On the Advice Complexity of the $k$-server Problem Under Sparse Metrics

Sushmita Gupta$^1$ · Shahin Kamali$^2$ · Alejandro López-Ortiz$^2$

Published online: 20 August 2015
© Springer Science+Business Media New York 2015

Abstract We consider the $k$-SERVER problem under the advice model of computation when the underlying metric space is sparse. On one side, we introduce $\Theta(1)$-competitive algorithms for a wide range of sparse graphs. These algorithms require advice of (almost) linear size. We show that for graphs of size $N$ and treewidth $\alpha$, there is an online algorithm that receives $O(n(\log \alpha + \log \log N))$ bits of advice and optimally serves any sequence of length $n$. We also prove that if a graph admits a system of $\mu$ collective tree $(q, r)$-spanners, then there is a $(q + r)$-competitive algorithm which requires $O(n(\log \mu + \log \log N))$ bits of advice. Among other results, this gives a 3-competitive algorithm for planar graphs, when provided with $O(n \log \log N)$ bits of advice. On the other side, we prove that advice of size $\Omega(n)$ is required to obtain a 1-competitive algorithm for sequences of length $n$ even for the 2-server problem on a path metric of size $N \geq 3$. Through another lower bound

Preliminary version of this paper appeared in the Proceedings of the International Colloquium on Structural Information and Communication Complexity, SIROCCO’13 [24].

*We use $\log x$ to denote $\log_2(x)$.

Shahin Kamali
shahin.kamali@gmail.com; s3kamali@uwaterloo.ca

Sushmita Gupta
sushmita.gupta@gmail.com; gupta.sushmita.2c@kyoto-u.ac.jp

Alejandro López-Ortiz
alopez-o@uwaterloo.ca

1 Graduate School of Informatics, Kyoto University, Research Bldg. No. 7, Yoshida Honmachi, Sakyo-ku, Kyoto, Japan

2 School of Computer Science, University of Waterloo, 200 University Avenue West, Waterloo, Ontario, Canada
argument, we show that at least $\frac{\alpha}{4} \log \alpha - 1.22$ bits of advice is required to obtain an optimal solution for metric spaces of treewidth $\alpha$, where $4 \leq \alpha < 2k$.

**Keywords** $k$-Server problem · Advice complexity · Competitive analysis

### 1 Introduction

Online algorithms have been extensively studied in the last few decades. In the standard setting, the input to an online algorithm is a sequence of requests which need to be answered sequentially. To answer each request, the algorithm has to take an irrevocable decision without looking at the incoming requests. For minimization problems, such a decision involves a cost and the goal is to minimize the total cost.

The standard method for analysis of online algorithms is the competitive analysis which compares an online algorithm with an optimal offline algorithm, OPT. The asymptotic competitive ratio of an online algorithm $A$ is the maximum ratio between the cost of $A$ and that of OPT for serving any sufficiently long sequence. Although competitive analysis is accepted as the main tool for analysis of online algorithms, its limitations have been known since its introduction. Inputs adversarially produced to draw out the worst performance of algorithms are not commonplace in real life applications. Therefore, in essence, competitive analysis mostly measures the benefit of knowing the future and not the true difficulty of instances. From the perspective of an online algorithm, the algorithm is overcharged for its complete lack of knowledge about the future. Advice model is an alternative framework for analysis of online algorithms in which an algorithm receives a number of bits of advice about the unrevealed parts of the sequence. Clearly, algorithms perform better when a larger number of bits of advice is available. Yet, there are problems for which only a small number of bits is sufficient to achieve optimal or near-optimal solutions. For example, for the well-known ski-rental problem [27, 28], a single bit of advice is sufficient to achieve an optimal solution.

We are interested in the advice complexity of the $k$-SERVER problem, as well as the relationship between the size of advice and the competitive ratio of online algorithms. To this end, we study the problem for a wide variety of sparse graphs.

#### 1.1 Advice Model of Computation

Under the advice model, we assume there is an offline oracle with infinite computational power which generates bits of advice for an online algorithm. Depending on how these bits are given to the algorithm, slightly different models have been proposed. Dobrev et al. [17] proposed a model in which the offline oracle answers to a series of questions posed by the online algorithm. These answers are sent to the algorithm in the form of blocks of answers. The problem with this model is that, besides the actual data, a lot of information can be encoded in the individual length of the blocks. Emek et al. [20] proposed another model in which the algorithm receives a fixed number of bits of advice per request. We refer to this model as Advice-Per-Request model. This model captures the scenarios in which the whole advice is not
available from the start and the advice needs to be calculated during the course of the algorithm in response to the decisions made by the online algorithm. Unfortunately this model does not allow sending a sublinear amount of information in the form of advice. To address this issue, another model is introduced by Böckenhauer et al. [9] where the online algorithm has access to an advice tape generated by the offline oracle. At each time, the algorithm can access any number of advice bits from the tape. The advice complexity of the algorithm is the total number of bits accessed on the advice tape. We refer to this model as Advice-On-Tape model.

In this paper, we adopt the Advice-On-Tape model. This model has been used to analyse the advice complexity of many online problems, which includes paging [9, 26, 29], disjoint path allocation [9], job shop scheduling [9, 29], knapsack [6], bipartite graph coloring [4], online coloring of paths [22], other coloring problems [4, 5, 38], set cover [7, 30], maximum clique [7], graph exploration [16], Bin Packing [12], and List Update problem [11], in addition to \(k\)-SERVER which we discuss in some details.

1.1.1 \(k\)-Server Problem under Advice Model

The \(k\)-SERVER problem was first introduced by Manasse et al. [34]. An instance of the problem contains a metric space \(M\), \(k\) mobile servers, and a request sequence \(I\). The metric space can be modelled as an undirected, weighted graph of size \(N > k\). We interchangeably use the terms metric space and graph. Each request in the input sequence \(I\) denotes a vertex of \(M\), and an online algorithm should move one of the servers to the requested vertex to serve the request. The cost of the algorithm is defined as the total distance moved by all \(k\) servers over \(M\) to serve \(I\). In this paper we only consider deterministic algorithms. For these algorithms, it is known that no online algorithm can be better than \(k\)-competitive [34]. The \(k\)-SERVER conjecture states that for any metric space there is a \(k\)-competitive algorithm. While the conjecture is still open after more than two decades, its correctness is verified for \(k = 2\) [34] and some special metric spaces [13–15, 31, 32]. For general metric spaces, the best existing online algorithm is the Work Function Algorithm which has a competitive ratio of at most \(2k - 1\) [31].

We formally define the \(k\)-SERVER problem under the advice model as follows, based on the definition of advice by Böckenhauer et al. [9]:

**Definition 1** In the \(k\)-SERVER problem with advice, the input is a sequence \(\sigma = \langle x_1, x_2, \ldots, x_n \rangle\) of requests to vertices of an undirected graph \(G\) with non-negative weights on edges. These requests are revealed to the algorithm in an online manner. To answer each request, the algorithm should move one of the \(k\) servers to the requested vertex. The servers are originally located in an initial configuration which is known to the algorithm. The goal is to minimize the total distance travelled by all servers. At time step \(t\), the algorithm should move a server to vertex \(x_t\). The decision of the algorithm to select such a server is a function of \(\Phi\) and \(x_1, \ldots, x_t\), where \(\Phi\) is the content of the advice tape. An algorithm \(A\) is \(c\)-competitive with advice complexity \(s(n)\) if there is an advice tape with content \(\Phi\) so that the algorithm accesses at most \(s(n)\) bits of the tape, and there exists a constant \(c_0\) such that, for every \(n\) and
for each input sequence \( \sigma \) of length at most \( n \), we have \( A(\sigma) \leq c \text{OPT}(\sigma) + c_0 \). An algorithm \( A \) is optimal if \( A(\sigma) = \text{OPT}(\sigma) \).

The \( k \)-SERVER problem has been among the first studied problems under the advice model. For the general metrics, Emek et al. [20] proved that there is an algorithm which achieves a competitive ratio of \( k^{O(1/b)} \) for \( b \leq k \), when provided with \( s(n) = bn \) bits of advice. This ratio was later improved to \( 2 \lceil \lceil \log k / (b - 1) \rceil \rceil \) by Böckenhauer et al. [8] and then to \( \lceil \lceil \log k / (b - 2) \rceil \rceil \) by Renault and Rosén [36]. Comparing these results with the lower bound \( k \) for the competitive ratio of any online algorithm [33, 34], one can see how advice of linear size can dramatically improve the competitive ratio. Regarding lower bounds, it is known that any online algorithm requires advice of size \( s(n) \geq n^2 (\log k - c) \) for some \( c < 1.443 \) to optimally service a sequence [8].

Since the introduction of the \( k \)-SERVER problem, there has been an interest in studying the problem under specific metric spaces. These include, trees [14], metric spaces with \( k + 2 \) points [1], Manhattan and Euclidean spaces [2], and the cross polytope space, which is a generalization of the uniform metric space [3]. For trees, it is known that the competitive ratio of any online algorithm is at least \( k \), and there exist algorithms which achieve this ratio [14].

Under the advice model, Böckenhauer et al. [8] studied the \( k \)-SERVER problem when the metric space is the Euclidean plane and proved that for a fixed \( b \geq 3 \), there exists an algorithm \( A \) with advice complexity \( s(n) = bn \) that achieves a competitive ratio \( r \leq \frac{1}{1-2 \sin \left( \frac{\pi}{2b} \right)} \). Renault and Rosén [36] considered tree metric spaces and introduced a 1-competitive algorithm which receives \( 2n + 2 \lceil \log (p + 2) \rceil n \) bits of advice, where \( p \) is the caterpillar dimension of the tree (it will be defined in Section 1.2). There are trees with size \( N \) for which \( p \) is as large as \( \lceil \log N \rceil \). Thus, this algorithm needs \( \Theta(n \log \log N) \) bits of advice.

1.2 Preliminaries

In this section, we formally define a few concepts which are used throughout the paper. As per convention, we use \( k \) to denote the number of servers, \( N \), the size of metric space (graph), \( n \), the length of input sequence, and \( \alpha \), the treewidth of the metric space.

1.2.1 Tree Decomposition and Treewidth

The concept of tree decomposition [25, 37] was introduced as a way of mapping a graph into a tree while preserving locality properties of the graph.

**Definition 2** (Tree Decomposition of a Graph) For any undirected graph \( G = (V, E) \), a tree decomposition of \( G \) with width \( \alpha \) is a pair \( (\{X_i \mid i \in B\}, T) \) where \( \{X_i \mid i \in B\} \) is a family of subsets of \( V \) (bags), and \( T \) is a tree whose vertices/nodes are the subsets \( X_i \) such that
\[ \bigcup_{i \in B} X_i = V \text{ and } \max_{i \in B} |X_i| = \alpha + 1. \]

- for each \((v, w) \in E\): there exists an \(i \in B\) such that \(v \in X_i\) and \(w \in X_i\).
- for each \(\{i, j, k\} \subseteq B\): if \(X_j\) is on the path from \(X_i\) to \(X_k\) in \(T\), then \(X_i \cap X_k \subseteq X_j\).

The treewith of a graph \(G\) is the minimum width among all tree decompositions of \(G\).

Informally speaking, the tree decomposition of a graph is a mapping of the graph into a tree so that the vertices associated to each node(bag) of the tree are close to each other; the treewidth measures how close the graph is to such a tree. Figure 1 shows an undirected graph and a tree decomposition of width 2 associated with it.

### 1.2.2 Caterpillar Decomposition

The caterpillar decomposition is a way of mapping a tree into several paths. This measure was first introduced by Matousek [35] and then applied in the context of \(k\)-SERVER problem by Renault and Rosén [36]. We use the definition given in [36]:

**Definition 3** (Caterpillar Decomposition and Dimension) A caterpillar decomposition of a rooted tree \(T\) is a hierarchy of disjoint paths defined as follows. The first level of the decomposition is formed by edge-disjoint paths \(P_1, \ldots, P_q\) each beginning at the root and ending at a leaf. This decomposition is recursively applied on any any component \(T_j\) in the forest \(T - P_1 - P_2 - \ldots - P_q\), i.e., the forest formed by removing vertices of \(P_i\) (1 \(\leq i \leq q\)). At the lowest level of the decomposition, each component is formed by a terminal-path in which no vertex has a child outside of the path. Given a caterpillar decomposition, each node \(x\) has a caterpillar dimension, denoted by \(\text{cdim}_x\), which is defined via the path that it belongs to. Vertices on the terminal paths have dimension 0. Other vertices have dimension \(k\), where \(k\) is

![Fig. 1](image-url)  
(a) An undirected graph \(G\).  
(b) A tree decomposition of \(G\) with width 2.
the smallest number such that all children of nodes in $P_i$ (except those in $P_j$) have
dimension at most $k - 1$. The caterpillar dimension of a tree is the smallest dimension
of the root of $T$ in any caterpillar decomposition.

For our purposes, the path-level of each of vertex in $T$ can be used ana-
logously as the height label in an arbitrarily rooted tree. It is known that $\text{cdim}(T) \leq \min\{\text{height}(T), \log N\}$ for a tree of size $N$ [35]. Hence, the caterpillar dimension is
preferable over other measures such as height as it remains a constant for degener-
ate trees, such as the line, the spider and the caterpillar trees. Figure 2 illustrates the
caterpillar decomposition of a tree with dimension 3.

1.2.3 Collective Tree Spanners

The spanner of a weighted graph is a spanning tree of the graph in which the dis-
tance between any pair of vertices in the tree is a good approximation of the (shortest)
distance between the pair in the original graph. The concept of collective tree span-
ners was introduced in [19] as an extension of tree spanners from a single tree to a
collection of spanning trees.

**Definition 4** (Collective Tree Spanners) We say that a graph $G = (V, E)$ admits
a system of $\mu$ collective tree $(q, r)$-spanners if there is a set $\mathcal{T}(G)$ of at most $\mu$
spanning trees of $G$ such that for any two vertices $x, y \in V$, there exists a spanning
tree $T \in \mathcal{T}(G)$ such that $d_T(x, y) \leq q \cdot d_G(x, y) + r$.

![Fig. 2](image) A caterpillar decomposition of a rooted tree with dimension 3. The numbers indicate the caterpillar dimensions of vertices
Figure 3 shows two spanning trees of an example graph so that for any pair of vertices, the distance between the pair in at least one of the trees is at most 1.5 times their actual distance in the graph. Hence, they form a system of 2 collective (1.5,0)-spanners.

1.3 Contribution

We study the $k$-server problem for a large family of sparse graphs. Similar to most related works (e.g., [8, 36]), we do not make any assumption on servers, and in particular, our algorithms do not need servers to be labelled. If servers are labelled, an online algorithm can serve any input, under any metric, with $O(n \log k)$ bits of advice, where the advice bits explicitly encodes which server is moved by an optimal algorithm for each request. In this case, our results are useful when the number of servers is $\Omega(\log \log n)$. We show the following results for the $k$-server problem:

– Algorithm 1: For a metric space with treewidth $\alpha$, we introduce an optimal algorithm with advice complexity $s(n) = O(n \log \alpha + \log \log N)$. In particular,
for graphs of constant treewidth, $O(n \log \log N)$ bits of advice are sufficient to achieve an optimal solution.

-- Algorithm 2: For a metric space that admits a system of $\mu$ collective tree $(q, r)$-spanners, we introduce a $(q + r)$-competitive algorithm with advice complexity $s(n) = O(n \log \mu + \log \log N)$. As a consequence, we have competitive algorithms for a large family of graphs, e.g., a 3-competitive algorithm for planar graphs with advice of size $O(n \log \log N)$.

-- Lower Bound 1: We show that sublinear advice does not suffice to provide close-to-optimal solution, even if we restrict the problem to 2-server problem on paths of size $N \geq 3$. Precisely, $\Omega(n)$ bits of advice are required to obtain a $c$-competitive algorithm for any value of $c \leq 3/2 - \epsilon$, where $\epsilon$ is an arbitrary small constant. This implies that when the metric space is a path, advice of linear size is necessary and sufficient for an optimal algorithm; this is because there is an optimal algorithm which receives $O(n)$ bits of advice for these metric spaces [36].

-- Lower Bound 2: We prove a lower bound for the size of advice required for an online algorithm to perform optimally in graphs with small treewidth $\alpha$ where $4 \leq \alpha < 2k$. We present metric spaces with treewidth $\alpha$ for which any optimal online algorithm requires advice of size at least $\frac{n}{2}(\log \alpha - 1.22)$ on a metric of treewidth $\alpha$, where $4 \leq \alpha < 2k$.

For graphs with constant treewidth, the advice-size of our algorithm (Algorithm 1) is almost linear. Considering that advice of linear size is required for an optimal algorithm (Lower Bound 1), the algorithm has advice of nearly optimal size. For graphs with treewidth $\alpha = \Omega(\log N)$, the advice-size (for Algorithm 1) is $O(n \log \alpha)$, which is asymptotically tight when $4 \leq \alpha < 2k$, because at least $\frac{n}{2}(\log \alpha - 1.22)$ bits are required to be optimal in these cases (Lower Bound 2).

2 Upper Bounds

2.1 Graphs with Small Treewidth

In this section, we introduce an algorithm called GRAPHPATHCOVER, denoted by $G\text{PC}$, to show that $O(n(\log \alpha + \log \log N))$ bits of advice are sufficient to optimally serve a sequence of length $n$ on any metric space of treewidth $\alpha$. We start with the following essential lemma.

Lemma 1 Let $T$ be a tree decomposition of a graph $G$. Also, let $x$ and $y$ be two nodes of $G$ and $P = (x = p_0, p_1, \ldots, p_{l-1}, y = p_l)$ be any path between $x$ and $y$. Let $X$ and $Y$ be two bags in $T$ which respectively contain $x$ and $y$. Any bag on the unique path between $X$ and $Y$ in $T$ contains at least one node $p_i (0 \leq i \leq l)$ from $P$.

Proof By the definition of the tree decomposition, each vertex $v$ of $G$ is listed in the bags of a contiguous subtree $T_v$ of $T$. Consider two vertices $p_i$ and $p_{i+1}$ in $P$. Since $p_i$ and $p_{i+1}$ are neighbours, there is a bag in $T$ which contains both of them. So the
union of the subtrees $T_{p_i}$ and $T_{p_{i+1}}$ forms a (connected) subtree of $T$. Similarly, the union of all the subtrees of the nodes $p_0, \ldots, p_l$ forms a (connected) subtree in $T$. Such a subtree contains $X$ and $Y$ and hence, any bag on the path between them. So any bag between $X$ and $Y$ contain at least one vertex $p_i$ of $P$.

The PathCover algorithm introduced for trees in [36] moves its servers on the same trajectories as OPT does. For example, when a server moves from a node $x$ to a node $y$ in the tree, the advice indicates the index of the common ancestor $z$ of $x$ and $y$, and the algorithm moves the servers to $z$ after serving the request at $x$. Note that indicating the common ancestor requires $O(\log h)$ bits of advice for a tree of height $h$. $\mathcal{GPC}$ works similarly and uses advice to indicate the trajectories that are used by OPT for serving requests. Suppose that OPT uses a server $s_i$ to serve the requests $[x_{a_1}, \ldots, x_{a_{ni}}]$ ($i \leq k$ and $n_i \leq n$). So, $s_i$ is moved on the path from its initial position to $x_{a_1}$, and then from $x_{a_1}$ to $x_{a_2}$, and so on. Note that there might be more than one shortest path between two requests served by a server of OPT; in what follows, we fix one such path and consider a fixed optimal offline algorithm OPT with a fixed shortest path between any pair of vertices.

Consider a rooted tree decomposition of $G$; we later indicate what bag is the root of the tree. For any node $v$ in $G$, $\mathcal{GPC}$ treats one of the bags which contains $v$ as the representative bag of $v$. Moreover, it assumes an ordering of the the nodes in each bag. Each node in $G$ is addressed via its representative bag, and its index among the nodes of that bag. A server $s_i$, located at a vertex $v$ of $G$, is addressed via a bag which contains $v$ (not necessarily the representative bag of $v$) and the index of $v$ in that bag. Note that there is a unique way to address a node while there might be several different ways to address a server.

Assume that for serving a request $y$, OPT moves a server $s_i$ from a node $x$ to $y$ in $G$. Let $X$ and $Y$ be respectively the representative bags of $x$ and $y$, and $Z$ be the lowest common ancestor of $X$ and $Y$ in $T$. Note that $X$ and $Y$ might not be present in bag $Z$; however, by Lemma 1, the shortest path between $x$ and $y$ passes at least one node $z$ in $Z$. This node can be indicated by $\lceil \log h \rceil + \lceil \log \alpha \rceil$ bits of advice ($h$ denotes the height of the tree in the tree decomposition), with $\lceil \log h \rceil$ bits indicating $Z$ and $\lceil \log \alpha \rceil$ bits indicating the index of $z$ among vertices in $Z$. After serving $x$, $\mathcal{GPC}$ moves $s_i$ to $z$, provided with the address of $z$ as a part of the advice for $x$. For serving $y$, $\mathcal{GPC}$ moves $s_i$ to $y$, provided with the address of $s_i$ (address of $z$) as part of the advice for $y$. Figure 4 provides an illustration. In what follows, we elaborate this in more detail.

Before serving any request, $\mathcal{GPC}$ moves each server $s_i$ from its initial position $x_0$ to a node $z_0$ on the shortest path between $x_0$ and the first node $x_{a_1}$ served by $s_i$ in the OPT’s scheme. $\mathcal{GPC}$ selects $z_0$ in a way that it will be among the vertices on the shortest path between $x_0$ and $x_{a_1}$ and among the vertices in the lowest common ancestor of the representative bags of the two vertices in the tree decomposition (by Lemma 1 such a node $z_0$ exists). To move all servers in this way, $\mathcal{GPC}$ reads $(\lceil \log h \rceil + \lceil \log \alpha \rceil) k$ bits of advice. After these initial moves, moves servers on the same trajectories of OPT as argued earlier. Assume that $x$, $y$ and $w$ denote three requests which are consecutively served by $s_i$ in OPT’s scheme. The advice for serving $x$ contains $\lceil \log h \rceil + \lceil \log \alpha \rceil$ bits which represents a node $z_1$, which lies on the
The \( \mathcal{GPC} \) algorithm moves servers on the same trajectories as OPT. Assume OPT moves a server from node \( H \) to node \( S \) on the shortest path between them (highlighted in (a)). \( \mathcal{GPC} \) moves the server from \( H \) to a node on the shortest path between \( H \) and \( S \) that appears in the lowest common ancestor of representative bags of \( H \) and \( S \) in the tree decomposition. Here, the lowest common ancestor of \( H \) and \( S \) is the bag that contains \{\( A, B, D \)\}. By Lemma 1, this bag contains a node on the shortest path between \( H \) and \( S \), namely \( D \). So, provided with the advice, the algorithm moves the server to \( D \) after serving the request at \( H \).

shortest path between \( x \) and \( y \) and is situated inside the lowest common ancestor of the respective bags in \( T \) moves \( s_i \) to \( z_1 \) after serving \( x \). The first part of the advice for \( y \) contains \( \lceil \log h \rceil + \lceil \log \alpha \rceil \) bits indicating the node \( z_1 \) from which \( s_i \) is moved to serve \( y \). The second part of the advice for \( y \) indicates a node \( z_2 \) on the shortest path between \( y \) and \( w \) in the lowest common ancestor of their bags in \( T \). This way, \( 2(\lceil \log h \rceil + \lceil \log \alpha \rceil) \) bits of advice per request are sufficient to move the servers on the same trajectories as OPT.

The above argument implies that advice of size \( (\lceil \log h \rceil + \lceil \log \alpha \rceil)(2n + k) \) is sufficient to achieve an optimal algorithm. The value of \( h \) (the height of the tree decomposition) can be as large as \( N \), however we can apply the following lemma to obtain height-restricted tree decompositions.

**Lemma 2** [10, 21] Given a tree decomposition with treewidth \( \alpha \) for a graph \( G \) with \( N \) vertices, one can obtain a tree decomposition of \( G \) with height \( O(\log N) \) and width at most \( 3\alpha + 2 \).

If we apply \( \mathcal{GPC} \) on a height-restricted, rooted tree decomposition, we obtain the following result.

**Theorem 1** For any metric space of size \( N \) and treewidth \( \alpha \), there is an online algorithm which can optimally serve any input sequence of length \( n \), if provided with \( O(n(\log \alpha + \log \log N)) \) bits of advice.
2.2 Graphs with Small Number of Collective Tree Spanners

In this section, we introduce an algorithm which receives advice of almost linear size and achieves constant competitive ratio for a large family of graphs. The main idea behind this algorithm is similar to the PATHCOVER algorithm of [36], namely, the algorithm moves servers in the same way that OPT does, except that instead of shortest paths, the algorithm uses approximate trajectories which are defined by a system of collective tree spanners. To be more precise, assume a metric space $G$ of size $N$ admits a system of $\mu$ collective tree $(q, r)$-spanners. This implies that the (shortest) path between any two vertices in $G$ can be approximated by at least one of the tree spanners so that the approximated distance is no more than $q + r$ times the actual distance. Using this observation, we devise an offline algorithm, called APX, which moves servers in the same way that OPT does except that it uses the best alternative trajectory on one of the spanners for moving a server between any two consecutive requests. In the APX solution, the optimal trajectory of each server $s$ is replaced with an approximate trajectory formed by a set of segments, one for each request served by $s$. Each segment indicates the best approximate path, given by collective tree spanners, between two consecutive requests served by $s$. So, the cost incurred by APX for each segment is at most $(q + r)$ times more than the length of the shortest path. Consequently, the total distance travelled by each server in APX solution is at most $(q + r)$ times the distance travelled in the optimal scheme. This implies that APX has an approximation ratio of at most $q + r$. In addition, APX has certain properties that facilitate devising an online algorithm $A$ which receives advice of almost linear size (to the length of sequence) while incurring the same cost as APX. This ensures a competitive ratio of $q + r$ for $A$. Before we present a formal description, we intuitively explain how the advice can be used to move servers on the same or approximate trajectories as an offline algorithm would.

Assume at some point, OPT moves a server $s$ from $x$ to $y$ to serve requests in these vertices. APX moves $s$ from $x$ to $y$ on the trajectory devised by the tree with the smallest distance between $x$ and $y$ in a system of collective tree spanners. One way to move servers on such a tree is to address nodes by their heights. After serving a request to a vertex $x$, APX can move the server to the lowest common ancestor of $x$ and $y$ in the tree that best approximates their distance. This way, the advice for each request has two parts; the first part indicates where the server used by the offline algorithm is located before serving the request, and the second part indicates where that server is moved after serving the requests. For both parts, we need to indicate the tree that best approximates the distances in $\Theta(\log \mu)$ bits, and the height of the lowest common ancestors of $x$ and previous/subsequent requests in $\Theta(\log h)$ bits, where $h$ is an upper bound on the height of the trees. Since $h$ can be as large as $N$, the size of the advice is as large as $\Theta(n(\log N + \log \mu))$ bits. We show that if we use the caterpillar decomposition to address nodes, this can be improved to $O(n(\log \log N + \log \mu))$ bits. In what follows, we fix an optimal caterpillar dimension for each member of the collective tree spanners.

Assume that, after serving $x$, instead of moving $s$ to the common ancestor of $x$ and $y$, we move it to the node $\text{maxPath}_T(x, y)$ where $\text{maxPath}_T(x, y)$ is the nearest node to $x$ with the highest path-level on the path between $x$ and $y$ in the tree.
We always select $T$ as the tree spanner that provides the best approximation of the distance between $x$ and $y$. Note that the common ancestor of $x$ and $y$ is among the nodes with the highest path-level on such a path (since the path-levels are non-decreasing on any directed path starting from the root); however, it might not be the closest node to $x$. For instance, in the tree of Fig. 5, the highest path-level between $x$ and $y$ is 2 and $\maxPath_T(x, y)$ is node $u$. On a request to $y$, the advice indicates the tree $T$ that approximates the distance between $x$ and $y$ (in $\Theta(\log \mu)$ bits). Furthermore, the advice indicates the path-level $mp$ of $\maxPath_T(x, y)$; hence, the algorithm knows that the server for serving $y$ is located on the unique path $P$ of level $mp$ that intersects the path between $y$ and the root of the tree that approximates the distance between $x$ and $y$. In Fig. 5, $P$ is highlighted as a blue path. If servers do not ‘cross’ each other in the APX scheme, there are two possible servers on $P$ to serve $y$, one is closer to the root and one is further. We refer to these two servers as the ‘top’ and the ‘bottom’ servers, respectively. The advice for $y$ includes

**Fig. 5** An illustration of how an online algorithm $A$ can move servers in the same way as APX. Assume APX uses the server $s$ for requests to $x$, $y$, and $z$. After serving $x$, $s$ is moved to the closest node with path-level $mp_1$, where $mp_1$ is a part of the advice and indicates the maximum path-level in the path between $x$ and $y$ in the tree. In this example, $mp_1 = 2$ and $s$ is moved to node $u$. For serving $y$, $A$ reads the path-level of the node that $s$ is located at (i.e., 2), and considers the unique path of level 2 (the blue path) that intersects the path between $y$ and the root of the tree. $A$ also reads another bit which indicates a bottom server should be moved, i.e., it selects the closest vertex on the part of the path that is further from the root (the highlighted part). The second part of the advice for the request at $y$ indicates the tree that best approximates the distance between $y$ and $z$ on that tree. Since $mp_2 = 1$ and $y$ has level 1, $s$ is kept at node $y$. For serving $z$, $A$ reads the path-level of $s$ (which is 1) and locates $s$ as the top server, as indicated in advice, on the unique path of level 1 (the red path) that intersects the path between $z$ and the root.
another bit which indicates which of these two servers should be moved to serve the request in \( y \).

Using the above intuitions, we formally explain how the offline algorithm \( \text{APX} \) can be transformed into an online algorithm \( \mathcal{A} \). As mentioned earlier, \( \text{APX} \) moves servers on the approximate trajectories defined by the collective tree spanners, and it has an approximation ratio of \( q + r \). We assume \( \text{APX} \) moves only one server for each request and servers do not cross each other. Note that if an algorithm moves more than one server for a request, all moves except one can be delayed without an increase in the cost of the algorithm. Similarly, if a server \( s_1 \) crosses another server \( s_2 \) for serving a request \( r \), \( s_2 \) can be moved to serve \( r \) while \( s_1 \) serves the next request served by \( s_2 \). In the resulting scheme, the servers do not cross each other and the cost is not increased. Hence, we may assume that \( \text{APX} \) moves only one server for each request and servers do not cross each other. Using advice, we devise an online algorithm \( \mathcal{A} \) that moves servers on the same trajectories as \( \text{APX} \). Consider a request to a node \( y \) is served by server \( s \) in the \( \text{APX} \) scheme. Assume \( x \) and \( z \) are requests that are served by \( s \) respectively before and after serving \( y \) (\( x \) is the initial position of \( s \) if \( y \) is the first request served by \( s \)). The advice for serving \( y \) has two parts. The first part has the form \( \langle \beta_1, \beta_2, \beta_3 \rangle \) and indicates the server that should be moved to serve \( y \). \( \beta_1 \) has size \( \Theta(\log \mu) \) and indicates the tree \( T_1 \) which best approximates the distance between \( x \) and \( y \). \( \beta_2 \) has size \( \Theta(\log \log N) \) and indicates the path-level \( mp_1 \) (in the caterpillar decomposition of \( T_1 \)) of the node at which \( s \) is located just before serving \( y \). \( \beta_3 \) is a single bit and indicates whether the top or bottom server (on the path of level \( mp_1 \) that intersects the path between \( y \) and the root of \( T_1 \)) should be moved. Assuming that no two servers cross each other in \( \text{APX} \), the first part of the advice for \( y \) is sufficient to indicate the unique server \( s \) that \( \text{APX} \) uses for serving \( y \).

The second part of the advice has the form \( \langle \gamma_1, \gamma_2 \rangle \) and determines the server that \( s \) should be moved to after serving \( y \). Here, \( \gamma_1 \) has size \( \Theta(\log \mu) \) and indicates the tree \( T_2 \) which best approximates the distance between \( y \) and \( z \). \( \gamma_2 \) has size \( \Theta(\log \log N) \) and indicates the maximum path-level \( mp_2 \) of vertices on the path between \( y \) and \( z \) in \( T_2 \). To serve a request at \( y \), the online algorithm \( \mathcal{A} \) reads the first part of the advice to detect the unique server \( s \) to serve the request. \( \mathcal{A} \) proceeds by moving \( s \) to the closest vertex with path-level \( mp_2 \) on tree \( T_2 \) (\( mp_2 \) and \( T_2 \) are indicated in the second part of the advice). This ensures that \( \mathcal{A} \) moves servers in the same way as \( \text{APX} \), thereby, yielding the following result.

**Theorem 2** If a metric space \( G \) of size \( N \) admits a system of \( \mu \) collective tree \((q, r)\)-spanners, then there is an online algorithm which receives \( O(n (\log \mu + \log \log N)) \) bits of advice, and achieves a competitive ratio of at most \( q + r \) on any sequence of length \( n \).

**Proof** As explained above, the offline algorithm \( \text{APX} \) moves servers on the approximate trajectories to achieve an approximation ratio of at most \( q + r \). The online algorithm \( \mathcal{A} \) moves servers in a similar way that \( \text{APX} \) does, aided by advice of size \( O(\log \mu + \log \log N) \) per request. As described above, the advice for each request indicates the approximate trees and a path-level on each
tree. In total, the advice has size $O(n(\log \mu + \log \log N))$ for any sequence of length $n$. 

In recent years, there has been wide interest in providing collective tree spanners for various families of graphs. The algorithms which create these spanners run in polynomial time. It is known that any planar graph of size $N$ has a system of $O(\log N)$ collective (3,0)-spanners [23]; every AT-free graph (including interval, permutation, trapezoid, and co-comparability graphs) admits a system of two (1,2)-spanners [18]; every chordal graph admits a system of at most $\log N$ collective (1,2)-spanners [19]; and every unit disk graphs admits a system of at most $(2 \log_{1.5} N + 2)$ collective (3,12)-spanners, [39].

**Corollary 1** For metric spaces of size $N$, $O(n \log \log N)$ bits of advice are sufficient to obtain
- a 3-competitive algorithm for planar graphs,
- a 3-competitive algorithm for AT-free graph (including interval, permutation, trapezoid, and co-comparability graphs),
- a 3-competitive algorithm for chordal graphs, and
- a 15-competitive algorithm for unit disk graphs,

for input instances of length $n$.

### 3 Lower Bounds

#### 3.1 2-server Problem on Path Metric Spaces

In this section, we show that advice of sublinear size does not suffice to achieve close-to-optimal solutions, even for the 2-server problem on a path metric space of size $N \geq 3$. Without loss of generality, we only consider online algorithms which are ‘lazy’ in the sense that they move only one server at the time of serving a request. As mentioned earlier, any online algorithm can be converted to a lazy algorithm without an increase in its cost. Hence, a lower bound for the performance of lazy algorithms applies to all online algorithms. In the remainder of this section, the term *online algorithm* means a lazy algorithm.

Consider a path of size $N \geq 3$ in which the vertices are indexed from 1 to $N$. Assume that the servers are initially positioned at vertices 1 and 3. We refer to these servers as ‘left’ and ‘right’ servers, respectively (assuming the vertices form a horizontal line). We build a set of instances of the problem so that each instance is formed by $m = n/5$ rounds of requests. Each round is defined by requests to vertices (2, 1|3, 2, 1, 3), where the second request of a round is either to vertex 1 or vertex 3. Note that a reasonable algorithm does not moves servers to indices larger than 3. So, we might assume servers are at indices 1, 2, and 3 at any given time. Since each round ends with requests to vertices 1 and 3, it is reasonable to move servers to these vertices for serving the last requests of each round. This is formalized in the following lemma.
Lemma 3 Consider an algorithm $A$ that serves an instance of the $k$-server problem as defined above. There is another algorithm $A'$, with a cost no more than that of $A$, in which the servers are positioned at vertices 1 and 3 before serving requests of each round.

Proof Consider the first round $R_t$ such that $A$ does not have the servers positioned at 1 and 3, i.e., the servers are positioned at vertices 2 and 3. This is because the algorithm is lazy and the two servers cannot be at the same position; moreover, since the last request of the previous round has been to vertex 3, one server is located at vertex 3. As another consequence of being a lazy algorithm, the last two requests of the previous round $R_{t-1}$ are served by the same server. So, one server has crossed the other one in the scheme of $A$. Consider an algorithm $A'$, which moves the same servers as $A$ for serving all requests before the last two requests of $R_{t-1}$. To serve these requests, $A'$ moves the servers to vertices 1 and 3. This requires a cost of at most 1 which happens when one of the server is positioned at 2. Hence, $A'$ incurs a cost of at most 1 for the last two requests in $R_{t-1}$ and, when compared to $A$, saves a cost of at least 1 in round $R_{t-1}$. At the beginning of round $R_t$, the servers of $A$ are positioned at 2 and 3 and the servers of $A$ are at 1 and 3. In the future rounds, $A'$ moves the server positioned at 1 in the same way that $A$ moves the server position at 2. The total cost would be the same for both algorithms except that the cost for the first request served by the server positioned at 1 in $A'$ might be at most 1 unit more when served by the server positioned at vertex 2 in $A$.

To summarize, when compared to $A$, $A'$ saves a cost of at least 1 for requests in $R_{t-1}$ and incurs an extra cost of at most 1 for the requests in rounds after $R_{t-1}$. Hence, the cost of $A'$ is no more than that of $A$. To complete the proof, it suffices to apply the above procedure on all rounds for which there is a server at position 2 at the beginning of the round. The result would be an algorithm which has servers located at positions 1 and 3 at the beginning of any round.

By the above lemma, in order to provide a lower bound on the performance of online algorithms, it suffices to consider only those algorithms which keep servers at positions 1 and 3 at the beginning of each round. For any input sequence, we say a round $R_t$ has type 0 if the round is formed by requests to vertices $(2, 1, 2, 1, 3)$ and has type 1 otherwise, i.e., when it is formed by requests to vertices $(2, 3, 2, 1, 3)$. The first request of a round is to vertex 2. Assume the second request is to vertex 3, i.e., the round has type 1. An algorithm can move the left vertex $s_l$ positioned at vertex 1 to serve the first request (to vertex 2) and use the right server $s_r$ positioned at 3 to serve the second request the same vertex. For serving other requests of the round, the algorithm can move the servers to their initial positions and incur a total cost of 2 for the round (see Fig. 6a). Note that this is the minimum cost that an algorithm can incur for a round. Next, assume that the algorithm moves the right vertex $s_r$ to serve the first request (to vertex 2). The algorithm has to serve the second request (to vertex 3) also with $s_r$. The third request (to vertex 2) can be served by any of the servers. Regardless, the cost of the algorithm will not be less than 4 for the round (see Fig. 6b).
With a symmetric argument, in case the second request is to vertex 1 (i.e., the round has type 0), if an algorithm moves the right server to serve the first request, it incurs a total cost of 2, and if it moves the left server for the first request, it incurs a cost of at least 4 for the round.

To summarize, an algorithm should ‘guess’ the type of a round at the time of serving the first request of the round. In case it makes a right guess, it incurs a total cost of 2, and if it makes a wrong guess, it incurs a cost of at least 4. This relates the problem to the Binary String Guessing Problem.

**Definition 5** [7, 20] The Binary String Guessing Problem with Known History (2-SGKH) problem is the following online problem. The input is a bitstring of length $n$, and the bits are revealed one by one. For each bit $b_t$, the online algorithm $A$ must guess if it is a 0 or a 1. After the algorithm has made a guess, the value of $b_t$ is revealed to the algorithm.

**Lemma 4** [7] On an input of length $m$, any deterministic algorithm for the 2-SGKH problem that is guaranteed to guess correctly on more than $\alpha m$ bits, for $1/2 < \alpha < 1$, needs to read at least $(1 + (1 - \alpha) \log(1 - \alpha) + \alpha \log \alpha)m$ bits of advice.

We reduce the 2-SGKH problem to the 2-server problem on paths.

**Lemma 5** If there is a 2-server algorithm with cost at most $\gamma n$ ($2/5 < \gamma < 3/5$) for an instance of length $n$ (as defined earlier), then there is a 2-SGKH algorithm which guesses at least $\left(\frac{4 - 5\gamma}{2}\right)m$ bits correctly for any input bit string of size $m = n/5$.

**Proof** Let $B$ denote a bit string of length $m = n/5$, which is the input for the 2-SGKH problem. Consider the instance of the 2-server problem in which the types of rounds are defined by $B$, i.e., the $t$th round has type 0 if the $t$th bit of $B$ is 0 and has type 1 otherwise. We run the 2-server algorithm on this instance. At the time of serving the first request of the $t$th round, the 2-server algorithm guesses the type of round $t$ by moving the left or right server. In particular, it guesses the type of the round to be 0 if it moves the right server for the first request and 1 otherwise. Define a 2-SGKH algorithm which performs according to the 2-server algorithm, i.e., it guesses
the $t$th bit of $B$ as being 0 (respectively 1) if the 2-server algorithm guesses the $t$th round as having type 0 (respectively 1). As mentioned earlier, the 2-server algorithm incurs a cost of 2 for each right guess and a cost of at least 4 for each wrong guess. So, the cost of the algorithm is at least $2\beta m + 4(1 - \beta)m = (4 - 2\beta)m$ in which $\beta m$ is the number of correct guesses ($\beta \leq 1$). Consequently, if a 2-server algorithm has cost of at most $(4 - 2\beta)m$, it correctly guesses the types of at least $\beta m$ rounds, i.e., it correctly guesses at least $\beta m$ bits of a bit string of length $m$. Defining $\gamma$ to be $(4 - 2\beta)/5$ completes the proof.

Lemmas 4 and 5 give the following theorem.

**Theorem 3** On an input of length $n$, any deterministic algorithm for the 2-server problem which has a competitive ratio smaller than $\tau$ ($1 < \tau < 3/2$) needs to read at least $(1 + (\tau - 1)\log(\tau - 1) + (2 - \tau)\log(2 - \tau))n/5$ bits of advice, and this holds for metric spaces of size as small as 3.

**Proof** Consider instances of the problem as defined above and let $m$ denote the number of rounds in an input sequence. As mentioned earlier, there is an offline algorithm which incurs a cost of 2 for each round and a total cost of $2m = 2n/5$ for all rounds. Hence, in order to have a competitive ratio of at most $\tau$, the cost of an online algorithm should be at most $2\tau n/5$. According to Lemma 5, this requires the existence of a 2-SGKH algorithm which correctly guesses at least $(2 - \tau)m$ bits of a bit string of length $m$ (we define $\gamma = 2\tau/5$). By Lemma 4, this would require at least $(1 + (1 - (2 - \tau))\log(1 - (2 - \tau)) + (2 - \tau)\log(2 - \tau)m = (1 + (\tau - 1)\log(\tau - 1) + (2 - \tau)\log(2 - \tau))n/5$ bits of advice (we define $\alpha = 2 - \tau$). Note that $2 - \tau$ is in the range required by Lemma 4 when $1 < \tau < 3/2$.

For a competitive ratio of $\tau = 3/2$, the formula in 3 takes the value 0 and thus does not provide a non-trivial bound. However, for doing strictly better than 3/2, a linear number of bits of advice is required. For example, to achieve a competitive ratio of $\tau = 4/3$, at least $0.0163n$ bits of advice are needed, and for the improved ratio of $\tau = 5/4$, at least $0.0375n$ bits of advice are needed.

### 3.2 Metrics With Small Treewidth

In this section, we show that there are instances of the $k$-SERVER problem defined on a metric space with treewidth $\alpha$ ($4 \leq \alpha \leq 2k$), for which any online algorithm requires at least $\frac{\alpha}{4}(\log\alpha - 1.22)$ bits of advice to perform optimally. Our construction is based on the one described in [8] where a lower bound for general metric spaces is discussed. We start by introducing unit graphs and module graphs.

**Unit Graphs** A $\gamma$-unit graph is a bipartite graph $G = (U \cup W, E)$ where $U = \{u_1, \ldots, u_\gamma\}$ contains $\gamma$ vertices, and $W$ contains $2^\gamma - 1$ vertices each representing a proper subset of $U$. There is an edge between two vertices $u \in U$ and $w \in W$ iff $u \notin \text{Set}(w)$, where $\text{Set}(w)$ denotes the set associated with
a vertex \( w \in W \). Let \( B_i \subseteq W \) denote the set of vertices of \( W \) whose associate sets have size \( i \). i.e., for \( w \in B_i \) we have \( |\text{Set}(w)| = i \). A valid request sequence for a unit graph is defined as \( \{x_0, x_1, \ldots, x_{\gamma-1}\} \) so that for each \( i \), \( x_i \in B_i \) and \( \text{Set}(x_i) \subseteq \text{Set}(x_{i+1}) \). In other words, a valid sequence starts with a request to the vertex associated with the empty set, and with each step one element is added to get a larger set defining the next request. With this definition, one can associate every valid sequence \( I \) with a unique permutation \( \pi \) of set \( \{1, 2, \ldots, \gamma\} \).

**Module Graphs** A \( \gamma \)-module graph \( G \) includes two \( \gamma \)-unit graphs \( G_1 = (U_1 \cup W_1, E_1) \) and \( G_2 = (U_2 \cup W_2, E_2) \). In this graph, those vertices in \( W_1 \) which represent sets of size \( i \) are connected to the \((i+1)\)st vertex of \( U_2 \); the vertices of \( W_2 \) and \( U_1 \) are connected in the same manner (see Fig. 7).

Consider an instance of the \( k \)-SERVER problem defined on a \( k \)-module graph i.e., the number of servers is \( k = \gamma \). Assume that all servers are initially located at the vertices of \( U_1 \). A valid request sequence for a module is defined by repeating rounds of requests. Each round starts with a valid sequence for (unit graph) \( G_1 \) denoted by \( \pi_1 \), followed by \( k \) requests to distinct vertices of \( U_2 \), a valid sequence for \( G_2 \), and \( k \) requests to distinct vertices of \( U_1 \). It can be verified that there is a unique optimal solution for serving any valid sequence on \( G \), and consequently a separate and unique advice string is required for each sequence [8]. Since there are \((k!)^{(n/2k)}\) valid sequences of length \( n \), at least \((n/(2k))\log(k!) \geq n(\log k - \log e)/2 \) bits of advice are required to “separate” all valid sequences. When servers are labelled, this lower bound is asymptotically tight for metric spaces with treewidth \( \alpha \geq 2k \); this is because \( n \lceil \log k \rceil \) bits of advice are sufficient to serve each sequence optimally (by simply indicating the servers that \( \text{OPT} \) would move to serve each request). In what follows, we provide a better lower bound for metric spaces with treewidth \( \alpha \) such that \( 4 \leq \alpha \leq 2k \). We start by showing that the tree width of graphs designed in the above construction of [8] is at most \( 2k \). Later, we use module graphs as components which form a larger graph, also with bounded tree width, which is used in our lower bound argument hold.

**Lemma 6** Any \( \gamma \)-module graph has a tree decomposition of width \( 2\gamma \).

**Proof** Let \( G_1 = (U_1 \cup W_1, E_1) \) and \( G_2 = (U_2 \cup W_2, E_2) \) be the unit graphs which define the \( \gamma \)-module graph. Define a tree decomposition as follows. Consider \( 2 \times 2^k \) bags so that each bag contains all vertices from \( U_1 \) and \( U_2 \), and exactly one vertex from \( W_1 \) or \( W_2 \). Any tree which spans all these \( 2 \times 2^k \) bags is a valid tree decomposition; this is because \( W_1 \cup W_2 \) form an independent set of the graph. Moreover, there are exactly \( 2\gamma + 1 \) vertices in each bag which completes the proof. Figure 7) provides an illustration.

Assume that \( \alpha \) is an even integer and we have \( k = m\alpha/2 \) for some positive integer \( m \). Consider a metric space \( G_b \) defined by a set of \( \gamma \)-modules where \( \gamma = \alpha/2 \). There are \( k/\gamma = m \) such modules in \( G_b \). Let \( M_1, \ldots, M^m \) denote these modules, and let \( G_1^i = (U_1^i \cup W_1^i, E_1) \), \( G_2^i = (U_2^i \cup W_2^i, E_2) \) denote the unit graphs involved in the
The metric space $G_b$ has a tree decomposition of width $\alpha$. 

**Lemma 7** The metric space $G_b$ has a tree decomposition of width $\alpha$. 

construction of $M^i$ ($i \leq m$). For each module $M^i$, select exactly one vertex from $U^i$, and connect all of the selected vertices to a common source vertex. This makes $G_b$ a connected graph. Figure 8 illustrates this construction.
Fig. 8 The metric space $G_b$ and a tree decomposition associated with it. The source $s$ is connected to the selected vertex $a^1_i$ of module $M^i$.

Proof  By Lemma 6, each module has a tree decomposition of width $\alpha$. Let $T^i$ denote the tree associated with the decomposition of the $i$th module. For any tree $T^i$, create a bag $B^i$ of size 2 which contains the source $s$ and the other endpoint of the edge between $s$ and $M^i$. Connect $B^i$ to an arbitrary bag of $T^i$. Add $m-1$ arbitrary edges between all $B^i$'s to form a connected tree. Such a tree represents a valid tree decomposition of $G_b$ with width $\alpha$. Figure 8 provides an illustration.

Since there are $m$ modules and in the $i$th module $U^i_1$ contains $\gamma$ vertices, there are $m \times \gamma = k$ vertices in all of the $U^i_1$s. Assume that the $k$ servers are initially placed at separate nodes in the $U^i_1$s. A valid sequence for $G_b$ is defined by a sequence of rounds of requests in a way that the subsequence formed by requests in each module forms a valid sequence for that module. Recall that each module $M_i$ has two unit graphs $G^i_1 = (U^i_1 \cup W^i_1, E_1)$ and $G^i_2 = (U^i_2 \cup W^i_2, E_2)$, and a valid sequence for $M_i$ starts with a valid sequence for $G_1$ denoted by $\pi^i_1$, followed by $\gamma$ requests to distinct vertices of $U_2$ denoted with $b^i_1, \ldots, b^i_\gamma$, a valid sequence for $G_2$ denoted by $\pi^i_2$, and $k$ requests to distinct vertices of $U_1$ denoted by $a^i_1, \ldots, a^i_\gamma$. Each round in a valid sequence for $G_b$ includes a round from each module $M_i (1 \leq i \leq m)$. The rounds of requests to modules are combined in a way that the $t$th request in $M_i$ appears before the $(t+1)$th request in $M_{i'}$ ($i \neq i'$, $1 \leq t \leq 4\gamma$. More precisely, each round in a valid sequence for $M_i$ has the following form:

$$f(\pi^1, \ldots, \pi^m), (b^1_1, \ldots, b^m_1), (b^1_\gamma, \ldots, b^m_\gamma), f(\pi^1_2, \ldots, \pi^m_2), (a^1_1, \ldots, a^m_1), \ldots, (a^1_\gamma, \ldots, a^m_\gamma)$$

Here, $f$ is a function that combines the requests from $m$ permutations. Let $(\pi^1, \ldots, \pi^m)$ denote $m$ permutations such that $\pi^i$ contains $\gamma$ requests, denoted
by \( \langle r^i_1, \ldots, r^i_\gamma \rangle \), which defines a permutation in the module \( M^i \). Thus, \( f \) gives a sequence of length \( m \times \gamma \) starting with \( m \) requests to \( r^i_1 \)'s, followed by \( m \) requests to \( r^i_2 \)'s, and so on. For each \( j \), \( (1 \leq j \leq \gamma) \) we have fixed orderings on the vertices such that \( (a^1_j, \ldots, a^m_j) \in \langle U^1_1, \ldots, U^m_1 \rangle \) and \( (b^1_j, \ldots, b^m_j) \in \langle U^1_2, \ldots, U^m_2 \rangle \). With this definition, when a valid sequence of \( G_b \) is projected to the requests arising in a module \( M \), the resulting subsequence is a valid sequence for \( M \).

**Lemma 8** There is a unique optimal solution to serve a valid sequence on the metric space \( G_b \). Also, each valid sequence requires a distinct advice string in order to be served optimally.

**Proof** We present an algorithm \( \mathcal{PERM} \) and show that its solution is the unique optimal solution for serving any valid sequence. To serve a request in \( W^1_i \) (respectively \( W^2_i \)), \( \mathcal{PERM} \) moves a server from \( U^1_i \) (respectively \( U^2_i \)) to act according to the corresponding permutation \( \pi^1_i \) (respectively \( \pi^2_i \)). To be more precise, to serve a request \( x_i \) (for \( 0 \leq i \leq \gamma - 2 \)) it moves the server positioned at \( \text{Set}(x_i+1) \setminus \text{Set}(x_i) \), and thus leaving a unique choice for the last request \( x_{\gamma-1} \). To serve a request \( a^1_i \) in \( U^1_i \) (respectively \( U^2_i \)), \( \mathcal{PERM} \) moves the single server which is located at an adjacent node in \( W^2_i \) (respectively \( W^1_i \)). Therefore, \( \mathcal{PERM} \) incurs a cost of one on each request.

There are \( \gamma \) servers initially located in each \( \gamma \) module, and \( \mathcal{PERM} \) never moves a server from one module to another (no server passes the common source). To show that the solution given by \( \mathcal{PERM} \) is the unique optimal solution for serving any valid sequence, it is sufficient to show the following statements hold:

1. No optimal algorithm moves a server from one module to another.
2. Among all algorithms which do not move a server between modules, \( \mathcal{PERM} \) provides the unique optimal solution.

Assume that there is an optimal algorithm \( \text{OPT} \) which moves a server from one module to another. So at some point after serving the \( t \)th request, there is a \( \gamma \)-module \( M \) which has \( \gamma + p \) servers stationed on it, for some \( p \geq 1 \). We show that the cost incurred by \( \text{OPT} \) for serving the requests of \( M \) in each round after the \( t \)th request is lower bounded by \( 4\gamma - p \). Note that \( \mathcal{PERM} \) incurs a total cost of \( 4\gamma \) in each round for the requests of any module. Assume that the cost incurred by \( \text{OPT} \) for serving a round in \( M \) is strictly less than \( 4\gamma - p \). This implies that more than \( p \) requests incur no cost in that round. Since the same vertex is not requested twice in the same round, more than \( p \) servers must not be moved in that round. So there is at least one server \( s \) which is moved by \( \mathcal{PERM} \) and not by \( \text{OPT} \). We show that moving \( s \) in the same way as \( \mathcal{PERM} \) does decreases the cost for \( \text{OPT} \). Assume that \( \text{OPT} \) keeps \( s \) at some vertex in \( U^1_1 \) of \( M \). Thus, \( \mathcal{PERM} \) moves \( s \) from \( U^1_1 \) to serve a request in \( W^1_1 \), and then moves it to serve a request in \( U^2_2 \), followed by a move to serve a request in \( W^2_2 \), and finally a move to serve a request in \( U^1_1 \). Each of these moves cost of one for \( \mathcal{PERM} \). However, each of the involved requests imposes a cost of \( 2 \) for \( \text{OPT} \) since it has to use the same server to serve at least two requests arising in \( W^1_1 \) in that round, thus, requiring a move via some vertex in \( U^1_1 \) or \( U^2_1 \). The same holds for the requests in \( U^1_1, U^2_1 \) and \( W^2_2 \). We can make similar arguments when \( \text{OPT} \) keeps \( s \) at some vertex in...
$W_1$ or $U_2$ of $M$, and conclude that $\text{OPT}$ saves a cost of $x$ by not moving $s$ in a round
while incurring a cost of $2x$ in the remainder of the round. Hence, $\text{OPT}$ must incur a
cost of at least $4\gamma - p$ for requests arising in $M$.

Let $M'$ be a $\gamma$-module in which $\text{OPT}$ has stationed $\gamma - q$ ($q \geq 1$) servers after
serving the $t$th request. We show that the cost incurred by $\text{OPT}$ to serve the requests
from $M'$ in each round starting after the $t$th request is lower bounded by $4\gamma + 8q$. Simi-
lar to the previous argument, since at least one server is missing, for some request(s) arising in
$W_1$, $\text{OPT}$ has to use server(s) already located in $W_1$. So instead of incurring a
cost of 1 as $\text{PERM}$, $\text{OPT}$ incurs a cost of 2 for each of those requests (the same
holds for requests in $U_1$, $U_2$, and $W_2$).

To conclude, if $\text{OPT}$ moves $x$ servers between modules, compared to $\text{PERM}$, it
saves at most $x$ units of cost on the requests arising in the modules which receive
extra servers, while it has to pay at least an extra $8x$ units for the requests in modules
which lose their servers. This is in addition to the cost involved in moving servers
from one module to another. We conclude that an optimal algorithm never moves
servers between modules (statement 1 holds).

Inside each module, $\text{PERM}$ acts the same as the unique optimal algorithm pre-
 spirited in [8]. Recall that the requests projected to each module form a valid sequence
for that module, and can be treated independently (since the servers do not move
between modules in an optimal scheme). Hence, both statements 1 and 2 hold, and
$\text{PERM}$ is the unique optimal algorithm for serving any valid request in $G_b$.

Next, we show that each valid sequence requires a distinct advice string. Assume
that two valid sequences $I$ and $I'$ differ for the first time at the $t$th request. Note
that two valid sequences of $G_b$ can only differ on the requests which define the
permutations. Hence, $t$ should be a request belonging to $\pi_1^i$ or $\pi_2^i$ of some mod-
ule $M^i$, i.e, one of the permutations representing a valid subsequence for the unit
graphs defining $M^i$. Let $t_0 < t$ denote the index of the previous request to an
item in the same unit graph (that is, the previous request in the same permu-
tation). While serving the request indexed $t_0$ in the two sequences, an optimal
algorithm will move different servers in anticipation of the $t$th request. Hence an
online algorithm should receive different advice strings to perform optimally for
both sequences.

To find a lower bound for the length of the advice string, we count the number
of distinct valid sequences for the metric space $G_b$. In each round, there are $(\gamma!)^2$
valid sequences for each $\gamma$-module. Since there are $m$ such modules, there are $(\gamma!)^{2m}$
possibilities for each round. A valid sequence of length $n$ involves $n/(4\gamma m)$ rounds;
hence there are $(\gamma!)^{n/(2\gamma)}$ valid sequences of length $n$. Each of these sequences
need a distinct advice string. Hence, at least $\log((\gamma!)^{n/(2\gamma)}) \geq (n/2)\log(\gamma/e) =
(n/2)\log(\alpha/(2e))$ bits of advice are required to serve a valid sequence optimally.
This proves the following theorem.

**Theorem 4** Consider the $k$-Server problem on a metric space of treewidth $\alpha$ such
that $4 \leq \alpha < 2k$. In order to optimally serve any sequence of length $n$, any online
algorithm needs to read at least $\frac{n}{2}(\log \alpha - 1.22)$ bits of advice.
4 Concluding Remarks

For the path metric spaces, we showed that any 1-competitive algorithm requires advice of size $\Omega(n)$. This bound is asymptotically tight as there is an optimal algorithm which receives $O(n)$ bits of advice [36] to optimally server any input of size $n$, and this holds for all values of $k$. The same lower bound applies to trees; however, the best algorithm for tree receives advice of size $O(n \log \log N)$. We conjecture that the lower bound argument can be improved for trees to match the upper bound.

Acknowledgments We gratefully acknowledge the useful reviews by the anonymous reviewers on an earlier version of this manuscript.

References

1. Bartal, Y., Koutsoupias, E.: On the competitive ratio of the work function algorithm for the $k$-server problem. Theor. Comput. Sci. 324, 337–345 (2004)
2. Bein, W., Chrobak, M., Larmore, L.L.: The 3-server problem in the plane. Theor. Comput. Sci. 287, 387–391 (2002)
3. Bein, W., Iwama, K., Kawahara, J., Larmore, L.L., Oravec, J.A.: A randomized algorithm for two servers in cross polytope spaces. Theor. Comput. Sci. 412(7), 563–572 (2011)
4. Bianchi, M.P., Böckenhauer, H., Hromkovič, J., Keller, L.: Online coloring of bipartite graphs with and without advice. Algorithmica 70(1), 92–111 (2014)
5. Bianchi, M.P., Böckenhauer, H., Hromkovič, J., Krug, S., Steffen, B.: On the advice complexity of the online $L(2, 1)$-coloring problem on paths and cycles. Theor. Comput. Sci. 554, 22–39 (2014)
6. Böckenhauer, H., Komm, D., Královič, R., Rosmanith, P.: The online knapsack problem: Advice and randomization. Theor. Comput. Sci. 527, 61–72 (2014)
7. Böckenhauer, H.J., Hromkovic, J., Komm, D., Krug, S., Smula, J., Sprock, A.: The string guessing problem as a method to prove lower bounds on the advice complexity. Theor. Comput. Sci. 554, 95–108 (2014)
8. Böckenhauer, H.J., Komm, D., Královič, R., Královič, R.: On the advice complexity of the $k$-server problem. In: Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP), LNCS, vol. 6755, pp. 207–218 (2011)
9. Böckenhauer, H.J., Komm, D., Královič, R., Královič, R., Mömke, T.: On the advice complexity of online problems. In: Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC), LNCS, vol. 5878, pp. 331–340 (2009)
10. Bodlaender, H.L.: A tourist guide through treewidth. Acta Cybernet. 11, 1–23 (1993)
11. Boyar, J., Kamali, S., Larsen, K.S., López-Ortiz, A.: On the list update problem with advice. In: Proceedings of the 8th International Conference on Language and Automata Theory (LATA) (2014)
12. Boyar, J., Kamali, S., Larsen, K.S., López-Ortiz, A.: Online bin packing with advice. In: Proceedings of the 31st Symposium on Theoretical Aspects of Computer Science (STACS) (2014)
13. Chrobak, M., Karloff, H., Payne, T.H., Vishwanathan, S.: New results on server problems. SIAM J. Discret. Math. 4, 172–181 (1991)
14. Chrobak, M., Larmore, L.L.: An optimal online algorithm for $k$-servers on trees. SIAM J. Comput. 20, 144–148 (1991)
15. Coppersmith, D., Doyle, P.G., Raghavan, P., Snir, M.: Random walks on weighted graphs and applications to on-line algorithms. J. ACM 40, 421–453 (1993)
16. Dobrev, S., Královič, R., Markou, E.: Online graph exploration with advice. In: Proceedings of 19th International Colloquium on Structural Information and Communication Complexity (SIROCCO), LNCS, vol. 7355, pp. 267–278 (2012)
17. Dobrev, S., Královič, R., Pardubská, D.: How much information about the future is needed? In: Proceedings of the 34th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM), LNCS, vol. 4910, pp. 247–258 (2008)
18. Dragan, F.F., Yan, C., Corneil, D.G.: Collective tree spanners and routing in at-free related graphs. J. Graph Algorithm. Appl. 10(2), 97–122 (2006)
19. Dragan, F.F., Yan, C., Lomonosov, I.: Collective tree spanners of graphs. SIAM J. Discret. Math. 20(1), 241–260 (2006)
20. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. Theor. Comput. Sci. 412(24), 2642–2656 (2011)
21. Farzan, A., Kamali, S.: Compact navigation and distance oracles for graphs with small treewidth. In: Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP), LNCS, vol. 6755, pp. 268–280 (2011)
22. Forišek, M., Keller, L., Steinová, M.: Advice complexity of online coloring for paths. In: Proceedings of the 6th International Conference on Language and Automata Theory (LATA), LNCS, vol. 7183, pp. 228–239 (2012)
23. Gupta, A., Kumar, A., Rastogi, R.: Traveling with a pez dispenser (or, routing issues in mpls). SIAM J. Comput. 34(2), 453–474 (2004)
24. Gupta, S., Kamali, S., López-Ortiz, A.: On advice complexity of the k-server problem under sparse metrics. In: Proceedings of the 20th International Colloquium on Structural Information and Communication Complexity (SIROCCO), LNCS (2013)
25. Halin, R.: S-functions for graphs. J. Geom. 8(1-2), 171–186 (1976)
26. Hromkovič, J., Královíc, R., Královíc, R.: Information complexity of online problems. In: Proceedings of the 35th International Symposium on Mathematical Foundations of Computer Science (MFCS), LNCS, vol. 6281, pp. 24–36 (2010)
27. Karlin, A., Manasse, M., McGeoch, L., Owicki, S.: Randomized competitive algorithms for non-uniform problems. In: Proceedings of the 1st ACM-SIAM Symposium on Discrete algorithms (SODA), pp. 301–309 (1990)
28. Karlin, A., Manasse, M., McGeoch, L., Owicki, S.: Competitive randomized algorithms for nonuniform problems. Algorithmica 11, 542–571 (1994)
29. Komm, D., Královíc, R.: Advice complexity and barely random algorithms. In: Proceedings of the 37th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM), LNCS, vol. 6543, pp. 332–343 (2011)
30. Komm, D., Královíc, R., Mömke, T.: On the advice complexity of the set cover problem. In: Proceedings of the 7th International Computer Science Symposium in Russia (CSR), LNCS, vol. 7353, pp. 241–252 (2012)
31. Koutsoupias, E., Papadimitriou, C.: On the k-server conjecture. J. ACM 42, 971–983 (1995)
32. Koutsoupias, E., Papadimitriou, C.: The 2-evader problem. Inf. Process. Lett. 57, 249–252 (1996)
33. Manasse, M.S., McGeoch, L., Sleator, D.D.: Competitive algorithms for on-line problems. In: Proceedings of the 20th ACM Symposium on Theory of Computing (STOC), pp. 322–333 (1988)
34. Manasse, M.S., McGeoch, L.A., Sleator, D.D.: Competitive algorithms for server problems. J. Algorithm. 11(2), 208–230 (1990)
35. Matoušek, J.: On embedding trees into uniformly convex banach spaces. Israel J. Math. 114, 221–237 (1999)
36. Renault, M.P., Rosén, A.: On online algorithms with advice for the k-server problem. In: Proceedings of the 9th Workshop on Approximation and Online Algorithms (WAOA), LNCS, vol. 7164, pp. 198–210 (2011)
37. Robertson, N., Seymour, P.D.: Graph minors. iii. planar tree-width. Journal of Combinatorial Theory. Ser. B 36(1), 49–64 (1984)
38. Seibert, S., Sprock, A., Unger, W.: Advice complexity of the online coloring problem. In: Proceedings of the 8th International Conference on Algorithms and Complexity (CIAC), LNCS, vol. 7878, pp. 345–357 (2013)
39. Yan, C., Xiang, Y., Dragan, F.F.: Compact and low delay routing labeling scheme for unit disk graphs. Comput. Geom. 45(7), 305–325 (2012)