On the advice complexity of the k-server problem

On the Advice Complexity of the k-Server Problem

Hans-Joachim Böckenhauer[1], Dennis Komm[1],
Rastislav Královič[2], and Richard Královič[1]

[1] Department of Computer Science, ETH Zurich,
Universitätstrasse 6, 8092 Zurich, Switzerland
{hjb,dennis.komm,richard.kralovic}@inf.ethz.ch
[2] Department of Computer Science, Comenius University,
Mlynská dolina, 84248 Bratislava, Slovakia
kralovic@dcs.fmph.uniba.sk

Abstract. Competitive analysis is the established tool for measuring the output quality of algorithms that work in an online environment. Recently, the model of *advice complexity* has been introduced as an alternative measurement which allows for a more fine-grained analysis of the hardness of online problems. In this model, one tries to measure the amount of information an online algorithm is lacking about the future parts of the input. This concept was investigated for a number of well-known online problems including the k-server problem.
In this paper, we first extend the analysis of the k-server problem by giving both a lower bound on the advice needed to obtain an optimal solution, and upper bounds on algorithms for the general k-server problem on metric graphs and the special case of dealing with the Euclidean plane. In the general case, we improve the previously known results by an exponential factor, in the Euclidean case we design an algorithm which achieves a constant competitive ratio for a very small (i. e., constant) number of advice bits per request.
Furthermore, we investigate the relation between advice complexity and randomized online computations by showing how lower bounds on the advice complexity can be used for proving lower bounds for the competitive ratio of randomized online algorithms.

1 Introduction

Since many algorithmic setups deal with an environment where algorithms are needed which have to permanently produce chunks of output depending on an input they read continuously, we are interested in designing high-quality so-called *online algorithms*. These algorithms aim at having a high output quality without knowing the whole input at specific time steps, i. e., they may merely base their computations on the input they have read so far.

At first, we formally define the terms of an online algorithm and its competitive ratio which is usually used to measure the algorithm's output quality on some input.

Definition 1. *Consider an input sequence $I = (x_1, \ldots, x_n)$ for some minimization problem U. An* online algorithm $\mathtt{A}$ *computes the output sequence $\mathtt{A}(I) = (y_1, \ldots, y_n)$, where $y_i = f(x_1, \ldots, x_i)$ for some function f. The* cost *of the solution is given by* $\mathrm{cost}(\mathtt{A}(I))$. *By $\mathrm{Sol}_{\mathtt{A}} = \mathtt{A}(I)$ we denote a solution computed by $\mathtt{A}$ on I. An algorithm $\mathtt{A}$ is* c-competitive, *for some $c \geq 1$, if there exists a constant α such that, for every input sequence I, $\mathrm{cost}(\mathtt{A}(I)) \leq c \cdot \mathrm{cost}(\mathtt{Opt}(I)) + \alpha$, where $\mathtt{Opt}$ is an optimal offline algorithm for the problem. If $\alpha = 0$, then $\mathtt{A}$ is called* strictly c-competitive. *Finally, $\mathtt{A}$ is* optimal *if it is strictly* 1*-competitive.*

Note that Definition 1 can be easily adapted to include the case of maximization problems. The concept of the *competitive analysis* was introduced in [7] by Sleator and Tarjan. For a more detailed introduction to it and online algorithms in general, we refer to the standard literature, e. g., [2, 5].

However, comparing online algorithms to optimal solutions, which may only be constructed if the whole input is known in advance, does not seem very realistic because, by the nature of many real-world online scenarios, optimal results can never be reached. We want to get a deeper understanding of what online algorithms really lack, i. e., investigate what additional information we need to supply these algorithms with to increase their performance. The idea of *online algorithms with advice* was proposed in [3]. However, the original model used advice over a two letter alphabet $\{0, 1\}$ and an additional delimiter \$. In [1], a new model was introduced to prevent the hidden encoding of information by implicitly using the delimiter symbol. In this new model, such online algorithms are considered that are allowed to access an *advice tape* ϕ, which has an infinite number of bits written on it. These *advice bits* are calculated by an oracle which has access to the whole input before the online algorithm gets the first part of it. At any time step, an online algorithm $\mathtt{A}$ may then read, together with the chunk of input it gets, as many advice bits as needed. We use the same model in this paper.

Definition 2. *An* online algorithm $\mathtt{A}$ with advice *computes the output sequence* $\mathrm{Sol}_{\mathtt{A}}^{\phi} = \mathtt{A}^{\phi}(I) = (y_1, \ldots, y_n)$ *such that* y_i *is computed from* $\phi, x_1, \ldots, x_i$*, where* ϕ *is the content of the advice tape, i. e., an infinite binary sequence.* $\mathtt{A}$ *is* c-competitive with advice complexity $s(n)$ *if there exists a constant* α *such that, for every* n *and for each input sequence* I *of length at most* n*, there exists some* ϕ *such that* $\mathrm{cost}(\mathtt{A}^{\phi}(I)) \leq c \cdot \mathrm{cost}(\mathtt{Opt}(I)) + \alpha$ *and at most the first* $s(n)$ *bits of* ϕ *have been accessed during the computation of* $\mathtt{A}^{\phi}(I)$*.*

Emek et al. [4] proposed a different way to fix the model from [3] by restricting the online algorithm to read a fixed number of advice bits in every time step. With such an approach, it is not possible to analyze sublinear advice complexity, what is a serious issue for many online problems [1]. It is easy to simulate the model from [4] with our model, which is more general in this sense, and all lower bounds in our model directly carry over to the model from [4]. On the other hand, the upper bounds carry over from the model from [4] to our model. Furthermore, the model presented in [4] captures the idea that the advice is not available from the start, but comes gradually as more and more of the input is revealed. Although it seems that for concrete problems some bootstrapping techniques may be used to transform upper bounds from our model to the model from [4], the two models seem to be incomparable in general.

Our interest focuses on the number of advice bits $s(n)$ that are necessary [sufficient] to achieve optimality or a specific competitive ratio.

In this paper, we consider the problem k-SERVER in which k agents (so called *servers*) move in a metric space satisfying requests which appear in an online fashion. More formally, we look at the following problem.

Definition 3. *Let* $G = (V, E, d)$ *be a complete undirected weighted graph, where* V *is a (not necessarily finite) set of vertices,* $E = \{\{v, w\} \mid v, w \in V, v \neq w\}$ *is the set of edges, and* $d : E \to \mathbb{R}$ *is a* metric *cost function, that is,* d *satisfies the triangle inequality* $d(\{v, u\}) \leq d(\{v, w\}) + d(\{w, u\})$ *for every* $u, v, w \in V$*. Furthermore, we are given a set of* k *servers, residing in some vertices of the graph. Let* $C_i \subseteq V$ *be the multiset of vertices occupied by servers at time step* i*: A vertex occupied by* j *servers occurs* j*-times in* C_i*. We also call* C_i *the* configuration *at time step* i*. Then, a vertex* v_i *is requested and some servers may be moved yielding a new configuration* C_{i+1}*. The* request v_i *is*

satisfied *if, after this movement of servers, some server resides in v_i, i. e., if $v_i \in C_{i+1}$. The* distance *between two configurations C_1 and C_2 is given by the unique cost of a minimum-weight matching between C_1 and C_2.*

*The k-*SERVER *problem is the problem to satisfy all requests while minimizing the sum of the distances of all consecutive configurations.*

Although Definition 3 allows to place several servers to the same point, it is easy to see that this is not necessary; it is easy to modify any algorithm for k-SERVER such that it never places more than one server to one vertex and such that this modification do not increase costs of any solution produced by the algorithm.

The solution of k-SERVER is a sequence of configurations, and between two configurations, arbitrary number of servers can be moved. However, sometimes it is convenient to restrict to so called *lazy algorithms*, which move at most one server in response to each request. This can be done without loss of generality, as any algorithm for k-SERVER can be transformed to a lazy one without any increase in the costs of produced solutions [2]. It is easy to see that, for the case of lazy algorithms, the produced solutions can be uniquely described as a sequence of servers used to satisfy individual requests.

This paper is organized as follows: In Section 2, we give a lower bound on the number of advice bits needed to achieve optimality. Section 3 is devoted to designing an algorithm for the special case where the servers move on the Euclidean plane. The main result is presented in Section 4 where we give an algorithm for the general case which performs very well using only a constant number of bits per request. This algorithm is also valid in the restricted model from [4] and exponentially improves over the k-SERVER algorithm presented there. In Section 5, we relate the advice complexity to the model of randomized online algorithms and show how the advice complexity can be used to prove lower bounds on the competitive ratio achievable by randomized online algorithms.

For the ease of presentation, throughout this paper, we denote $\log_2$ by log.

2 A Lower Bound on the Optimality

In this section, we focus on the number of bits needed to obtain an optimal solution. Hence, we show that, if an online algorithm with advice `A` is optimal, there exist instances in which `A` needs to read large advice together with every request. First, we give a bound for inputs of fixed length k which we extend afterwards to instances of arbitrary length. Note that any graph with a cost function d that maps edges from $V \times V$ to values of 1 and 2 only, trivially respects the triangle inequality and is therefore a metric.

Lemma 1. *For any $k \in \mathbb{N}$, there exists an instance of the k-*SERVER *problem with k requests in total, for which any online algorithm* `A` *with advice needs at least $k(\log k - c)$ bits of advice to be optimal for some constant $c < 1.443$.*

Proof. Let $k \in \mathbb{N}$ and let $G = (U \cup W, E)$ be a complete bipartite graph with a metric cost function $d : E \to \{1, 2\}$, where $U = \{u_1, u_2, \ldots, u_k\}$ and $W = \{w_1, w_2, \ldots, w_{2^k}\}$. Since $|W| = 2^k$, we can define a bijective mapping $\mathrm{Set}\colon W \to 2^U$ which maps every vertex from W to a unique subset of vertices in U. We define the edge costs in G as follows: for $v \in U$ and $w \in W$, let

$$d(\{v, w\}) = \begin{cases} 2, & \text{if } v \in \mathrm{Set}(w) \\ 1, & \text{otherwise.} \end{cases}$$

Additionally, since formally the instance for the k-SERVER problem has to contain a complete weighted graph, we define the cost for all edges from $(U \times U) \cup (W \times W)$ to be 2. A schematic view of the constructed graph for $k = 4$ is shown in Figure 1. Let $G_i \subseteq W$ denote the vertices from W corresponding to subsets of U with exactly i elements, i. e., $G_i = \{w \in W \mid |\mathrm{Set}(w)| = i\}$. Clearly,

$$|G_i| = \binom{k}{i}.$$

We construct a class of instances $\mathcal{I}'$ in the following way. An instance $I \in \mathcal{I}'$ consists of a graph G as above, where each vertex of U is covered by a single server, and a sequence $(x_0, x_1, x_2, \ldots, x_{k-1})$ of requests such that

1. $x_j \in G_j$ and
2. $\mathrm{Set}(x_j) \subseteq \mathrm{Set}(x_{j+1})$.

Intuitively, the first requested vertex is the unique vertex from W with only cheap edges to U. Every following request has exactly one more expensive edge, chosen in such a way that the set of expensively connected vertices from U is extended by one vertex in each time step.

Clearly, we may represent I as a permutation π_I of $\{1, 2, \ldots, k\}$ in the following way:

$$\pi_I(j) = \mathrm{Set}(x_j) \setminus \mathrm{Set}(x_{j-1}) \text{ for } j = 1, 2, \ldots, k-1, \text{ i. e., } \pi_I(j) = y_j \in U,$$
$$\pi_I(k) = U \setminus \{\pi_I(j) \mid j = 1, 2, \ldots, k-1\}.$$

In other words, $\pi_I(j)$ denotes the index of that vertex from U which is connected to the requested vertex via an expensive edge from request x_j on. The unique optimal solution Sol_I for I with cost of exactly k can also be described by π_I in the following way. For every j, Sol_I satisfies the j-th request x_{j-1} by moving one server from some vertex from U, in particular the one in vertex $\pi_I(j)$, to the requested vertex from W, over a cheap edge. It is easy to see that there is no solution with cost of less than k, since all the servers start in U, all requests are unique vertices from W, and every edge has a cost of at least 1. To see why Sol_I is indeed the unique optimal solution, consider an offline environment where an optimal offline algorithm `Opt` receives the whole input at once and may satisfy the requests in an arbitrary order. It does so in the opposite order the requests are made: the last vertex requested is from the group G_{k-1} and there is one unique vertex $v_{\pi_I(k)}$ connected to it with a cheap edge. The vertex that was requested before is from G_{k-2}. Due to the construction, it also has a cheap edge to $v_{\pi_I(k)}$ and to a second vertex $v_{\pi_I(k-1)}$, so that `Opt` now uses this second edge. Following this strategy, it is immediately clear that `Opt` uses exactly k edges of cost 1 and that its strategy is the only one being no more expensive than k.

Since we may represent every instance from $\mathcal{I}'$ by a unique permutation of $\{1, 2, \ldots, k\}$, we need to distinguish $k!$ different cases. It remains to show that we also need a unique advice string for every input to be solved optimally by any online algorithm `A` with advice. Towards contradiction, let I_1 and I_2 be two different inputs from $\mathcal{I}'$. Suppose that `A` is optimal for both of these inputs. However, for the same advice string ϕ, the algorithm `A` behaves deterministically. Let us take the algorithm's point of view: In time step 1, the only vertex from G_0 is requested and `A` uses some server to satisfy this request. Then, in time step 2 it is revealed whether this was a good choice, i. e., if the server at $\pi_I(1)$ was used to serve the first request. After that, the algorithm chooses a

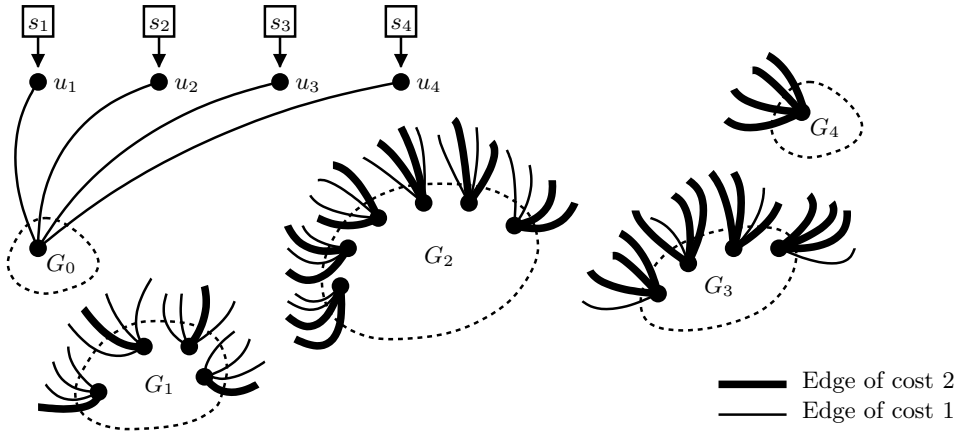

Fig. 1. A sample instance for 4-server as used in the proof of Lemma 1. Note that, for the ease of presentation, not all edges are shown completely.

second server to move and again, in time step 3, it is revealed whether this was a good choice, and so on. Suppose that the corresponding permutations of I_1 and I_2 differ at position j for the first time. This means that, in time step $j-1$, the algorithm has to make two different choices for the different inputs. But since it reads the same prefix of the input up to this point and furthermore uses the same advice string, it has to act the same way. But this directly implies that A cannot be optimal for both I_1 and I_2. We conclude that we need a different advice string for every instance and therefore $\log(k!)$ advice bits. Using Stirling's Formula, we get

$$\begin{aligned}\log(k!) &\geq \log\left(\sqrt{2\pi k}\left(\frac{k}{\mathrm{e}}\right)^k\right)\\ &= \frac{1}{2}\log 2\pi + \frac{1}{2}\log k + k(\log k - \log \mathrm{e})\\ &\geq k(\log k - c),\end{aligned}$$

where $c = \log \mathrm{e} < 1.443$, which concludes our proof.

We generalize this statement in the following theorem to an arbitrary number of requests. The idea is to use two instances as in the proof of Lemma 1, which are connected together. The optimal solution is forced to perform optimally within these two instances in an alternating way and any wrong step within any instance cannot be compensated later. We postpone the precise proof of the theorem to the appendix due to space limitations.

Theorem 1. *The number of bits necessary to enable* A *to be optimal can be bounded from beneath by* $n\left(\frac{1}{2}\log k - \frac{1}{2}c\right)$ *for some* $c < 1.443$*, where* n *is the number of requests.*

Please note that, if we allow the graph to have an unbounded size, it is easy to construct a lower bound of $n(\log k - \log \mathrm{e})$ by branching the graph infinitely often.

Note that all lower bounds presented in this chapter directly carry over also to the model of [4]. Furthermore, [4] presents a very simple upper bound of $\log k$ bits per request. Hence, the lower bound in Theorem 1 is tight up to a factor of 2.

3 An Upper Bound for the Euclidean Case

In this section, we restrict the k-SERVER problem to the case where the underlying metric space is the two-dimensional Euclidean plane. For this case, we propose a simple algorithm that achieves a constant competitive ratio with an advice of linear size (in particular, the algorithm reads a constant number of bits with every request).

Let us fix a parameter b such that the algorithm uses b bits of advice per request. The algorithm `A` works as follows: if the requested point is $r = (r_x, r_y)$, it divides the plane into 2^b disjoint segments $S_1, \ldots, S_{2^b}$ with their origin in r, and angle $\frac{2\pi}{2^b}$ each. Then it reads b bits of advice that identify a segment S_i, and serves the request greedily with the closest server from S_i. We show that `A` has a constant competitive ratio with linear advice.

Theorem 2. *For a fixed $b \geq 3$, `A` uses b bits of advice per request, and achieves a competitive ratio of at most $1/\left(1 - 2\sin\left(\frac{\pi}{2^b}\right)\right)$.*

Before proving Theorem 2, let us first show the following technical lemma which we need in the analysis.

Lemma 2. *For $0 < \alpha \leq \frac{\pi}{4}$, let $\mathcal{C}(\alpha, R)$ be the region of the Euclidean plane $\mathcal{C}(\alpha, R) = \{[r\cos\varphi, r\sin\varphi] \mid 0 < \varphi \leq \alpha, 0 < r \leq R\}$. For any point $p \in \mathcal{C}(\alpha, R)$ let*

$$f(p) = \frac{\|p\|}{R - \|[R, 0] - p\|},$$

where $\|v\|$ is the Euclidean length of v. Then $f(p)$ is maximized over region $\mathcal{C}(\alpha, R)$ for $p_{max} = R \cdot [\cos\alpha, \sin\alpha]$ and $f(p_{max}) = \frac{1}{1-2\sin\left(\frac{\alpha}{2}\right)}$.

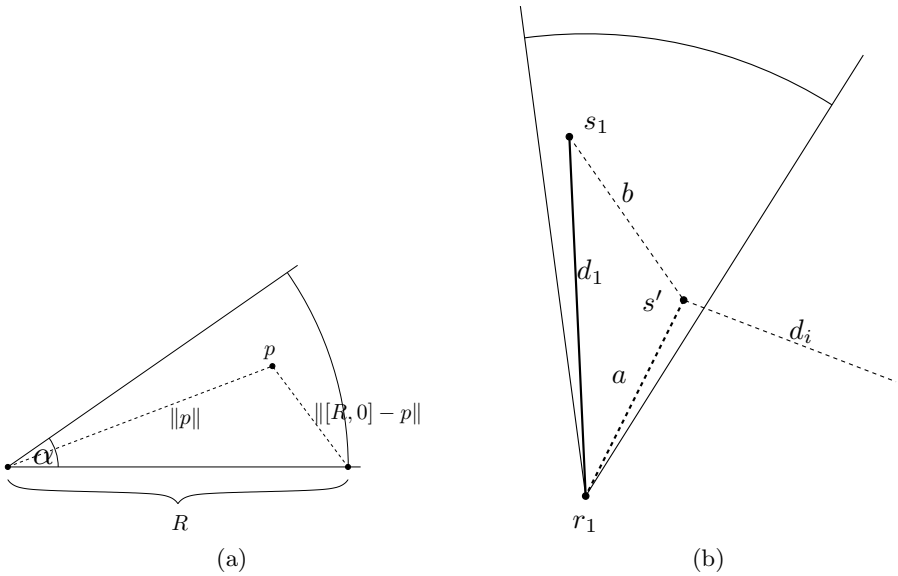

Fig. 2. (a) Illustration of Lemma 2. (b) Illustration of the algorithm.

The situation described by the Lemma is illustrated in Figure 2(a). The proof of the Lemma is very straightforward and is postponed to the appendix due to space restrictions.

Now we are ready to analyze the competitive ratio of A. Let us adopt the following notation: a configuration C is a multiset of k points that are occupied by the servers. A configuration $C_{p_1 \mapsto p_2}$ is obtained from C by moving a server from $p_1 \in C$ to p_2. An instance of the problem is a configuration, and a sequence of requests (points); the length of the instance is the number of requests. We restrict ourselves to lazy algorithms, hence, as already noted, a solution of an instance is a sequence of servers. To describe a server used to satisfy certain request, it is sufficient to specify the point occupied by this server, hence the solution can be described by a sequence of points as well. We prove the following theorem:

Theorem 3. *For any instance $I = (C, r_1, r_2, \ldots, r_n)$, and a solution Sol_I, the cost of* A *on I is at most $q \cdot \mathrm{cost}(\mathrm{Sol}_I)$ where $q = \frac{1}{1-2\sin\left(\frac{\pi}{2^b}\right)}$.*

Proof. Let Sol_I serve the i-th request r_i by a server located at s_i, with a cost of $d_i = \|s_i - r_i\|$. In the first request, A uses b bits to specify the segment of angle $\alpha = \frac{2\pi}{2^b}$ around r_1 in which s_1 is located, and moves the closest server in this segment, s' to r_1 incurring distance $a \leq d_1$. Hence, after the first request, Sol_I lead to configuration $C_{s_1 \mapsto r_1}$, whereas A is in configuration $C_{s' \mapsto r_1}$. This situation is illustrated in Figure 2(b).

The proof is done by induction on n. If $n = 1$, the cost of A is $a \leq d_1 = \mathrm{cost}(\mathrm{Sol}_I)$.

Let $n > 1$, and let r_i be the first request that is served by s' in Sol_I. Consider the instance $I' = (C_{s' \mapsto r_1}, r_2, \ldots, r_n)$; the sequence $(s_2, \ldots, s_{i-1}, s_1, s_{i+1}, \ldots, s_n)$ is a solution of I' with cost at most $b + \sum_{i=2}^{n} d_i$. By induction, the cost of A on I' is at most $q \cdot (b + \sum_{i=2}^{n} d_i)$, hence the cost of A on I is at most

$$a + q \cdot \left(b + \sum_{i=2}^{n} d_i \right).$$

Due to Lemma 2, $a \leq q(d_1 - b)$ and the result follows.

Theorem 3 immediately implies that A reaches competitive ratio q using b advice bits per request. For 3, 4, and 5 bits per request, the corresponding ratios are 4.261972632, 1.639829878, and 1.243834129. With $b \mapsto \infty$, the competitive ratio converges to 1.

4 An Upper Bound for the General Case

We now focus on the trade-off between the advice size and the competitive ratio achievable in general metric spaces.

Theorem 4. *For every $b \geq 2$, there is an online algorithm solving the k-*SERVER *problem that uses $b \cdot n$ advice bits for inputs with n requests and achieves competitive ratio $2 \left\lceil \frac{\lceil \log k \rceil}{b-1} \right\rceil \leq 4 + \frac{2}{b-1} \log k$.*

Proof. At first, we fix the algorithm A. With each request, A reads one bit of advice called a *control bit*. If this bit is zero, A satisfies the request greedily with the nearest server. Afterwards, the used server is returned to its original position before the next request.

If the control bit is one, A reads the next $\log k$ bits. These bits specify which server should be used to satisfy the request. After the request is satisfied, the server is left at its new position.

We prove that, for every input instance, there exists an advice such that A has competitive ratio $r := 2\left\lceil \frac{\lceil \log k \rceil}{b-1} \right\rceil \leq 4 + \frac{2}{b-1}\log k$. Furthermore, A uses at most bi bits of advice while processing the first i requests.

Consider any input instance I consisting of n requests $r_1, r_2, \ldots, r_n$. Let $S^{(0)}$ be an optimal solution of I; assume that $S^{(0)}$ satisfies request i with cost c_i. We construct a sequence $S^{(1)}, \ldots, S^{(n)}$ of feasible solutions of I sequentially, together with a sequence of advice tapes $\phi^{(1)}, \ldots, \phi^{(n)}$, with the following properties:

1. Let $j \leq i$. Solution $S^{(i)}$ satisfies the request j with cost $c_j^{(i)}$ which is at most twice the sum of some request costs of $S^{(0)}$, i. e.,
$$c_j^{(i)} \leq 2 \sum_{k \in D_j^{(i)}} c_k,$$
where $D_j^{(i)}$ is some subset of $\{1, \ldots, n\}$. We call the j-th step of solution $S^{(i)}$ *dead step*. Similarly, we call the set $D_j^{(i)}$ *dead set*, and the cost $c_j^{(i)}$ *dead cost*.
2. Let $j > i$. Similarly to the previous case, solution $S^{(i)}$ satisfies the request j with cost $c_j^{(i)}$ which is at most *once* the sum of some request costs of $S^{(0)}$, i. e.,
$$c_j^{(i)} \leq \sum_{k \in D_j^{(i)}} c_k,$$
where $D_j^{(i)}$ is some subset of $\{1, \ldots, n\}$. We call the j-th step of solution $S^{(i)}$ *live step*. Similarly, we call the set $D_j^{(i)}$ *live set*, and the cost $c_j^{(i)}$ *live cost*.
3. Let i be arbitrary but fixed. For every j, request cost c_j contributes to at most $r/2$ costs $c_k^{(i)}$, i. e., j belongs to at most $r/2$ sets $D_k^{(i)}$.
4. Let i be arbitrary but fixed. For every j, request cost c_j contributes to at most one live cost $c_k^{(i)}$, i. e., j belongs to at most one set $D_k^{(i)}$ such that $k > i$.
5. Consider any live step j of $S^{(i)}$ (i. e., $j > i$). The request r_j is satisfied by $S^{(i)}$ by moving a single server to vertex r_j, without any other moves (i. e., the moved server is left at r_j).
6. For any $j \leq k$ and any i, the sets $D_j^{(i)}$ and $D_k^{(i)}$ are either disjoint or $D_j^{(i)} \subseteq D_k^{(i)}$. Note that disjointness of live sets is already guaranteed by Property 4.
7. Consider an arbitrary i. Algorithm A, given advice $\phi^{(i)}$, processes the first i requests identically as solution $S^{(i)}$.

It is not difficult to see that Properties 1–3 ensure that, for any i, the total cost of $S^{(i)}$ does not exceed the total cost of $S^{(0)}$ multiplied by $2 \cdot r/2$. Property 7 ensures that A, given the advice $\phi^{(n)}$, processes all requests identically as $S^{(n)}$, hence A reaches competitive ratio r. In the sequel, we show how to construct the sequence of solutions $S^{(1)}, \ldots, S^{(n)}$, as well as the sequence of advice tapes $\phi^{(1)}, \ldots, \phi^{(n)}$. Afterwards, we analyze how many advice bits are necessary for this construction.

We define $S^{(i)}$ and $\phi^{(i)}$ inductively. Each $\phi^{(i)}$ contains some finite defined prefix followed by arbitrary (undefined) bits. For $i = 0$, the defined prefix of $\phi^{(0)}$ is empty.

It is easy to see that $S^{(0)}$ and $\phi^{(0)}$ satisfy Properties 1–7; it is sufficient to define $D_j^{(0)} := \{j\}$.

Assume that $S^{(i-1)}$ and $\phi^{(i-1)}$ are defined. We obtain $S^{(i)}$ by applying a few modifications to $S^{(i-1)}$. In particular, the first $i-1$ steps of $S^{(i)}$ are always identical to $S^{(i-1)}$. The live i-th step of $S^{(i-1)}$ needs to be transformed into a dead i-th step of $S^{(i)}$. While doing so, some other live steps might be changed as well. Advice tape $\phi^{(i)}$ is obtained by taking the defined prefix of $\phi^{(i-1)}$ and appending the information necessary for `A` to perform the i-th step in the same way as $S^{(i)}$ does.

We consider two cases:

Case 1 *For any $k \in D_i^{(i-1)}$, there are less than $r/2$ sets $D_j^{(i-1)}$ containing k.* Assume that request r_i is satisfied in solution $S^{(i-1)}$ by moving a server from vertex v to r_i. (Property 5 ensures that this is always the case.) In $S^{(i)}$, we replace this step by a greedy step, i. e., the request r_i will be satisfied by the nearest server located in vertex w, and the server will be returned to w afterwards. Because we took the closest server, $c_i^{(i)} \leq 2c_i^{(i-1)}$, hence it is sufficient to define $D_i^{(i)} := D_i^{(i-1)}$. The next steps of $S^{(i)}$ can be defined to be identical to the steps of $S^{(i-1)}$, until the solution $S^{(i-1)}$ uses the server located in r_i to satisfy some request r_j. Since there is no server located in r_i in $S^{(i)}$, we need to modify this step: In $S^{(i)}$, the request r_j is satisfied by moving the server from v into r_i and then to r_j. This is possible, because the server at v was not used after request r_i in $S^{(i-1)}$. The cost $c_j^{(i)}$ of such move is at most $c_i^{(i-1)} + c_j^{(i-1)}$, hence it is sufficient to define $D_j^{(i)} := D_i^{(i-1)} \cup D_j^{(i-1)}$ (note that Property 4 applied for $S^{(i-1)}$ ensures that this union is disjoint). All remaining steps of $S^{(i)}$ are the same as in $S^{(i-1)}$.

It is easy to see that Properties 1, 2, and 5 hold for $S^{(i)}$. After transforming $S^{(i-1)}$ into $S^{(i)}$, only the elements in $D_i^{(i-1)}$ occur in more sets. Hence, due to assumption of this case, Property 3 holds. Since, due to Property 4 applied for $S^{(i-1)}$, no element of $D_i^{(i-1)}$ is in $D_k^{(i-1)}$ for any $k > i$, and since $D_i^{(i)}$ is not a live set, Property 4 holds for $S^{(i)}$ as well. Since Property 4 holds for $S^{(i)}$, Property 6 could be violated only if $D_j^{(i)}$ was not superset of some $D_k^{(i)}$ for $k \leq i$. This, however, cannot happen because $D_j^{(i)} \subseteq D_i^{(i)}$ and Property 6 holds for $S^{(i-1)}$.

To satisfy Property 7, it is sufficient to extend the advice $\phi^{(i-1)}$ by a single bit – a control bit 0.

Case 2 *Some element of $D_i^{(i-1)}$ is contained in $r/2$ sets $D_j^{(i-1)}$.* In this case, the solution $S^{(i)}$ is defined to be exactly the same as $S^{(i-1)}$. Hence, Properties 1–6 hold trivially. In order to satisfy Property 7, we extend the advice $\phi^{(i-1)}$ by $1 + \lceil \log k \rceil$ bits: a control bit 1 followed by $\lceil \log k \rceil$ bits describing which server should be used to satisfy request r_i. Note that, in this case, no element $k \in D_i^{(i)}$ occurs in any live set $D_j^{(i)}$. Furthermore, k cannot reappear later, i. e., k does not occur in any $D_j^{(i')}$ for $i' \geq i$, $j > i$.

Properties 1–7 ensure that `A`, given advice $\phi^{(n)}$, solves I correctly with competitive ratio r. It remains to analyze the length of the advice consumed by `A`.

The construction of the advice tapes easily implies that `A`, given advice $\phi^{(i)}$, accesses only the defined prefix of $\phi^{(i)}$ during the first i steps of computation. Next, we show that the defined prefix of $\phi^{(i)}$ has length at most bi.

It is easy to see that $\phi^{(i)}$ contains exactly i control bits. Consider any block of $\lceil \log k \rceil$ consecutive non-control bits appearing in $\phi^{(i)}$. These bits were created in Case 2 when

performing the j-th step of induction (i. e., when $S^{(j)}$ and $\phi^{(j)}$ were created). This, however, means that some element of $D_j^{(j-1)}$ occurs in $r/2-1$ sets $D_k^{(j-1)}$ $(k<j)$. For any such k, Case 1 occurred for the k-th step of induction (as noted in Case 2, if that case occurred, no element of $D_k^{(k)} = D_k^{(j-1)}$ would occur in $D_j^{(j-1)}$, a contradiction). Hence, to any block of $\lceil \log k \rceil$ consecutive non-control bits appearing in $\phi^{(i)}$, we can injectively assign $(r/2-1)$ induction steps where no non-control bits were created. The fact that the assignment is injective is guaranteed by Property 6 and the fact that elements in a set $D_j^{(j-1)}$ handled in Case 2 cannot reappear later. Equivalently, we can injectively assign $r/2$ control bits to every block of $\lceil \log k \rceil$ non-control bits. Thus, there are at most

$$i \cdot \frac{\lceil \log k \rceil}{\frac{r}{2}} = i \cdot \frac{2\lceil \log k \rceil}{2\left\lceil \frac{\lceil \log k \rceil}{b-1} \right\rceil} \leq i \cdot \frac{\lceil \log k \rceil}{\frac{\lceil \log k \rceil}{b-1}} = i(b-1)$$

non-control bits in $\phi^{(i)}$. To sum up, $\phi^{(i)}$ contains at most $i(b-1)+i = ib$ defined bits.

We have proven Theorem 4 in the model of an advice tape. Nevertheless, the result of this theorem can be easily adapted to the model with fixed number of advice bits received with each request, as used in [4]. In this case, consider the advice tape created in the proof of Theorem 4. Separate this tape into two bit sequences, one containing only control bits, the other containing non-control bits only. Afterwards, interleave these bits, always taking $b-1$ non-control bits followed by a single control bit to create a single b-bit advice message. Algorithm `A` then reads $b-1$ non-control bits with every message and stores them into a FIFO data structure. Afterwards, it reads the control bit; if this bit is 1, $\lceil \log k \rceil$ bits are extracted from the FIFO. The proof of Theorem 4 ensures that there are always enough bits stored in the FIFO when a control bit 1 arrives.

Note that Theorem 4 improves the result of [4] exponentially: With b bits per request, a feasibility of competitive ratio of $k^{\Theta(\frac{1}{b})}$ was proven in [4], while Theorem 4 proves competitive ratio of $\log\left(k^{\Theta(\frac{1}{b})}\right)$.

5 Relation Between Randomization and Advice

One intriguing aspect of the advice-based analysis of online problems is the relationship of advice and randomization. Specifically, one can ask if there is some connection between the existence of an efficient randomized algorithm and the existence of an efficient algorithm with small advice. One can view the randomized algorithm as randomly selecting one witness object from a universe of possible witnesses. To achieve a good performance, the universe must be constructed in such a way that, for any input, there are many reasonably good witnesses. On the other hand, the advice can be interpreted as selecting a particular witness from a given universe. Hence, for algorithms with advice it is necessary to have a small universe such that for any input at least one good witness exists.

From Yao's minimax principle [8] it follows that, if there is a randomized algorithm with a given expected cost, then there is a small universe of witnesses such that for each input there is a reasonably good witness, as stated in the following theorem.

Theorem 5. *Consider a minimization online problem P such that there are $|\mathcal{I}(n)| = I(n)$ possible inputs of length n, where $\mathcal{I}(n)$ is the set of all possible inputs of length n. Furthermore, let us suppose that there is a randomized algorithm `R` with worst-case expected competitive ratio at most $E(n)$. Then, for any fixed $\varepsilon > 0$, it is possible to*

construct a deterministic algorithm A *with advice* $\log n + 2\log\log n + \log\left(\frac{\log I(n)}{\log(1+\varepsilon)}\right)$ *with competitive ratio* $E(n)(1+\varepsilon)$.

Proof. Let us suppose that R uses $r(n)$ random bits, so there are $R(n) = 2^{r(n)}$ possibilities for a random string, which is the only source of randomness used by R. Let us construct a matrix A with $I(n)$ rows and $R(n)$ columns such that $A_{x,q}$ is the competitive ratio of R on input x with random string q. We show how to construct a set of strings $\mathcal{W}$ containing $\frac{\log I(n)}{\log(1+\varepsilon)}$ elements such that, for each input x, there is a string $w \in \mathcal{W}$ such that the competitive ratio of R on input x with random string w is at most $(1+\varepsilon)E(n)$. The first part of the advice used by A is the number of requests n encoded in a self-delimited form using $\log n + 2\log\log n$ bits. Such an encoding can be achieved by writing the value of $\log\log n$ in unary base, extending it by the value of $\log n$ in binary base, which uses $\log\log n$ digits, and finally extending it by the value of n in binary base, which uses $\log n$ digits. Algorithm A knows a-priori the set $\mathcal{W}$, since it depends on n only. The second part of the advice for a given input is the index of the particular string $w \in \mathcal{W}$.

Since the expected competitive ratio of R for any input is at most $E(n)$, we have

$$\forall x \in \mathcal{I}(n)\colon \frac{1}{R(n)} \sum_{q} A_{x,q} \le E(n).$$

Hence, the overall sum in the matrix is at most $R(n) \cdot I(n) \cdot E(n)$, and

$$\exists w\colon \sum_{x \in \mathcal{I}} A_{x,w} \le I(n) \cdot E(n). \tag{1}$$

This w will be included in $\mathcal{W}$, and will be used as advice for any input x where $A_{x,w} \le (1+\varepsilon)E(n)$; let us call such x a *hit* for w. We argue that w has many hits: for a given number s of hits we have

$$\sum_{x \in \mathcal{I}} A_{x,w} > (I(n) - s)\,(1+\varepsilon)E(n). \tag{2}$$

From (1) and (2) we get $s > I(n)\frac{\varepsilon}{1+\varepsilon}$. Now we have covered $I(n)\frac{\varepsilon}{1+\varepsilon}$ inputs by one string w. Consider the sub-matrix of A obtained by removing the column w and the rows corresponding to the already covered inputs. Since, for all remaining $I(n)\frac{1}{1+\varepsilon}$ inputs, the string w contributed more than $E(n)$ to the row-sum, the average of any row in the sub-matrix is still at most $E(n)$.

We can repeat the above argument, every time covering a fraction of $\frac{\varepsilon}{\varepsilon+1}$ of the inputs. Hence, after $\frac{\log I(n)}{\log(1+\varepsilon)}$ steps, all possible inputs are covered by some $w \in \mathcal{W}$.

Please note that the algorithm constructed in the previous example is not polynomial: in order to decode the advice, the algorithm has to construct the dictionary $\mathcal{W}$ by simulating the randomized algorithm on all inputs of length n.

In general, this bound cannot be extended to the case $\varepsilon = 0$, as can be seen by the following (rather artificial) "winner-takes-all" problem:

Definition 4 (Winner-takes-all problem). *The input consists of a sequence of requests* $x_1 = 0, x_2, \ldots, x_{n+1}$, *where* $x_i \in \{0,1\}$. *The solution is a sequence* $y_i \in \{0,1\}$. *The cost of a given solution is* 1, *if* $y_i = x_{i+1}$ *for all* $i \in \{1,\ldots,n\}$, *and* 2 *otherwise. Hence, the optimal solution has cost* 1 *and all other solutions pay an extra penalty of* 1.

It is easy to show that the best possible randomized algorithm is to guess each bit with equal probability, so the expected cost is $2 - \frac{1}{2^n}$. Obviously, any algorithm with advice less than n bits has worst-case performance 2, so n bits are needed to be on par with randomization.

Theorem 5 relates the advice and randomization in such a way that lower bounds on advice complexity can be used to deliver lower bounds on randomization. Specifically, for the k-server problem, the famous "randomized k-server" conjecture (RKSC) (for a survey see, e. g., [6]) states that, for any metrical space, there is a randomized online algorithm with competitive ratio $\mathcal{O}(\log k)$. This would imply that for a (discrete) metric space with M elements, there is an algorithm with advice $\mathcal{O}(\log n + \log\log\log M)$ that is $\mathcal{O}(\log k)$-competitive. Our results, however, grant the $\mathcal{O}(\log k)$ competitive ratio only by using $\mathcal{O}(n)$ advice bits. Hence, if one believes the RKSC, the upper bound is still exponentially far from the optimum. On the other hand, any lower bound of the form $\omega(\log n)$ would disprove the RKSC. Compare this to the specific case of k-server [1] in a space where all distances are 1 (i. e., the paging problem) where it is known that linear advice is needed for optimality, and constant (i. e., independent of n) advice is sufficient for being $\mathcal{O}(\log k)$-competitive.

References

1. H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič, and T. Mömke. On the advice complexity of online problems. In Y. Dong, D.-Z. Du, and O. H. Ibarra, editors, *Proc. of the 20th International Symposium on Algorithms and Computation (ISAAC 2009)*, volume 5878 of *Lecture Notes in Computer Science*, pages 331–340. Springer-Verlag, 2009.
2. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
3. S. Dobrev, R. Královič, and D. Pardubská. How much information about the future is needed? In V. Geffert, J. Karhumäki, A. Bertoni, B. Preneel, P. Návrat, and M. Bieliková, editors, *Proc. of the 34th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2008)*, volume 4910 of *Lecture Notes in Computer Science*, pages 247–258, Berlin, 2008. Springer-Verlag.
4. Y. Emek, P. Fraigniaud, A. Korman, and A. Rosén. Online computation with advice. In S. Albers, A. Marchetti-Spaccamela, Y. Matias, S. E. Nikoletseas, and W. Thomas, editors, *Proc. of the 36th International Colloquium on Automata, Languages and Programming (ICALP 2009)*, volume 5555 of *Lecture Notes in Computer Science*, pages 427–438. Springer-Verlag, 2009.
5. J. Hromkovič. *Design and Analysis of Randomized Algorithms: Introduction to Design Paradigms (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag, New York, 2005.
6. E. Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105 – 118, 2009.
7. D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.
8. A. C.-C. Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *FOCS*, pages 222–227. IEEE, 1977.

Appendix

Proof (Theorem 1). We now create the class of instances $\mathcal{I}$. Suppose we take two disjoint graphs $(U_1 \cup W_1, E_1)$ and $(U_2 \cup W_2, E_2)$ as used for constructing instances of $\mathcal{I}'$ in the proof of Lemma 1. We then connect all vertices from W_1 to the vertices of $U_2 = \{u_{2,1}, u_{2,2}, \ldots, u_{2,k}\}$ such that, for $1 \leq j \leq k$, the edges from G_j to $u_{2,j+1}$ have costs 1 and all other edges have costs 2. The vertices from W_2 are connected to the vertices of U_1 in the same manner. All other newly added edges are assigned costs 2. Let there be n requests in total and let n be a multiple of $4k$. At the beginning, the servers are located at the vertices from U_1. For any instance from $\mathcal{I}$, the first k requests are vertices from W_1 that correspond to a permutation π_1 as in the proof of Lemma 1. After that, there are k consecutive requests from U_2 where each of the vertices is requested exactly once. Next, k vertices from W_2 are requested according to some permutation π_2 followed by k requests of the k vertices from U_1. We continue in this fashion until we have made n requests in total. Each such input can be written as

$$\pi_1 \cdot (a_1, a_2, \ldots, a_k) \cdot \pi_2 \cdot (b_1, b_2, \ldots, b_k) \cdot \pi_3 \cdot (a_1, a_2, \ldots, a_k) \cdot \pi_4 \cdot \ldots \cdot \pi_{n/(2k)} \cdot (b_1, b_2, \ldots, b_k)$$

where π_{2i-1} is a subset of vertices from W_1, π_{2i} is a subset of vertices from W_2, and $\{a_1, a_2, \ldots, a_k\} = U_2$, $\{b_1, b_2, \ldots, b_k\} = U_1$. Let us call the $4k$ consecutive requests from W_1, U_2, W_2, and U_1 a round. These instances have an optimal solution $\mathrm{Sol}_{\mathtt{Opt}}$ which, whenever moving servers from U_1 to W_1 [U_2 to W_2], acts according to the corresponding permutation, and moves the unique server residing in G_j to $u_{2,j+1}$ when moving servers from W_2 to U_1 [W_1 to U_2].

Let us now show that $\mathrm{Sol}_{\mathtt{Opt}}$ is the unique optimal solution. Towards contradiction, suppose that there exists solution $\mathrm{Sol}_{\mathtt{A}}$ that differs from $\mathrm{Sol}_{\mathtt{Opt}}$ and is not worse than $\mathrm{Sol}_{\mathtt{Opt}}$. Clearly, $\mathrm{Sol}_{\mathtt{Opt}}$ has costs of 1 in every time step. Let us first show that there are no time steps in which $\mathrm{Sol}_{\mathtt{A}}$ may have costs 0, i. e., in which it does not move any server. By the construction of the inputs, the same vertex is not requested twice within the same round. This means that, if in some round a server already resides at some vertex that is now requested, this server was not moved since the last round (Note that, due to the triangle inequality, it is never helpful to move a server without satisfying a request with this move.) Without loss of generality, suppose that this server is positioned at some vertex from W_1. We know that $\mathrm{Sol}_{\mathtt{Opt}}$ used this server to satisfy a request from U_2, W_2, and U_1 afterwards which each causes cost 1. Instead, $\mathrm{Sol}_{\mathtt{A}}$ uses a server that already is located at some vertex from U_2 to satisfy some request from U_2 yielding cost 2 in this time step, because all vertices from U_2 to U_2 are connected by edges of costs 2. The same applies for at least one request from W_2 and U_1 and therefore, $\mathrm{Sol}_{\mathtt{A}}$ saved cost 1, but therefore paid 6 instead of 3 after one round is over. For $\mathrm{Sol}_{\mathtt{A}}$, it therefore follows that it also moves exactly one server per time step which causes it to pay costs 1. Now let i denote the first time step in which $\mathrm{Sol}_{\mathtt{A}}$'s strategy deviates from $\mathrm{Sol}_{\mathtt{Opt}}$.

For the ease of presentation, we only consider the following two cases. All other possible cases are handled analogously.

Case 1 $\mathrm{Sol}_{\mathtt{Opt}}$ *moves a server from* U_1 *to* W_1 *at time step* i. If $\mathrm{Sol}_{\mathtt{A}}$ uses a server from W_1 to satisfy this request, $\mathrm{Sol}_{\mathtt{A}}$ pays 2 instead of 1 which, as we just showed, cannot be compensated afterwards. Due to Lemma 1, we already know that, if $\mathrm{Sol}_{\mathtt{A}}$ chooses a server from U_1 other than the one taken by $\mathrm{Sol}_{\mathtt{Opt}}$, this also causes costs 2 eventually. Since i is the first time step in which the two solutions differ, it follows that no servers are located at U_2 and W_2. Thus, this case leads to a contradiction to the optimality of $\mathrm{Sol}_{\mathtt{A}}$ and hence cannot occur.

Case 2 $\mathrm{Sol}_{\mathtt{Opt}}$ *moves a server from* W_1 *to* U_2 *at time step* i. Using a server located at U_2 again causes cost 2. Moreover, observe that there is only one unique group G_j in W_1 which is connected to $u_{2,j+1}$ with edges of costs 1. Since $\mathrm{Sol}_{\mathtt{A}}$ and $\mathrm{Sol}_{\mathtt{Opt}}$ acted identically up to this point, before the first request from U_2 arrived in this round, they both positioned exactly one server at each group G_k. It directly follows that, if $\mathrm{Sol}_{\mathtt{A}}$ uses a different server than $\mathrm{Sol}_{\mathtt{Opt}}$, it again pays 2 instead of 1. Similarly as above, no servers may be located at U_1 or W_2.

Since there is one unique optimal solution which needs to act according to the permutations $\pi_1, \pi_2, \ldots, \pi_{n/(2k)}$, by the same argumentation as in the proof of Lemma 1, it directly follows that at least

$$\frac{n}{2k}\log(k!) \geq \frac{n}{2}(\log k - c)$$

bits of advice are necessary in total to be optimal for $c = \log \mathrm{e}$. It directly follows that at least

$$\frac{1}{2}\log k - \frac{1}{2}c$$

bits of advice are necessary in every time step.

Proof (Lemma 2). Consider any point $p \in \mathcal{C}(\alpha, R)$ such that $p = [r\sin\varphi, r\cos\varphi]$. Then

$$f(p) = \frac{r}{R - \sqrt{(R - r\cos\varphi)^2 + (r\sin\varphi)^2}} = \frac{r}{R - \sqrt{r^2 - 2Rr\cos\varphi + R^2}}.$$

For $0 < \alpha \leq \frac{\pi}{4}$, $f(p)$ is increasing in φ, and the maximum is attained for $\varphi = \alpha$. For the rest of the proof, let us treat $f(p)$ as a function of r only. Since the zeroes of the denominator are $\{0, 2R\cos\alpha\}$, and $0 < \alpha \leq \frac{\pi}{4}$, $f(p)$ is continuous for $r \in \langle 0, R\rangle$. Taking the derivative one gets

$$f'(p) = R\frac{X + r\cos\alpha - R}{X(R - X)^2}$$

where $X = \sqrt{r^2 - 2Rr\cos\alpha + R^2}$. The denominator is always positive, hence it is sufficient to prove

$$X = \sqrt{r^2 - 2Rr\cos\alpha + R^2} > R - r\cos\alpha$$

which holds, since $r^2(1 - \cos^2\alpha) > 0$. Hence, $f(p)$ is increasing in r, and the lemma follows.