On Online Algorithms with Advice for the k-Server Problem

Marc P. Renault[1] and Adi Rosén[2] *

[1] LIAFA, Univerité Paris Diderot - Paris 7; and UPMC, mrenault@liafa.jussieu.fr
[2] CNRS and Univerité Paris Diderot - Paris 7, adiro@lri.fr

Abstract. We consider the model of online computation with advice [5]. In particular, we study the k-server problem under this model. We prove two upper bounds for this problem. First, we show a $\left\lceil \frac{\lceil \log k \rceil}{b-2} \right\rceil$-competitive online algorithm for general metric spaces with b bits of advice per request, where $3 \leq b \leq \log k$. This improves upon the recent result of [1]. Moreover, we believe that our algorithm and our analysis are more intuitive and simpler than those of [1]. Second, we give a 1-competitive online algorithm for trees which uses $2 + 2\lceil \log(p+1) \rceil$ bits of advice per request, where p is the caterpillar dimension of the tree.

Keywords: online computation with advice, k-server problem, online algorithms, competitive analysis

1 Introduction

Online algorithms have been the subject of intense research activity over the past decades. The traditional setting is that of an online algorithm that does not have any knowledge about the future and that of a worst-case analysis using competitive analysis (cf. [3]). In the present paper we consider a model recently introduced by Emek et al. [5], dubbed *online computation with advice*, which is aimed at relaxing the "absolutely no knowledge about the future" setting and at giving a general framework to quantify the interplay between the amount of knowledge about the future and the possible improvement in the competitive ratio. Roughly speaking, this model augments the power of the online algorithm by a series of queries. Each query is issued by the online algorithm when it receives a new request. These queries map the whole request sequence, including future requests, to some domain of *advice*. Thus, they provide the online algorithm with some information about the future. One is typically interested in the interplay between the size of the domain of advice, i.e., how many bits of advice are received with each request, and the attainable competitive ratio. For a formal definition of this model, see Section 2.

A number of results for various online problems have been obtained in the above model and in a variant thereof introduced by Böckenhauer et al. [2]. In

* Research supported in part by ANR projects QRAC and ALADDIN.

the present paper, we consider the k-server problem under the model of online computation with advice. Emek et al. [5] gave an upper bound of $k^{O(\frac{1}{b})}$ on the competitive ratio of deterministic algorithms on general metric spaces, where b is the number of bits of advice per request. This upper bound was recently improved to $2\left\lceil \frac{\lceil \log k \rceil}{b-1} \right\rceil$ by Böckenhauer et al. [1]. Better bounds for specific metric space where also given (see the paragraph "Related Work" below).

In this paper, we improve the upper bound for deterministic k-server algorithms with advice on general metric spaces by giving a deterministic online algorithm with b bits of advice per request, for $b \geq 3$, whose competitive ratio is $\left\lceil \frac{\lceil \log k \rceil}{b-2} \right\rceil$. While the improvement over the previous result is only about a factor of 2, we believe that our algorithm and analysis are more intuitive and simpler than previous ones, and may lead to further improvements in the upper bound. Also, we consider the class of metric spaces of finite trees, and give a 1-competitive deterministic online algorithm. The number of bits of advice per request used by this algorithm is $2 + 2\lceil \log(p+1) \rceil$, where p is the caterpillar dimension of the tree (cf. [8]). We use this measure for the tree since it is at most the height of the tree, and it is at most $\log N$, where N is the number of nodes in the tree [8]. This measure is preferable over other measures, such as height, because it remains constant for degenerate trees, such as the line, the spider and the caterpillar.

Related Work The model of online computation with advice considered in the present paper was introduced by Emek et al. [5]. In that paper, the authors gave tight bounds of $\Theta(\log n/b)$ for deterministic and randomized online algorithms with advice for Metrical Task Systems, where n is the number of states of the systems and b is the number of bits of advice per request. They also gave a deterministic online algorithm with advice for the k-server problem which is $k^{O(\frac{1}{b})}$-competitive, where $\Theta(1) \leq b \leq \log k$. This was improved by Böckenhauer et al. [1] who gave a deterministic online algorithm with advice for general metric spaces with a competitive ratio of $2\left\lceil \frac{\lceil \log k \rceil}{b-1} \right\rceil$. Böckenhauer et al., also, gave a deterministic algorithm for the Euclidean plane with a competitive ratio of $\frac{1}{1-2\sin(\frac{\pi}{2^b})}$, where $b \geq 3$ is the number of bits of advice per request. For the uniform metric space (the problem of paging), a 1-competitive deterministic online algorithm with 1 bit of advice per request is implicit in [4].

Böckenhauer et al. [2] introduced a somewhat similar model for online algorithms with advice, where the advice is a single tape of bits instead of being given separately for each request. This allows an algorithm to read a different number of bits of advice per request, but it requires that the online algorithm knows how many bits of advice to read with each request. Thus, the two models are, in general, incomparable. We note that upper bounds in the model of [5], as those given in the present paper, carry over to the model of [2]. Several results were given in this model [4, 2, 6, 7, 1]. For example, in [4, 2], the authors explore the number of bits of advice required for deterministic and randomized paging algorithms, scheduling algorithms and routing algorithms to be 1-competitive.

2 Preliminaries

Online algorithms receive their input piece by piece. Each piece, or request, is an element of some set R, and the algorithm receives a *request sequence* denoted $\sigma = r_1, \ldots, r_n$, where $n = |\sigma|$ and r_i is the ith request. An online algorithm must perform all of the actions pertaining to a request before receiving the subsequent requests. These actions incur some cost to the online algorithm. In this paper, we consider only minimization problems.

We use the definition of *deterministic online algorithms with advice* as presented in [5]. An online algorithm with advice is defined as a request-answer game that consists of a request set, R; a sequence of finite nonempty answer sets, $A_1, A_2, \ldots$; and a sequence of cost functions, $\text{cost}_n : R^n \times A_1 \times A_2 \times \cdots \times A_n \rightarrow \mathbb{R}^+ \cup \{\infty\}$ for $n = 1, 2, \ldots$. In addition, online algorithms with advice have access via a query to an advice space, U, which is a finite set. The advice space has a size of 2^b, where $b \geq 0$ is the number of bits of advice provided to the algorithm with each request. With each request, the online algorithm receives some advice that is defined by a function, $u_i : R^* \rightarrow U$, that is applied to the whole request sequence, including future requests. A deterministic online algorithm with advice can, thus, be represented as a sequence of pairs (g_i, u_i), where $g_i : R^i \times U^i \rightarrow A_i$ for $i = 1, 2, \ldots$. The action that the online algorithm takes upon receiving request r_i is a function of the first i requests, $r_1, \ldots, r_i$, and the advice received so far, $u_1(\sigma), \ldots, u_i(\sigma)$.

The cost of the online algorithm is defined as $\text{ALG}(\sigma) = \text{cost}_n(\sigma, \text{ALG}[\sigma])$, where $\text{ALG}[\sigma] = < a_1, \ldots, a_n > \in A_1 \times \cdots \times A_n$ and $a_j = g_j(r_1, \ldots, r_j, u_i, \ldots, u_j)$ for $j = 1, \ldots, n$. At the risk of a slight abuse of notation, we will denote the cost of a subsequence of σ as $\text{ALG}(r_i, \ldots, r_j)$, where the prefix is understood implicitly.

For a minimization problem, we say that an algorithm is *c-competitive*, or has a *competitive ratio* of c, if, for every finite request sequence σ, $\text{ALG}(\sigma) \leq c \cdot \text{OPT}(\sigma) + \zeta$, where ζ is not dependent on the request sequence and $\text{OPT}(\sigma)$ is the optimum cost over σ. If $\zeta = 0$, we say that an online algorithm is *strictly c-competitive*.

The *k-server problem* consists of a metric space, $\mathcal{M}$, k mobile servers and a finite request sequence, σ. Let $\mathcal{M} = (M, d)$, where M is a set of nodes, $d : M \times M \rightarrow \mathbb{R}^+$ is a distance function on M and $|M| = N > k$. Each request of σ will be to a node of $\mathcal{M}$, and a server must be moved to the requested node before the algorithm will receive the subsequent request. The goal is to minimize the distance travelled by the k servers over σ. A *lazy* k-server algorithm is an algorithm that, upon each request, only moves a single server to the request if it is uncovered.

For a metric space which is a tree, we say that a server, s, is *adjacent* to a request, r_i, if, along the shortest path between the position of s and r_i, there are no other servers.

The *caterpillar dimension* of a rooted tree, T, with root r, denoted $\text{cdim}(T)$, is defined as in [8]. For a tree, T, composed of a single node, $\text{cdim}(T) = 0$. For a tree, T, with two or more nodes, $\text{cdim}(T) \leq k + 1$ if there exists edge disjoint paths, $P_1, \ldots, P_q$, beginning at the root r such that each component

T_j of $T - E(P_1) - \cdots - E(P_q)$ has $\mathrm{cdim}(T_j) \leq k$, where $E(P_i)$ are the edges of P_i. The components T_j are rooted at their unique vertex lying on some P_i. The decomposition of T into these edge disjoint paths is called the *caterpillar decomposition* of the tree. All the nodes of P_i, $1 \leq i \leq q$, except the root, are assigned path level $k+1$. The root is assigned path level $k+2$. Note that the root of the tree has a path level one more than the caterpillar dimension of the tree.

Given an unrooted tree, G, we define the caterpillar dimension of G as the minimum over all nodes, $v \in G$, of the caterpillar dimension of G when rooted at v. In what follows, we refer to the caterpillar dimension of unrooted trees as defined here.

3 An Upper Bound for General Metric Spaces

In this section, we present a $\left\lceil \frac{\lceil \log k \rceil}{b-2} \right\rceil$-competitive deterministic online algorithm with advice, called CHASE, for the k-server problem on general metric spaces, with b bits of advice per request, where $b \geq 3$. For convenience of notation, we use $\alpha = \left\lceil \frac{\lceil \log k \rceil}{b-2} \right\rceil$.

In order to clearly present the algorithm and proof, we will first design and analyze the algorithm such that it gets a variable number of bits of advice with each request. The algorithm will receive at least 2 bits of advice with each request, and the total number of advice bits will not exceed bn for any prefix of n requests. Afterwards, we will show how to adapt the algorithm so that it gets at most b bits of advice with each request using a transformation of [1].

Roughly speaking, our algorithm works as follows: given a request sequence, σ, we consider an optimal algorithm for this sequence. Based on this optimal algorithm, we partition σ into k subsequences, σ^s, such that all the requests of σ^s are served according to the optimal algorithm by server s. With $\log k$ bits of advice per request, we can indicate, with each request of σ^s, the identity of the server s, and, thus, our online algorithm with advice would precisely follow the optimum algorithm. If, however, we have only $b < \log k$ bits of advice per request, we do that only roughly every $\log k/b$ requests of σ^s. We call these requests "anchors". The rest of the requests of σ^s are served in a greedy manner, i.e., they are served by the closest server to the request which then returns to its previous position. By serving requests in this way, server s always stays at its last anchor. Thus, the cost of serving the $(\log k/b) - 1$ non-anchor requests between any two anchors is bounded from above by $2 \log k/b$ times the distance from the last anchor to the furthest non-anchor request. This gives us a competitive ratio of $O(\log k/b)$. Some fine tuning of the above ideas gives us our result. In what follows, we formally define the algorithm and prove its competitive ratio.

Algorithm CHASE: At the beginning, all servers are unmarked.
Given a request, r_j, and the advice, do:

- If the advice is 00, serve r_j with the closest server to r_j and return it to its previous position.
- If the advice is 10, serve r_j with the closest unmarked server and mark this server. Do not return the server to its previous position.
- If the advice is $11t$, where t is a server number encoded in $\lceil \log k \rceil$ bits, serve the request with server number t.

In order to define the advice, we will fix a optimum algorithm, OPT, that we assume to be a lazy algorithm. Henceforth, we refer to it as the lazy optimum. We will then partition the request sequence into k subsequences, $\sigma^1, \ldots, \sigma^k$, where σ^s is the trace of the server s in OPT. In other words, σ^s consists of the requests served by server s in the lazy optimum. It should be noted that the requests of σ^s are not necessarily consecutive requests in σ. Let r_j^s be the jth request served by server s over σ^s. Recall that $\alpha = \left\lceil \frac{\lceil \log k \rceil}{b-2} \right\rceil$. Independently, for each server, we choose an index $1 \leq q^s \leq \alpha$. The way to choose this index will be defined later. The request sequence σ^s is divided into α-length cycles starting at $r^s_{q^s+1}$. We will denote the ith cycle of σ^s by c_i^s. The first cycle, c_1^s, which starts at request r_1^s and ends at request $r^s_{q_s}$, may have a length less than α. Let C^s be the total number of cycles in σ^s.

The advice will be defined as follows for request r_j^s:

- 10, if $j = q^s$, i.e., the last request of the first cycle.
- $11t$, if $j = q^s + i\alpha$ for some $i \geq 1$, i.e., the last request of all cycles except the first one. Here, t is the server number that serves request $r^s_{q^s}$ in CHASE encoded in $\lceil \log k \rceil$ bits.
- 00, if $j \neq q^s + i\alpha$, i.e., everywhere else.

The first two bits of the advice per request will be referred to as the control bits.

First, we state a technical lemma that we will use in our proof.

Lemma 1. *Given a sequence of α non-negative values, $a_1, \ldots, a_\alpha$, there is an integral value, q, where $1 \leq q \leq \alpha$, such that*

$$\sum_{i=1}^{q} (2(q-i)+1)a_i + \sum_{i=q+1}^{\alpha} 2(\alpha+q-i)a_i \leq \alpha \sum_{i=1}^{\alpha} a_i \ .$$

Proof. Summing the expression over all possible values of q, we get

$$\begin{aligned}\sum_{q=1}^{\alpha} \left[\sum_{i=1}^{q} (2(q-i)+1)a_i + \sum_{i=q+1}^{\alpha} 2(\alpha+q-i)a_i \right] &= \left[\sum_{q=1}^{\alpha} \big(2(\alpha-q)+1\big) \right] \cdot \sum_{i=1}^{\alpha} a_i \\ &= \alpha^2 \sum_{i=1}^{\alpha} a_i \ .\end{aligned}$$

It follows that one of the α possible values of q gives at most the average value, i.e., $\alpha \sum_{i=1}^{\alpha} a_i$. The lemma follows.

Now, we prove the main theorem of this section.

Theorem 1. *For every $b \geq 3$, algorithm CHASE is an $\left\lceil \frac{\lceil \log k \rceil}{b-2} \right\rceil$-competitive k-server algorithm for general metric spaces with b bits of advice per request .*

Proof. For the proof, we will compare the cost of CHASE and OPT separately for every subsequence σ^s, and cycle by cycle within each σ^s. Recall that $\alpha = \left\lceil \frac{\lceil \log k \rceil}{b-2} \right\rceil$. Note that the first cycle and the last cycle may be of length less than α.

Consider the ith cycle of server s in OPT for $i > 1$ (we will deal with the first cycle later). Let t be the server in CHASE that serves request $r^s_{q^s}$. We will denote $r^s_{(i-2)\alpha+q^s}$, the last request of the previous cycle, by INIT^s_i. We claim that, just before the cycle starts, both OPT and CHASE will have a server at INIT^s_i. This is true because the advice for request $r^s_{(i-2)\alpha+q^s}$ indicated to CHASE to bring server t to INIT^s_i and, by the definition of the algorithm, t will always return to INIT^s_i between $r^s_{(i-2)\alpha+q^s}$ and $r^s_{(i-2)\alpha+q^s+1}$. For OPT, by definition of the subsequence σ^s, OPT serves $r^s_{(i-2)\alpha+q^s}$ with s and does not move s between request $r^s_{(i-2)\alpha+q^s}$ and request $r^s_{(i-2)\alpha+q^s+1}$.

Also, observe that just before each of the requests between $r^s_{(i-2)\alpha+q^s+1}$ and $r^s_{(i-1)\alpha+q^s}$ inclusive, i.e., the requests of the ith cycle, server t of CHASE is at INIT^s_i. Recall that CHASE serves these requests except the last one by using the closest server and, then, returns it to its prior position. Therefore, the cost to CHASE for any request $r^s_{(i-2)\alpha+q^s+j}$, where $1 \leq j \leq \alpha - 1$, i.e., the requests of cycle i except the last one, is

$$\mathrm{CHASE}(r^s_{(i-2)\alpha+q^s+j}) \leq 2d(\mathrm{INIT}^s_i, r^s_{(i-2)\alpha+q^s+j}) \ . \tag{1}$$

By the triangle inequality and Equation (1),

$$\mathrm{CHASE}(r^s_{(i-2)\alpha+q^s+j}) \leq 2\sum_{l=1}^{j} d(r^s_{(i-2)\alpha+q^s+l-1}, r^s_{(i-2)\alpha+q^s+l}) \ . \tag{2}$$

For request $r^s_{(i-1)\alpha+q^s}$, i.e., the last request of cycle i, CHASE serves the request using server t that is at $r^s_{(i-2)\alpha+q^s}$. We have, by the triangle inequality,

$$\begin{aligned} \mathrm{CHASE}(r^s_{(i-1)\alpha+q^s}) &= d(\mathrm{INIT}^s_i, r^s_{(i-1)\alpha+q^s}) \\ &\leq \sum_{l=1}^{\alpha} d(r^s_{(i-2)\alpha+q^s+l-1}, r^s_{(i-2)\alpha+q^s+l}) \ . \end{aligned} \tag{3}$$

Observe that the cost of OPT to serve $r^s_{(i-2)\alpha+q^s+j}$ for $1 \leq j \leq \alpha$, i.e., the requests of cycle i, is $d(r^s_{(i-2)\alpha+q^s+j-1}, r^s_{(i-2)\alpha+q^s+j})$. Using this fact and Equations (2) and (3), we can bound the cost of CHASE over a cycle by the cost

of OPT as follows:

$$\sum_{j=1}^{\alpha} \mathrm{CHASE}(r^s_{(i-2)\alpha+q^s+j}) \leq \sum_{j=1}^{\alpha-1} \left(2 \sum_{l=1}^{j} \mathrm{OPT}(r^s_{(i-2)\alpha+q^s+l}) \right) + \sum_{l=1}^{\alpha} \mathrm{OPT}(r^s_{(i-2)\alpha+q^s+l}) = \sum_{j=1}^{\alpha} [2(\alpha-j)+1]\mathrm{OPT}(r^s_{(i-2)\alpha+q^s+j}) \ . \quad (4)$$

The analysis of the first cycle is, essentially, the same as the analysis of the ith cycle, $i > 1$, with the exception that an additive constant is introduced per request of the first cycle. The additive constant results from the fact that, during the first cycle of σ^s, CHASE does not necessarily maintain a server at the initial position of s. Nevertheless, by the definition of CHASE, there will always be an unmarked server in one of the locations of the initial configuration. Let Δ be the diameter of the initial configuration. Therefore, for any request of the first cycle, r^s_l, of σ^s, analogously to Equation (2), we have

$$\mathrm{CHASE}(r^s_l) \leq 2\Big(\Delta + \sum_{m=1}^{l} d(r^s_{m-1}, r^s_m)\Big) \ , \quad (5)$$

where r^s_0 is the initial position of s. Analogous to Equation (4), summing Equation (5) over all requests of the first cycle of s, gives

$$\sum_{l=1}^{q^s} \mathrm{CHASE}(r^s_l) \leq \sum_{l=1}^{q^s} [2(q^s-l)+1]\mathrm{OPT}(r^s_l) + 2\alpha\Delta \ . \quad (6)$$

If we assume the cost for requests with indexes less than 0 to be 0 for both OPT and CHASE, we can rewrite Equation (6) to be more congruent with Equation (4) as follows:

$$\sum_{j=1}^{\alpha} \mathrm{CHASE}(r^s_{-\alpha+q^s+j}) \leq \sum_{j=1}^{\alpha} [2(\alpha-j)+1]\mathrm{OPT}(r^s_{-\alpha+q^s+j}) + 2\alpha\Delta \ . \quad (7)$$

Using Equations (4) and (7), and summing over all cycles, gives

$$\mathrm{CHASE}(\sigma^s) \leq \sum_{i=1}^{C^s} \sum_{j=1}^{\alpha} [2(\alpha-j)+1]\mathrm{OPT}(r^s_{(i-2)\alpha+q^s+j}) + 2\alpha\Delta \ . \quad (8)$$

Define $a_1, \ldots, a_\alpha$ such that $a_j = \sum_{i=1}^{C^s} \mathrm{OPT}(r^s_{(i-1)\alpha+j})$, i.e., the cost of OPT for the requests in σ^s in jumps of α requests. We can rewrite Equation (8) as

$$\mathrm{CHASE}(\sigma^s) \leq \sum_{i=1}^{q} (2(q-i)+1)a_i + \sum_{i=q+1}^{\alpha} 2(\alpha+q-i)a_i + 2\alpha\Delta \ . \quad (9)$$

By Lemma 1, there is a value $1 \leq q^s \leq \alpha$ such that

$$\text{CHASE}(\sigma^s) \leq \alpha \sum_{i=1}^{\alpha} a_i + 2\alpha\Delta = \alpha\text{OPT}(\sigma^s) + 2\alpha\Delta \ .$$

We chose this q^s separately for each server s in order to define the cycles. Summing over all k subsequences σ^s concludes the proof of the competitive ratio.

Finally, we show that the algorithm uses at most bn bits over any prefix of n requests. There are 2 control bits with each request. Let t be the server in CHASE that serves $r^s_{q_s}$, i.e., the last request of the first cycle of σ^s. There are at least α requests of σ^s between any two requests, where the id of t is given in the advice. Since $\alpha = \left\lceil \frac{\lceil \log k \rceil}{b-2} \right\rceil$, the claim follows.

In order to adapt the algorithm so that it receives b bits of advice per request, we use a transformation of [1]. Two control bits will be given with each request, and the remaining $b-2$ bits will contain portions of server ids. The control bits will be as defined previously. We then define a string as the concatenation of all server ids given for the whole sequence. This string will be broken into $(b-2)$-bit chunks and a single chunk will be given with each request. The algorithm can store these $(b-2)$-bit chunks in a FIFO queue and will have $\lceil \log k \rceil$ bits available to be read from the queue when dictated by the control bits.

4 *k*-Server with Advice on Trees

In this section, we describe a deterministic online algorithm with advice for the k-server problem on finite trees, called PATH-COVER, that is 1-competitive and uses $2 + 2\lceil \log(p+1) \rceil$ bits of advice per request, where p denotes the minimal caterpillar dimension of the tree. Similar results can be obtained if other measures of the tree, such as its height, are used instead of the caterpillar dimension. We chose this measure since it gives a 1-competitive algorithm with a constant number of bits of advice per request for degenerate trees such as the line or a caterpillar. Furthermore, the caterpillar dimension is at most the height of the tree, and is at most $\log N$, where N is the number of nodes in the tree [8].

The algorithm and advice are based on the actions of a non-lazy optimum algorithm with certain properties for the given sequence. First, we describe this non-lazy algorithm and show that it has optimum cost.

4.1 Non-Lazy Optimum

We show that, for every sequence of requests, there is an algorithm, OPT_{nl}, with optimal cost that, also, has the following three properties given that the algorithm can chose its initial configuration.

1. Between r_i and just before r_{i+1}, OPT_{nl} moves at most a single server, s. Note that s may make multiple moves.

2. Just before r_i, s is at the same path level or higher than r_i. The path level is according to the caterpillar decomposition of the tree.
3. s is adjacent to r_i just before r_i.

Given the caterpillar decomposition of T that minimizes $\mathrm{cdim}(T)$ and a lazy optimum, OPT_l, we first build OPT'_{nl} which has the first two properties above. For $u, v \in T$, let $\mathrm{maxPath}(u, v)$ be the node nearest u on the highest path level on the path between u and v. We choose the initial configuration of OPT'_{nl} as follows: for each of the k servers s_i, place s_i at $\mathrm{maxPath}(u_i, v_i)$, where u_i is the initial position of s_i, and v_i is the position of the first request served by s_i in OPT_l. Then, each request, r_j, is served in OPT'_{nl} with the same server as OPT_l. After serving r_j, place the server, t, used for r_j at $\mathrm{maxPath}(r_j, r_q)$, where r_q is the next request served by t in OPT_l. Observe that the first two properties above hold for OPT'_{nl}.

Claim. For any σ, $\mathrm{OPT}'_{nl}(\sigma) = \mathrm{OPT}_l(\sigma)$.

Proof. The claim follows from the fact that the trajectories followed by each of the servers according to OPT'_{nl} and OPT_l are the same. The only difference being that some of the moves are done earlier in OPT'_{nl} than in OPT_l.

Now, we construct OPT_{nl} based on OPT'_{nl} to satisfy property 3 along with the first two properties without increasing the cost.

OPT_{nl} will be defined by induction on the request sequence. Let $T^* = (t^*_1, q_1), \ldots, (t^*_m, q_m)$ be all the server moves, in order, performed by OPT'_{nl} such that the ordered pair (x, y) defines a move of a server from position x to position y, and $m \geq n$ is the total number of server moves performed by OPT_{nl}. Let $S^* = (s^*_1, r_1), \ldots, (s^*_n, r_n)$ be the subsequence of T^*, where the ith ordered pair indicates that the server at position s^*_i serves r_i in OPT'_{nl}. We build T^j and S^j, $j \geq 0$, inductively, where $T^0 = T^*$ and $S^0 = S^*$. $T^j = (t^j_1, q_1), \ldots, (t^j_m, q_m)$ will have all three properties above until after serving request r_j (in fact Property 1 will hold for the entire sequence T^j), and $S^j = (s^j_1, r_1), \ldots, (s^j_n, r_n)$ will have the property that $s^j_1, \ldots, s^j_j$ are adjacent to their respective requests.

Assume that S^{i-1} and T^{i-1} are well defined. This is trivially true for T^0 and S^0. In order to construct S^i and T^i, we need to consider s^{i-1}_i which is either adjacent or not to r_i. If s^{i-1}_i is adjacent to r_i, then $S^i := S^{i-1}$ and $T^i := T^{i-1}$. Otherwise, if s^{i-1}_i is not adjacent to r_i, there exists a server at node x on the path between s^{i-1}_i and r_i which is adjacent to r_i. In this case, the relevant moves in both T and S will be modified such that the server at x serves request r_i, and the server at s^{i-1}_i will be used the very next time that the server at x would have been used. To formally define T^i and S^i we proceed as follows: S^i is defined as S^{i-1} from (s^i_1, r_1) to (s^i_{i-1}, r_{i-1}), and T^i is defined as T^{i-1} from (t^{i-1}_1, q_1) to (t^{i-1}_j, q_{j-1}), where q_j is r_i. The next moves in S^i and T^i are defined as $(s^i_i, r_i) := (t^i_j, q_j) := (x, r_i)$. From (t^i_{j+1}, q_{j+1}) to (t^i_m, q_m), T^i is defined as T^{i-1} except for the next occurrence of (x, q_l) in T^{i-1}, $l < m$. Set $(t^i_l, q_l) := (s^{i-1}_i, q_l)$ in T^i. If q_l is a request, say r_p, then we, also, set $(s^i_p, r_p) := (s^{i-1}_i, r_p)$ for S^i. The rest of S^i is defined as S^{i-1}.

Lemma 2. *On the tree, the cost of OPT_{nl} is no more than the cost of OPT'_{nl}, and OPT_{nl} has the following three properties:*

1. *Between r_i and just before r_{i+1}, OPT_{nl} moves at most a single server, s.*
2. *Just before r_i, s is at the same path level or higher than r_i.*
3. *s is adjacent to r_i just before r_i.*

Proof. First, we show that the cost of OPT_{nl} is no more than the cost of OPT'_{nl}. This is done by induction on the construction steps of T^i (and S^i). For $i = 0$ the claim is trivial. For the inductive step, we note that we change at most two moves between T^{i-1} and T^i as in the construction above. With the notations of the construction above, we note that the cost of T^i for q_i and q_l is at most $d(s_i^i, q_j) + d(s_i^{i-1}, s_i^i) + d(s_i^{i-1}, q_l) = d(s_i^{i-1}, q_j) + d(s_i^i, q_l)$ which is the cost paid by T^{i-1} for q_i and q_l.

Property 1 holds for OPT_{nl} because it holds for OPT'_{nl}, no moves are added in the construction, and the only changes are to the first move after a request is issued.

We now show by induction on i that property 2 holds for S^i. This is true for S^0 since property 2 holds for OPT'_{nl}. For the induction step, let s' be the server used by S^{i-1} to serve r_i, and let s be the server used by S^i to serve r_i . Let $\ell(v)$ denote the path level of a node v. We know by the induction hypothesis that $\ell(s') \geq \ell(r_i)$. Also, s lies on the path between s' and r_i, and, by the recursive nature of the caterpillar decomposition, it follows that $\ell(s) \geq \ell(r_i)$. If q_l is not a request, there is nothing else to prove. If q_l is a request, by the induction hypothesis, we know that $\ell(s) \geq \ell(q_l)$, and we have that $\ell(s') \geq \ell(s)$. Therefore, $\ell(s') \geq \ell(q_l)$. Thus, property 2 holds for S^i.

Property 3 is immediate from the inductive construction.

4.2 The Algorithm

There will be two stages to the algorithm. The initial stage will be for the first k requests and will be used to match the configuration of PATH-COVER to that of OPT_{nl} as defined in the previous section. Over the remaining requests, PATH-COVER will be designed to act exactly as OPT_{nl}. PATH-COVER will receive $2(l + 1)$ bits of advice per request, where $l = \lceil \log(p + 1) \rceil$ and p is the minimal caterpillar dimension of the tree. The advice will be of the form $wxyz$, where w and x will be 1 bit in length, and y and z will be l bits in length.

Algorithm and Advice for $r_1, \ldots, r_k$ From r_1 to r_k, PATH-COVER will serve each request with the nearest server regardless of the advice. As for the advice, for request r_i, where $1 \leq i \leq k$,

- if $w = 1$, the algorithm stores the node nearest r_i which has the path of level y.
- if $x = 1$, the algorithm stores the node nearest the initial position of the ith server which has the path of level z.

Note that both w and x can be 1 for request r_i. Immediately after serving r_k, PATH-COVER will use the first k stored nodes as a server configuration and will move to this configuration at minimal cost (minimum matching).

For $1 \leq i \leq k$, the advice for r_i will be defined as follows:

- w: 1, if the server used for r_i in OPT_{nl} does not serve another request up to r_k.
  0, otherwise
- x: 1, if the ith server does not serve any of the first k requests in OPT_{nl}.
  0, otherwise
- y: A number in binary indicating the path level to which the server used for r_i is moved to in OPT_{nl} after serving r_i.
- z: A number in binary indicating the path level to which the ith server is moved to before r_1 in OPT_{nl}.

Over the first k requests, w and x will be 1 a total of k times (once for each of the k servers of OPT_{nl}). For r_i, when w is 1, this means that, immediately after r_k, OPT_{nl} will have a server at the path of level y between r_i and the root. When x is 1, this means that, immediately after r_k, the ith server of OPT_{nl} will be at the path of level z between the initial position of the ith server and the root. This is the server configuration of OPT_{nl} immediately after r_k encoded in the bits of advice over the first k requests.

Algorithm and Advice for $r_{k+1}, \ldots, r_n$ From r_{k+1} to r_n, given a request, r_i, where $k+1 \leq i \leq n$, and the advice, let P be the unique path of level y between r_i and the root. Now, define a path, Q, on the tree as follows:

- if $x = 1$, Q runs from r_i to the end of P nearest the root.
- if $x = 0$, Q runs from r_i to the end of P furthest from the root.

PATH-COVER will serve r_i with the closest server along Q. After serving r_i, PATH-COVER will move this server to the node of the path of level z nearest to r_i.

For $k+1 \leq i \leq n$, the advice for r_i will be defined as follows:

- w: 0 (not used)
- x: Let s be the server used by OPT_{nl} to serve r_i, and let c be the node of the same path level as the location of s closest to r_i.
  1, if s is between c and the root of the tree.
  0, otherwise.
- y: A number in binary indicating the path level to which of the server that OPT_{nl} uses for r_i.
- z: A number in binary indicating the path level to which the server used for r_i is moved to in OPT_{nl} immediately after serving r_i.

Analysis

Theorem 2. *PATH-COVER is* 1*-competitive on finite trees.*

Proof. From r_1 to r_k, all the requests are served by the closest server. This cost can be bounded by $k\Delta$, where Δ is the diameter of the tree. Immediately after r_k, PATH-COVER matches the configuration of OPT_{nl}. The cost to match a configuration can be bounded by $k\Delta$.

According to the definition of the advice and the algorithm, the configuration of PATH-COVER matches the configuration of OPT_{nl} after serving r_k. We show that starting at request r_{k+1}, the configurations of PATH-COVER and OPT_{nl} match just before serving each request, and that PATH-COVER serves the request with a server from the same position as does OPT_{nl}. Assume that the configurations of PATH-COVER and OPT_{nl} match until immediately before serving r_i for some $k+1 \leq i \leq n$. Let s be the server used by OPT_{nl} for r_i. We claim that PATH-COVER and OPT_{nl} will use a server from the same position for r_i. The induction assumption that the configurations of PATH-COVER and OPT_{nl} match immediately before serving r_i and the third property of OPT_{nl} guarantee that there is no server between the position of s and r_i in PATH-COVER. The second property of OPT_{nl} guarantees that s is at the same path level or higher than r_i. This implies that the advice provided to PATH-COVER specifies a unique server that must be at the same position in PATH-COVER as s in OPT_{nl}. Immediately after serving r_i, PATH-COVER and OPT_{nl} move the server used for r_i to the same node in the tree as per their definitions. So, immediately before serving r_{i+1}, the configurations of OPT_{nl} and PATH-COVER are the same.

Therefore,

$$\text{PATH-COVER}(\sigma) \leq \mathrm{OPT}(\sigma) + 2k\Delta$$

4.3 Special Metric Spaces and PATH-COVER

This section presents some variations and implication of the previously described algorithm, PATH-COVER, for special metric spaces. More detailed proofs of each case can be found in [9]

The Line The caterpillar dimension for the line is 1 which implies that our algorithm PATH-COVER requires 4 bits of advice. However, as the servers essentially do not change path levels on the line, a single bit of advice indicating the direction of the server to be used is all that is needed for a strictly 1-competitive algorithm.

The Circle Applying the algorithm for the line to the circle provides a strictly 1-competitive algorithm with 1 bit of advice. The key is to define a clockwise orientation on the cycle. The 1 bit of advice will indicate whether to use the adjacent server in the clockwise direction or the counter-clockwise direction.

The Spider A spider graph consists of a single fork, the centre, and 0 or more branches without forks connected to the centre. The caterpillar dimension is 1 implying 4 bits of advice for PATH-COVER. We can define a variant of PATH-COVER for the spider that uses 2 bits of advice. One bit indicates the direction of the adjacent server, s, for the request while the second bit indicates if s should be moved to the centre after serving the request. This algorithm is 1-competitive.

5 Conclusions

We give an improved upper bound for the k-server problem with advice on general metric spaces. Moreover, we believe that our algorithm and our analysis are more intuitive and simpler than previous ones, and may, thus, lead to further improvements in the upper bound. We, also, give a 1-competitive k-server algorithm with advice for finite trees, using a number of bits of advice which is a function of the caterpillar dimension of the tree. The obvious open problem that remains is to give tight bounds for the k-server problem with advice on general metric spaces or for specific metric spaces.

Acknowledgements We thank Yuval Emek, Pierre Fraigniaud, Amos Korman, and Manor Mendel for useful discussions.

References

1. Böckenhauer, H.J., Komm, D., Královic, R., Královic, R.: On the advice complexity of the k-server problem. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP (1). Lecture Notes in Computer Science, vol. 6755, pp. 207–218. Springer (2011), see also technical report at ftp://ftp.inf.ethz.ch/pub/publications/tech-reports/7xx/703.pdf
2. Böckenhauer, H.J., Komm, D., Královic, R., Královic, R., Mömke, T.: On the advice complexity of online problems. In: Dong, Y., Du, D.Z., Ibarra, O.H. (eds.) ISAAC. Lecture Notes in Computer Science, vol. 5878, pp. 331–340. Springer (2009)
3. Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press, New York, NY, USA (1998)
4. Dobrev, S., Královič, R., Pardubská, D.: How much information about the future is needed? In: SOFSEM'08: Proceedings of the 34th conference on Current trends in theory and practice of computer science. pp. 247–258. Springer-Verlag, Berlin, Heidelberg (2008)
5. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. Theor. Comput. Sci. 412(24), 2642–2656 (2011)
6. Hromkovic, J., Královic, R., Královic, R.: Information complexity of online problems. In: Hlinený, P., Kucera, A. (eds.) MFCS. Lecture Notes in Computer Science, vol. 6281, pp. 24–36. Springer (2010)
7. Komm, D., Královic, R.: Advice complexity and barely random algorithms. In: Cerná, I., Gyimóthy, T., Hromkovic, J., Jeffery, K.G., Královic, R., Vukolic, M., Wolf, S. (eds.) SOFSEM. Lecture Notes in Computer Science, vol. 6543, pp. 332–343. Springer (2011)

8. Matoušek, J.: On embedding trees into uniformly convex banach spaces. Israel Journal of Mathematics 114, 221–237 (1999), http://dx.doi.org/10.1007/BF02785579, 10.1007/BF02785579
9. Renault, M.: Online Algorithms with Advice. Master's thesis, MPRI – Université Paris Diderot - Paris 7 (2010)