpha^k(\alpha-1) \\
&\quad \ge \left(\sum_{i=1}^{\infty}\left(2^k - 1 - \frac{3^k}{\alpha}\right) E(cost_{ADV_i}(\sigma^i_{T_i})) - P(i \le n(T))(2\alpha)^k\right) + E(n(T))\alpha^k(\alpha-1) - \alpha^{k+1} \\
&\quad \ge \left(2^k - 1 - \frac{3^k}{\alpha}\right) E\left(\sum_{i=1}^{n(T)} cost_{ADV_i}(\sigma^i_{T_i})\right) + E(n(T))\alpha^k(\alpha - 3^k) - \alpha^{k+1}. \qquad (12)
\end{aligned}
$$

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

$$n(T)\alpha^{k+1}. \qquad (13)$$

In the second strategy we consider, the adversary never moves a taxi from one subtree B^α_k 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

$$\sum_{\substack{i=1\\ i \text{ odd}}}^{n(T)} cost_{ADV_i}(\sigma^i_{T_i}). \qquad (14)$$

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

$$\alpha^{k+1} + \sum_{\substack{i=2\\ i \text{ even}}}^{n(T)} cost_{ADV_i}(\sigma^i_{T_i}). \qquad (15)$$

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

Since $E(cost_{ADV}(\sigma_T))$ is bounded by the expectations of (14) and (15), we have

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

Therefore, with (12) we get

$$E(cost_A(\sigma_T)) \geq \left(2^{k+1} - 2 - \frac{2\cdot 3^k}{\alpha}\right) E(cost_{ADV}(\sigma_T)) + E(n(T))\alpha^k(\alpha - 3^k) - (2\alpha)^{k+1}. \quad (16)$$

Since $E(cost_{ADV}(\sigma_T))$ is bounded by the expectation of (13), for $\alpha \geq 3^k$ we can bound the middle term of (16) by

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

Therefore,

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

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. □

2.3 Lower bound for memoryless algorithms

We now show the other lower bound of Theorem 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 11. *Let $C'_0, \dots, C'_n$ be a $(k+1)$-taxi schedule of cost 0 for a request sequence σ.*

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

(b) If $C_0, \dots, C_n$ 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 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 12. *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) $E(cost_{A_k}(C, \sigma)) \geq (2^k - 1)(2\alpha)^k(N - 2^k)$,

(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 $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ sorted lexicographically, 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). For an algorithm A, an initial configuration C and a request sequence σ, we denote by $A(C, \sigma)$ the corresponding schedule.

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 $A_{k+1}(C_{r\ell} \cup \{r\}, \sigma')$.

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

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

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

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

[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.

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.

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

$$\limsup_{m \to \infty} E(cost_{A_{k+1}}(C, \sigma)) < \infty, \tag{17}$$

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

Let us first examine the behaviour of $A_{k+1}(C_{r\ell} \cup \{r\}, \sigma_{r\ell} C_{r\ell}^m)$. 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

$$E(cost_{A_{k+1}}(C_{r\ell} \cup \{r\}, \sigma_{r\ell} C_{r\ell}^m)) \ge p_{r\ell,m} \alpha^{k+1} + (1 - p_{r\ell,m})(2^k - 1)(2\alpha)^k (N - 2^k) + \epsilon_{r\ell,m} \alpha m. \tag{18}$$

Claim 13. *$r \notin F_r(\sigma_{r\ell} C_{r\ell}^m)$ 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 11(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

$$P(F_r(\sigma_{r\ell} C_{r\ell}^m) = C_{r\ell} \cup \{\ell\}) = p_{r\ell,m} \tag{19}$$

$$P(F_r(\sigma_{r\ell} C_{r\ell}^m) = C_{r\ell} \cup \{r\}) = 1 - p_{r\ell,m} - \epsilon_{r\ell,m}. \tag{20}$$

For $i \geq 0$, we have

$$
\begin{aligned}
&P(F_r(\sigma_{r\ell}C_{r\ell}^m(C_{r\ell} \to C_{\ell r})(\sigma_{\ell r}C_{\ell r}^m)^i) = C_{\ell r} \cup \{\ell\}) \\
&\geq P(F_r(\sigma_{r\ell}C_{r\ell}^m) = C_{r\ell} \cup \{\ell\} \wedge \forall j = 1,\dots,i\colon F_r(\sigma_{r\ell}C_{r\ell}^m(C_{r\ell} \to C_{\ell r})(\sigma_{\ell r}C_{\ell r}^m)^j) = C_{\ell r} \cup \{\ell\}) \\
&= P(F_r(\sigma_{r\ell}C_{r\ell}^m) = C_{r\ell} \cup \{\ell\})P(F_\ell(\sigma_{\ell r}C_{\ell r}^m) = C_{\ell r} \cup \{\ell\})^i \\
&= p_{r\ell,m}(1 - p_{\ell r,m} - \epsilon_{\ell r,m})^i, \qquad (21)
\end{aligned}
$$

where the first equation uses memorylessness of A_{k+1} and the last equation uses (19) and the symmetric version of (20).

Using (21), (18) and the symmetric version of (18), we can bound the cost during σ_r by

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

Due to (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

$$
\begin{aligned}
&\limsup_{m\to\infty} E(cost_{A_{k+1}}(C_{r\ell} \cup \{r\}, \sigma_r) \\
&\geq p_{r\ell}\alpha^{k+1} + (1 - p_{r\ell})(2^k - 1)(2\alpha)^k(N - 2^k) + \frac{p_{r\ell}}{p_{\ell r}}\left(p_{\ell r}\alpha^{k+1} + (1 - p_{\ell r})(2^k - 1)(2\alpha)^k(N - 2^k)\right) \\
&= 2p_{r\ell}\alpha^{k+1} + \left(1 - 2p_{r\ell} + \frac{p_{r\ell}}{p_{\ell r}}\right)(2^k - 1)(2\alpha)^k(N - 2^k). \qquad (22)
\end{aligned}
$$

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

Claim 14. *For all $i \leq \alpha$, $P(F_r(\sigma_r^i) = C_{r\ell} \cup \{r\}) \to 1$ as $m \to \infty$.*

Proof. Since

$$P(F_r(\sigma_r^i) = C_{r\ell} \cup \{r\}) \geq P(\forall j = 1,\dots,i\colon F_r(\sigma_r^j) = C_{r\ell} \cup \{r\}) = P(F_r(\sigma_r) = C_{r\ell} \cup \{r\})^i,$$

we only need to show the claim for $i = 1$. Thanks to (19), (20) and $\epsilon_{r\ell} = 0$, it suffices to show

$$P(F_r(\sigma_r) = C_{r\ell} \cup \{r\} \mid F_r(\sigma_{r\ell}C_{r\ell}^m) = C_{r\ell} \cup \{\ell\}) \to 1 \text{ as } m \to \infty \qquad (23)$$

$$P(F_r(\sigma_r) = C_{r\ell} \cup \{r\} \mid F_r(\sigma_{r\ell}C_{r\ell}^m) = C_{r\ell} \cup \{r\}) = 1. \qquad (24)$$

We first show (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 (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,

$$
\begin{aligned}
&P(F_r(\sigma_r) = C_{r\ell} \cup \{r\} \mid F_r(\sigma_{r\ell}C_{r\ell}^m) = C_{r\ell} \cup \{\ell\}) \\
&= P(F_\ell((\sigma_{\ell r}C_{\ell r}^m)^m(C_{\ell r} \to C_{r\ell})) = C_{r\ell} \cup \{r\}) \\
&\geq P(F_\ell((\sigma_{\ell r}C_{\ell r}^m)^m) = C_{\ell r} \cup \{r\}) \\
&= 1 - P(\forall i \leq m\colon F_\ell((\sigma_{\ell r}C_{\ell r}^m)^i) \neq C_{\ell r} \cup \{r\}) \\
&= 1 - P(\forall i \leq m\colon F_\ell((\sigma_{\ell r}C_{\ell r}^m)^i) = C_{\ell r} \cup \{\ell\}) - P(\exists i \leq m\colon F_\ell((\sigma_{\ell r}C_{\ell r}^m)^i) \notin \{C_{\ell r} \cup \{\ell\}, C_{\ell r} \cup \{r\}\}) \\
&= 1 - (1 - p_{\ell r,m} - \epsilon_{\ell r,m})^m - P(\exists i \leq m\colon C_{\ell r} \not\subset F_\ell((\sigma_{\ell r}C_{\ell r}^m)^i)),
\end{aligned}
$$

where the second equation uses part (c) of the induction hypothesis in the same way as it was used in the proof of (24). Due to our assumptions $p_{\ell r} > 0$ and (17), both subtrahends in the last term tend to 0 as $m \to \infty$. ☐

Let $q_m = P\left(F_r(\sigma_r^\alpha(C_{r\ell} \to C_{\ell r})(C_{\ell r} \cup \{\ell\})^m) = C_{\ell r} \cup \{\ell\}\right)$. 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 (17), we have $q_m \to 1$ as $m \to \infty$. We are now ready to put the parts together:

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

and therefore

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

where the first inequality uses $q_m \to 1$ and Claim 14 and its symmetric case, the second inequality uses 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

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

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^kN times, we get $w(C,\sigma) \leq 2^kN(\alpha^{k+1}+\alpha^{k+1}) \leq (2\alpha)^{k+1}(N+k)$, 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

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

and, with assumption (17),

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

Together with the symmetric equivalents of these statements, this gives

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

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 $p_{\ell r} + p_{\ell r} \leq \frac{2^k}{N}$. This completes the proof. □

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 $L_{\ell-1}$.

It is known that the deterministic competitive ratio of k-LGT is between $\Omega(2^k)$ [15] and $O(k2^k)$ [8], and it is 9 for $k = 2$ [22]. By the following reduction, these lower bounds translate to the hard k-taxi problem, giving Theorem 3 as well as the lower bound of Theorem 4.[7]

Theorem 15. *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. [15] 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

[7] Recall that the $\Omega(2^k)$ lower bound also follows already from the lower bound for adaptive adversaries in Theorem 1.

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

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

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,

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

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. ☐

2.5 An optimally competitive algorithm for two taxis

We will define a deterministic 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 well-known double coverage (DC) algorithm for the k-server problem [9, 10]. 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 4.

Theorem 16. 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

$$\mathit{cost} + \Delta\Phi \leq 9OPT. \tag{25}$$

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 (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, $\mathit{cost} = 3$. 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 (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 (25) follows again. □

2.6 A competitive algorithm for three taxis on the line

In this section, we present an algorithm that achieves a constant competitive ratio for the hard 3-taxi problem when the metric space is the real line. The algorithm, which we call REGIONTRACKER, is somewhat similar to BIASEDDC in that it moves the taxis at different speeds towards the request. 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. Algorithm 1 contains the pseudocode of REGIONTRACKER. An example of the steps involved in serving a simple request is depicted in Figure 2.6.

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 $x_i \in I_i$ 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 17. *If* $(r_1, \ell_2, r_2, \ell_3) = sort(e_1, e_2, x_A, x_A)$*, 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 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). We use here and onwards the notation $x \wedge y = \min\{x, y\}$ and $x \vee y = \max\{x, y\}$. 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.

Algorithm 1 RegionTracker

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

$A \leftarrow 1$
$(r_0, \ell_1, r_1, \ell_2, r_2, \ell_3, r_3, \ell_4) \leftarrow (-\infty, -\infty, x_1, x_1, x_2, x_3, \infty, \infty)$
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 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
    $(e_1, e_2) \leftarrow (r_1, \ell_2, r_2, \ell_3) \setminus (x_A, x_A)$
    $x_A \leftarrow t$
    $(x_1, x_2, x_3) \leftarrow \mathrm{sort}(x_1, x_2, x_3)$
    $A \leftarrow \min\{i \mid x_i = t\}$
    $(r_1, \ell_2, r_2, \ell_3) \leftarrow \mathrm{sort}(e_1, e_2, x_A, x_A)$
end for

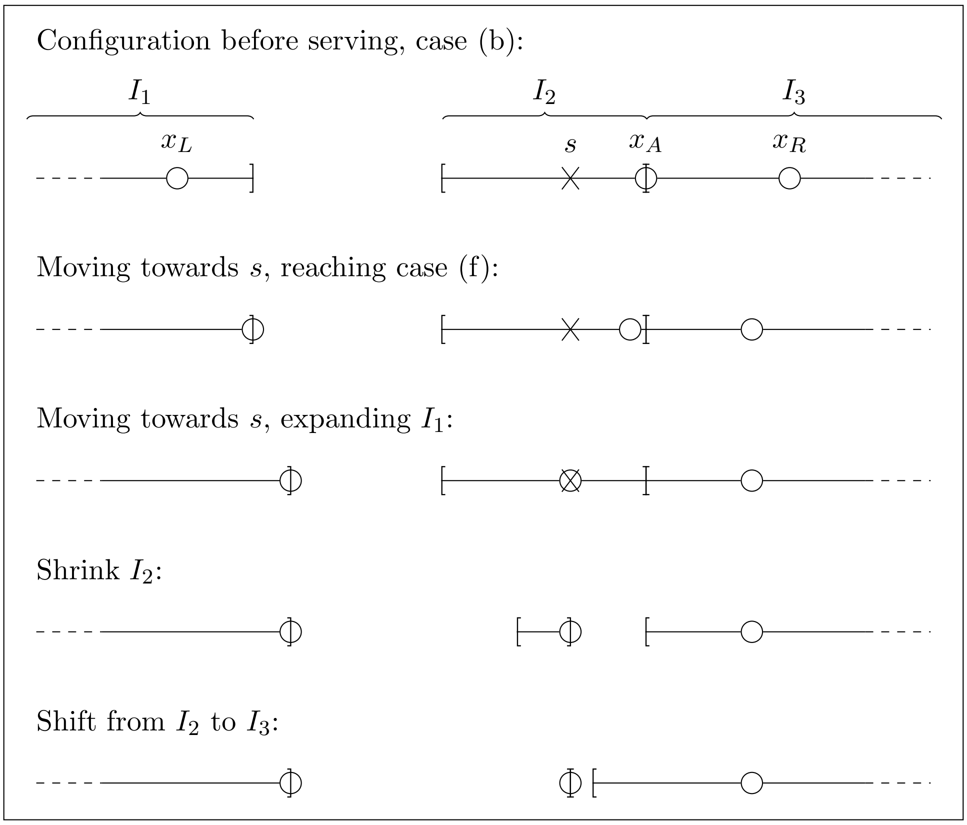

Figure 1: Example of RegionTracker serving a simple request at s.

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 $e_1 \le e_2$ 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 $x_1 \le x_2 \le x_3$ are again in the right order, and update A accordingly. Finally, we update the interval endpoints to react to the relocation.

	Conditions	x_1'	x_2'	x_3'	$\Sigma' \le$	if a changed: $\Psi' \le$
(a)	$s < x_1$	-1	0	0	"0"	$-(\gamma - \psi)$
(b)	$s > x_1, x_1 < r_1, \ell_2 < x_2, \ell_3 < x_3$	$b+1$	-1	$-b$	$-b$	0
(c)	$s > x_1, x_1 < r_1, \ell_2 < x_2, \ell_3 = x_3$	1	-1	0	-1	0
(d)	$s > x_1, x_1 = r_1, \ell_2 = x_2$	1	-1	0	1	$-\psi$
(e)	$s > x_1, x_1 < r_1, \ell_2 = x_2$	$b+1$	-1	0	1	$-(\gamma - \psi)b$
(f)	$s > x_1, x_1 = r_1, \ell_2 < x_2, A \ge 2$	1	$-(b+1)$	0	$b+1$	$-(\gamma - \psi)b$
(g)	$s > x_1, x_1 = r_1, \ell_2 < x_2, A = 1$	1	$-(c+1)$	0	$c+1$	$2\psi - c(\gamma - \psi)$

Table 1: Rates of movement if $s \in (-\infty, x_1) \cup (x_1, x_2)$, where $c > b > 0$ are constants. Last two columns for analysis.

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 $O(1)$-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 potential of the Double Coverage algorithm for the k-server problem [9]. 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 18. *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 17, 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 19. *Φ remains constant during the relocation in lines 24–27 of Algorithm 1.*

Proof. By Claim 18, 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 $r_{m-2} \leq x_{m-1} \leq \ell_m \leq x_m < z$, 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 $z \leq e_1$, 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, $z \leq r_m$ 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 $\ell_{m+1} = e_2 \geq z$. 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 20. *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 21. *During the shift step (lines 14–19), Φ does not increase.*

Proof. Similar to the proof of Claim 20. □

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 22. *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 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 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 5.

3 The easy k-taxi problem

We now turn to the easy k-taxi problem, and prove that it is equivalent to the k-server problem.

Proof of Theorem 6. 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 $r_0, \dots, r_{2kN}$ 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 $r_0, \dots, r_{2kN}$. Therefore, since A is ρ-competitive on $\tilde{M}$,

$$\begin{aligned} cost_A(\sigma_{server}) &\leq \rho OPT(\sigma_{server}) + c \\ &\leq \rho OPT(\sigma_{taxi}) + c \end{aligned} \tag{26}$$

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 (26). □

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 2. We know 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 [8], 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 [20].

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 distance). 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 10 and 12 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 [10]. 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 $2k - 1$.[8] 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. [18]; see also [23]). It would be interesting to prove this as a generic result for a broad class of online problems.

References

[1] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011*, pages 267–276, 2011. doi:10.1109/FOCS.2011.63.

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

[3] 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.

[4] Nikhil Bansal, Marek Eliás, and Grigorios Koumoutsos. Weighted k-server bounds via combinatorial dichotomies. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '17, pages 493–504, 2017.

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

[6] 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. doi:10.1007/BF01294260.

[7] 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. doi:10.1145/3188745.3188798.

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

[9] Marek Chrobak, Howard Karloff, Tom Payne, and Sundar Vishwanathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, 1991.

[8] 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 $2k - 1$.

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

[11] 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. doi:10.4230/LIPIcs.ICALP.2017.14.

[12] 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. doi:10.1145/174130.174131.

[13] 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)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 126:1–126:14, 2017.

[14] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pages 448–455, 2003. doi:10.1145/780542.780608.

[15] 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.

[16] 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. doi:10.1109/FSCS.1990.89566.

[17] Andrew P. Kosoresow. *Design and analysis of online algorithms for mobile server applications.* PhD thesis, Stanford University, 1996.

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

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

[20] 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.

[21] 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.

[22] Christos H. Papadimitriou and Mihalis Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84(1):127–150, 1991. doi:10.1016/0304-3975(91)90263-2.

[23] Prabhakar Raghavan and Marc Snir. Memory versus randomization in on-line algorithms. *IBM Journal of Research and Development*, 38(6):683–708, 1994. doi:10.1147/rd.386.0683.

[24] H. Ramesh. On traversing layered graphs on-line. *J. Algorithms*, 18(3):480–512, 1995. doi:10.1006/jagm.1995.1019.