Online Algorithms with Advice

Marc Renault

Supervisor: Adi Rosén, Algorithms and Complexity LRI

August 21, 2010

This report is written in English as the maternal language of Marc Renault is English and the supervisor, Adi Rosén, is not fluent enough in French.

Le contexte général

In online computation, the data is revealed one element at a time. Algorithms dealing with online problems must make decisions upon receipt of each element without knowing the entire sequences of elements. Online algorithms have many real world applications such as paging, task scheduling, routing and investing algorithms. One classical abstract online problem is termed the k-server problem. This problem is defined by a metric space, $\mathcal{M}$, k mobile servers and a request sequence σ. Each request is to a node of $\mathcal{M}$ and a server must be moved to the requested node before the next request is revealed to the algorithm. The goal is to minimize the distance traveled by the k servers over σ.

Competitive analysis is used to gauge the effectiveness of an algorithm against the optimal manner in solving the problem. Although competitive analysis is a meaningful method for performing a worst-case analysis, it has been criticized as being too pessimistic and that it does not convey enough information about the performance of the algorithm analyzed. For example, paging algorithms such as LRU (Least Recently Used), FIFO (First In First Our), FWF (Flush When Full) and others have all been shown to be k-competitive, where k is size of the cache, yet in practice, their performance varies dramatically between the different algorithms [3, 17].

In light of this criticism, different models have been proposed to more realistically analyze online algorithms. Recently, models of online algorithms with advice have been published [7, 6]. The model presented is that with each data element an unrestricted oracle sends along a predetermined and predefined number of bits to aid the algorithm in its computation. In [7], the authors showed that there is a $k^{O(\frac{1}{b})}$ competitive algorithm for the k-server problem where k is the number of servers and b is the bits of advice. In [6] and [2], the authors explore the amount of advice required for deterministic and random paging algorithms, scheduling algorithms and routing algorithms to be 1-competitive and the change in the competitive ratio as a function of the amount of advice.

Le problème étudié

The focus of the Master's project was to continue the work of [7] by exploring special cases of the k-server problem with advice. The motivation was to gain insight to help in determining a lower bound for the k-server problem with advice on general metric spaces.

La contribution proposée

Over the course of the Master's project:

- we formalized the proof for a 1-competitive deterministic algorithm with 1 bit of advice for the k-server problem where $N = k + 1$.
- we extended the deterministic paging algorithm with 1 bit of advice presented in [6] to the weighted paging problem with advice and show that it is 1-competitive. Then, we present a method for reducing the k-server problem with advice on the star to the weighted paging problem with advice and show that the 1-competitive solution for the

weighted paging problem with advice remains 1-competitive for the k-server problem with advice on the star.

- we developed an optimal algorithm with 1 bit of advice for the k-server problem on the line and the cycle. This algorithm utilizes a minimum weighted matching between the servers of the optimum and the algorithm for the advice where the server used by the optimum to serve a request is matched to an adjacent server of the algorithm.
- we extended the algorithm for the k-server on the line to the k-server problem on the spider graph and showed it to be 1-competitive.

Les arguments en faveur de sa validité

As there is no lower bound established for the k-server problem with advice and the best result to date is a $k^{O(\frac{1}{b})}$ competitive deterministic algorithm, the results of this Master's are currently the best results established for the metric spaces we considered.

Le bilan et les perspectives

The next step would be to extend the algorithm presented for the k-server problem on the spider graph to the tree. Following that, we would hope to establish a lower bound for the k-server problem on the tree with advice and, then, a lower bound on the general metric space.

In a more general context, as online algorithms are typically approximation algorithms, it would be interesting to explore if there is a connection between achievable competitive ratios with advice and approximability results.

1 Introduction

1.1 Online algorithms

Online algorithms are algorithms that receive their input sequentially over time and have to perform their operations after each piece of the input is received before they receive the next piece of the input. Since the online algorithm does not have knowledge about the entire input sequence, an operation performed upon receipt of a piece of the input may prove not to be optimal in the future. The study of online algorithms tends to focus more on the quality of the decisions made by the algorithm against an optimal algorithm rather than the computational complexity of the algorithm.

A classical, simple, and illustrative online problem is called the ski rental problem [9]. Suppose that the cost to buy skis is €200 while renting skis costs €20. The problem is to determine whether or not to buy skis. If we knew how many times we would go skiing the problem would be trivial. However, as the future is uncertain (e.g. snow conditions), is there a way to solve this problem with a guaranteed result? There is an online algorithm that guarantees that we will not pay more than twice what is optimal. The algorithm works as follows: rent skis until the cost of renting is equal to or exceeds the cost to buy skis. In the example, we would rent skis 10 times and then buy them. If we go skiing 10 times or less, we would be optimal. If we go skiing more than 10 times, we would spend €400 while optimal would have been to buy the skis initially for a cost of €200.

Online problems are prevalent in many areas such as financing, scheduling, routing, robot motion, and memory management. For this Master's project we focused on one particular online problem, the k-server problem which is one of the most extensively studied online problems. Although randomization does play an important role in many online algorithms, our study focused on deterministic algorithms.

1.2 Competitive analysis

In the study of online algorithms, the most common method used to compare an online algorithm's performance to that of the optimal performance is called competitive analysis. With competitive analysis, we are interested in the worst case, over all valid finite request sequences, of the comparison of the performance of an online algorithm against the optimal performance.

For the ski rental algorithm, we would say that it has a competitive ratio of 2. Alternatively, we could say that our ski rental algorithm is 2-competitive or more specifically it is strictly 2-competitive as the competitive ratio of our algorithm is exactly 2 (i.e. there are no additive constants).

Different authors [9, 3, 11] have raised concerns over competitive analysis. Specifically, it has been noted that competitive analysis is too pessimistic and does not convey enough information about the performance of the analyzed algorithm. It has been shown that all marking paging algorithms are k-competitive [16] which is optimally competitive for the paging problem [15]. However, when the performance of FWF (Flush When Full), LRU (Least Recently Used), and FIFO (First In First Out) are compared empirically it can be shown that LRU performs better than FIFO which performs better than FWF [17].

1.3 k-server problem

The k-server problem was first presented in [14]. It consists of a metric space $\mathcal{M}$, k mobile servers and a request sequence σ. 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. In the most general k-server problem there are no restrictions placed on $\mathcal{M}$, the k servers or the locations of the requests. The goal is to minimize the distance traveled by the k servers over σ.

The k-server problem is one of the most studied online problems. In [14], the authors showed that the lower bound for the competitive ratio for a deterministic k-server algorithm is k. In addition, they conjectured that there exists a k-competitive deterministic algorithm for the k-server problem in any metric space. To date, the best results for the k-server problem is the Work Function Algorithm, a deterministic algorithm that is $2k - 1$-competitive [12].

1.3.1 Paging

The paging problem consists of a collection of pages, a cache or fast memory that can hold k pages and a request sequence σ. Each request is for one of the pages. If the requested page is in fast memory, the algorithm does not need to perform any operation. If the requested page is not in fast memory, there is a page fault and the algorithm must bring the requested page into fast memory. If the cache is full, the algorithm must evict a page to make room for the requested page. The goal is to reduce the number of page faults.

In [14], the authors show that the paging problem is in fact a special case of the k-server problem where $\mathcal{M}$ is a uniform metric space.

Numerous deterministic algorithms such as FWF, FIFO and LRU have been shown to be k-competitive for the paging problem [10, 15]. It has been proven that for the paging problem, a competitive ratio of k is competitively optimal for deterministic algorithms [15].

A variation of the paging problem is the weighted paging problem. The motivation for this variation comes, for example, from distributed computing where the cost to retrieve a page may vary due to the different communication costs with the different systems. The cost of page faults varies depending on the page that faults.

In [4, 3], the authors show that the weighted paging problem can be expressed as a k-server problem on a star graph and that there is a k-competitive algorithm for it.

1.3.2 Special metric spaces

Many of the special cases of the k-server problem are restrictions applied to the metric space $\mathcal{M}$. The following is a listing of some such special cases and the known results for deterministic algorithms:

$N = k + 1$ The restriction in this case is that N, the number of nodes of $\mathcal{M}$, is 1 more than k. In [14], the authors present a k-competitive deterministic algorithm.

k-server on the line The restriction in this case is that $\mathcal{M}$ can be embedded on the real line. In [4], the authors showed that for the k-server problem on the line there is a k-competitive deterministic algorithm.

k-server on the cycle The restriction in this case is that $\mathcal{M}$ can be embedded on a circle. The WFA with a competitive ratio of $2k - 1$ is the best result for the k-server problem on the cycle. Prior to that, in [8], the authors showed that there is a $O(k^3)$-competitive deterministic algorithm for the k-server problem on the cycle.

k-server on the tree The restriction in this case is that $\mathcal{M}$ can be embedded on the real tree. In [5], the authors showed that the k-competitive algorithm of [4] for the line could be generalized to the tree and that it remains k-competitive.

1.4 Online algorithms with advice

The authors of [7] and [6] present similar methods of strengthening an online algorithm by providing it with some quantifiable information about the future. The interest in online algorithms with advice is to provide insight into the impact information can have on the effectiveness of an algorithm. In other words, we are exploring the impact different amounts of future information have on the competitive ratio. This advice is provided by an oracle with no computational restrictions. For this project, we are continuing the work done by [7] by continuing the exploration of advice on the k-server problem.

Although the study of online algorithms with advice has been mainly theoretical, there are practical applications particularly in the areas of distributed computing and wireless computing where the cost of communication is not negligible. In those instances, it can be useful to know a guaranteed performance of an algorithm of a remote device as a function of the amount of information communicated to the device.

1.5 k-server problem with advice

In [7], the authors present a deterministic algorithm for the k-server problem with advice with a competitive ratio of $k^{O(\frac{1}{b})}$ where $\Theta(1) \leq b \leq \log(k)$.

In [6], the authors present a 1-competitive deterministic algorithm for the paging problem with 1 bit of advice.

1.6 Preliminaries

From [1, 3], online algorithms can be expressed as a request-answer system 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 $\text{cost}_n : R^n \times A_1 \times A_2 \times \cdots \times A_n \to \mathbb{R}^+ \cup \{\infty\}$ for $n = 1, 2, \ldots$. In the context of a request-answer system, it is possible to define a deterministic online algorithm as a sequence of functions, $g_i : R^i \to A_i$ for $i = 1, 2, \ldots$.

As online problems are minimization or maximization problems, we are trying to minimize or maximize some cost over a request sequence, $\sigma = r_1, \ldots, r_n$. Let $\text{ALG}(\sigma)$ and $\text{OPT}(\sigma)$ be the cost to an online algorithm and the optimum over σ respectively. Returning to the definition of [1, 3], we can more formally define $\text{ALG}(\sigma) = \text{cost}_n(\sigma, \text{ALG}[\sigma])$ where $\text{ALG}[\sigma] = < a_1, \ldots, a_n > \in A_1 \times \cdots \times A_n$ and $a_j = g_j(r_1, \ldots, r_j)$ for $j = 1, \ldots, n$.

With a risk of a slight abuse of notation, we will note the cost of a subsequence of σ as $\text{ALG}(r_i, \ldots, r_j) = \text{cost}_{i,\ldots,j}(r_i, \ldots, r_j, \text{ALG}[r_i, \ldots, r_j])$ where $r_i, \ldots, r_j$ is a subsequence of σ such that all the requests between $r_i \in \sigma$ and $r_j \in \sigma$ are contained in $r_i, \ldots, r_j$. Also, $\text{ALG}[r_i, \ldots, r_j] =< a_i, \ldots, a_j > \in A_i \times \cdots \times A_j$ where $< a_i, \ldots, a_j >$ is a subsequence of $< a_1, \ldots, a_n >$. and the initial configuration is the configuration of ALG at $r_{i-1} \in \sigma$ Further, $\text{ALG}(r_i) = \text{ALG}(r_1, \ldots, r_i) - \text{ALG}(r_1, \ldots, r_{i-1})$ represents the cost to ALG to process a request r_i.

If it is possible to break the processing of a request into operations, we will denote the costs to the separate operations for a request by using primes. For example, if there are two operations performed by ALG per request, then $\text{ALG}(r_i) = \text{ALG}(r_i)' + \text{ALG}(r_i)''$ where $\text{ALG}(r_i)'$ is the cost for the first operation and $\text{ALG}(r_i)''$ is the cost for the second operation.

For a minimization problem, we say that an algorithm is c-competitive or has a competitive ratio of c, if, for every finite σ, $\text{ALG}(\sigma) \leq c\text{OPT}(\sigma) + \zeta$ where ζ is a constant that is not dependent on the length of σ. When $\zeta = 0$, we say that the algorithm is strictly c-competitive.

The k-server problem consists of a metric space, $\mathcal{M} = (M, \text{dist})$ where M is a set of nodes and $\text{dist} : M \times M \to \mathbb{R}^+$ is a metric on M. A metric on M implies that $\text{dist}(x, y)$ is symmetric ($\text{dist}(x, y) = \text{dist}(y, x)$ for all $x, y \in M$), has triangle inequality ($\text{dist}(x, z) \leq \text{dist}(x, y) + \text{dist}(y, z)$ for all $x, y, z \in M$) and $\text{dist}(x, y) = 0$ if and only if $x = y$. The k-server also consists of k mobile servers and a request sequence σ. It should be noted that $|M| = N > k$ and the length of σ is n.

Lazy k-server algorithms are k-server algorithms that move, at most, one server to serve a request and will only do so if the request is not covered by server. By the triangle inequality property of the metric space, it can be shown that any k-server algorithm can be converted to a lazy algorithm without increasing the cost. In [14], the authors prove the following lemma:

Lemma 1.1. *For any algorithm B, there is a modified algorithm B' that is lazy, does not cost more, and is online if B is.*

The paging problem consists of a collection of N pages, P, a cache or fast memory that can hold k pages and a request sequence σ.

For the weighted paging problem, there is a function wt : $P \to \mathbb{R}^+$. Let p be the page requested at r_i. If there is a page fault for r_i, the cost to the algorithm is wt(p).

Formally, we will quantify and define online algorithms with advice in the same manner as [7]. The advice will be an additional b bits of information provided with each request. Let U be a finite set known as the advice space. The advice space is considered to have a size of 2^b for some $b \geq 0$ which could allow for an inaccuracy of the results by, at most, a factor of 2. The online algorithm has access to U via a query $u_i : R^* \to U$. So, a deterministic online algorithm with advice can be expressed as a sequence of pairs (g_i, u_i) where $g_i : R^i \times U^i \to A_i$. The cost of a deterministic algorithm with advice is $ALG[\sigma] = \text{cost}_n(\sigma, \text{ALG}[\sigma])$ 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 $j = 1, \ldots, n$. This model allows for the expression of everything from classical online algorithms to optimal algorithms.

1.7 Outline

In Section 2, we present a 1-competitive algorithm for the k-server problem with advice where $N = k+1$. In Section 3, we present a 1-competitive algorithm for the weighted paging problem and the k-server problem on the star. In Section 4, we present an optimal algorithm for the k-server problem on the line and the cycle. In Section 5, we extend the algorithm used for the k-server problem on line to the k-server problem on the spider graph, a graph with one node of degree greater than 2. Finally, we end with a brief discussion in Section 6.

2 k-server problem with advice where $N = k + 1$

The k-server problem where $N = k+1$ is a special case where the number of nodes, N, of the metric space is 1 more than the number of servers. The node that does not contain a server is called the hole.

For example, BALANCE, an algorithm presented in [13], is shown to be a k-competitive for the k-server problem without advice where $N = k+1$. We present an algorithm, NEAREST, with 1 bit of advice that is 1-competitive for the k-server problem with advice where $N = k+1$. Although the concept of the algorithm and the results were known before beginning the Master's project, we formalized the analysis.

2.1 1-competitive algorithm with 1 bit of advice

The algorithm will have two stages. The first stage is used to match the configurations of ALG and OPT such that the hole of ALG and OPT match and that ALG has a subset of servers that correspond to servers of OPT from which OPT will use to serve any subsequent requests. Let V be the set of servers that have not served a request. For the initial stage, our algorithm will use the nearest server of V.

In the second stage, the algorithm makes use of marked servers. A marked server is a server that has responded to a request and the bit of advice from the last request served by the server was a 1. Let W be the set of marked servers in ALG and let Y be the set of unmarked servers. The algorithm will serve each request of this stage with the closest server of Y. At the start of the request sequence, all the servers are considered unmarked.

More formally, the algorithm can be defined as follows:

Algorithm NEAREST: If V is not empty, let t be the nearest server of V else let t be the nearest server of Y. Serve the request with t and update the marking of t based on the advice.

2.1.1 Advice

A single bit of advice will be sent along with each request. The advice for r_i will be:

1, if the location of r_i will remain covered by a server in OPT until the next request at this same location or the end of the request sequence.

0, otherwise.

2.2 Analysis

For the analysis that follows, we will, without loss of generality, assume that OPT is lazy.

Claim 2.1. *Let r_l be a request to the hole of ALG after there have been $k+1$ requests to the hole of ALG. At r_l, the hole of ALG and the hole of OPT are at the same node.*

Proof. Given that NEAREST is a lazy algorithm and that, for the first $k+1$ requests to the hole of ALG, NEAREST will use a server of V or Y, the hole of ALG changes after serving each request and does not repeat until all the servers of NEAREST have served a request. Therefore, after $k+1$ requests to the hole of ALG, every node has received a request.

Assume to the contrary that the node requested by r_l is covered by a server in OPT but not in ALG. This implies that at the previous request to this node the 1 bit of advice was a 1. If the advice was a 1, ALG would not have moved the server, so there would not be a hole in ALG at this node which is a contradiction. □

Claim 2.2. *Let r_l be a request to a node not covered by a server in W after there have been $k+1$ requests to the hole of ALG. For r_l, OPT will always use a server in the same position as a server of ALG in Y.*

Proof. Let t be the server of ALG at the same position as the server that OPT uses for r_l. Claim 2.1 shows that such a server exists. Let r_m be the previous request served by t. Assume to the contrary that $t \in M$. This implies that the advice at r_m should have been 0, but since $t \in M$ the advice was a 1 which is a contradiction. □

Theorem 2.3. *NEAREST is 1-competitive.*

Proof. Let r_q be the $k+1$th request to the hole of ALG. From r_1 to r_q, the algorithm either has a server at the position of the request or uses the nearest server of V or Y to serve the request. The cost for this can be bounded by $(k+1)d$ where d is the diameter of the metric space as ALG only moves a server $k+1$ times over the first q requests.

By Claim 2.1, after r_q, any request to the hole of ALG is to the hole of OPT and, from Claim 2.2, OPT will only use a server that is at the position of a server of ALG in Y. Since NEAREST always uses the closest server of Y,

$$\mathrm{ALG}(r_q, \ldots, r_n) \leq \mathrm{OPT}(r_q, \ldots, r_n)$$

and over the entire request sequence,

$$\mathrm{ALG}(\sigma) \leq \mathrm{OPT}(\sigma) + (k+1)d.$$

□

3 k-server problem with advice on the star

The star graph is a graph where one vertex has a degree greater than 1. The rest of the vertices are leaves. The vertex with a degree greater than 1 is known as the centre. As shown in [4] and [3], the k-server problem on the star can simulate the weighted paging problem.

In this section, we present a 1-competitive algorithm called WEIGHTED-MARK for the weighted paging problem with advice and show that it can be used to construct an algorithm for the k-server problem with advice on the star. This algorithm is an extension of the algorithm for paging with advice found in [6] to the weighted paging problem with advice.

3.1 1-competitive algorithm with 1 bit of advice for the weighted paging problem

For each page request, r_i, 1 bit of advice is given which indicates whether the requested page is marked or not. At any given instance there are two sets of pages in fast memory, marked pages and unmarked pages. At the start of the algorithm all the pages in fast memory are unmarked. The algorithm is defined as follows.

Algorithm WEIGHTED-MARK: If the requested page is not in fast memory, then evict an arbitrary unmarked page and bring the requested page into fast memory. For every request, if the 1 bit of advice is a 1, mark the requested page otherwise unmark the requested page.

3.1.1 Advice

1 bit of advice will be given with each request. The advice will be as follows:

1, if the requested page will be in OPT's fast memory the next time it is requested or at the end of the request sequence.

0, otherwise.

3.1.2 Analysis

Claim 3.1. *For every request after the first k distinct page requests, whenever ALG has a page fault so does OPT.*

Proof. Assume to the contrary that, after the first k distinct page requests, ALG has a page fault when OPT does not. Let r_f be that request. Let p be the requested page at r_f. As we have had k distinct requests, for OPT to have p in its fast memory it must have been requested earlier in the request sequence. Let r_p be the last request, before r_f, where p was requested. At some request between r_p and r_f, ALG evicts p. This implies that the advice given at request r_p was a 0 which is a contradiction. □

Theorem 3.2. *WEIGHTED-MARK is 1-competitive.*

Proof. For the first k distinct page requests, the cost to WEIGHTED-MARK can be bounded by kw where w is weight of the heaviest page.

After the first k distinct page requests, the cost incurred by ALG and OPT are the same as shown by Claim 3.1. Therefore, $\text{ALG}(\sigma) \leq \text{OPT}(\sigma) + kw$. □

3.2 Reducing the k-server problem on the star to the weighted paging problem

In this section, we present a method of reducing the k-server problem with advice on the star to the weighted paging problem with advice and, in the reduction, we keep the same number of bits of advice. This reduction enables us to show that the k-server problem on the star is 1-competitive with 1 bit of advice. The reduction is based on the reduction from the weighted paging problem to the k-server problem as informally described in [3].

Given an instance of the k-server problem on the star graph, STAR, we will construct an instance of the weighted paging problem, PAGE.

3.2.1 Configuration

STAR consists of k mobile servers and a metric space $\mathcal{M} = (M, \text{dist})$ where M is the set of nodes, $|M| = N$, $\text{dist}(x, y)$ is the distance between x and y in M and there exists a $c \in M$ such that, for all $x, y \in M$, $\text{dist}(x, y) = \text{dist}(x, c) + \text{dist}(y, c)$ where $x \neq y$. c will denote the centre of the star.

We will now build an instance of the weighted paging problem PAGE. For each $m \in M$, there will be a page, p_m, in PAGE such that $\text{wt}(p_m) = 2\text{dist}(m, c)$ where $\text{wt}(p_m)$ is the weight of p_m. The size of the fast memory of PAGE will be k.

For each $m \in M$ such that there is a server at m in the initial configuration of STAR, p_m will be in the fast memory of PAGE for its initial configuration.

3.2.2 Requests

Let σ_{STAR} be the request sequence for STAR. We build the request sequence of PAGE, σ_{PAGE}, as follows. The length of σ_{STAR} and the length of σ_{PAGE} are equal and, for all $r_i \in \sigma_{\text{PAGE}}$, $r_i = \{p_m : m \text{ is the node requested by } r_i \in \sigma_{\text{STAR}}\}$

3.3 Responses from PAGE to STAR

In this section we will present an algorithm to converts the responses of a complete and correct paging algorithm, PAGE-ALG, to σ_{PAGE} on PAGE to a series of response for σ_{STAR} on STAR and show the costs differ by a constant not dependant on σ.

Algorithm PAGE-STAR: Whenever PAGE-ALG evicts a page, p_e, if there is a server at the node e in STAR, then move the server from e to c else do nothing.

Whenever PAGE-ALG moves a page, p_m, into fast memory, if there is a server at the centre in STAR, then move a server from c to m else do nothing.

We will show that either case where PAGE-STAR does nothing while PAGE-ALG performs some action cannot happen.

3.3.1 Analysis

Claim 3.3. *Let p_m be a page in PAGE-ALG and let m be the node in PAGE-STAR that p_m represents. If p_m is in fast memory in PAGE-ALG, then there is a server at node m in PAGE-STAR.*

Proof. This is true for the initial configuration of both PAGE-ALG and PAGE-STAR. Assume that it is true from operation 1 to $i-1$ of PAGE-ALG. At operation i, PAGE-ALG will do one of two operations. They are:

1. PAGE-ALG evicts a page. Let p_e be the page evicted from fast memory and let e be the node it represents in PAGE-STAR. As the claim was true for the previous action, there is a server on node e. Let s be the server of PAGE-STAR at e. PAGE-STAR will move s from e to c.

2. PAGE-ALG brings a page into fast memory. Let p_m be the page to be brought into fast memory and let m be the node it represents in PAGE-STAR. In order for PAGE-ALG to bring a page into fast memory, there must be at least one vacant memory slot. As the claim was true for the previous action and given that there is a vacant memory slot, there must be a server at the centre in PAGE-STAR. Therefore, PAGE-STAR will move a server from c to m.

Therefore, after performing operation i, the claim remains true. □

Lemma 3.4. *The response algorithm, PAGE-STAR, is complete and correct for STAR.*

Proof. Let p_m be the page requested at r_i of σ_{PAGE}. By definition of the request sequences, m will be the page requested at r_i of σ_{STAR}. Upon serving r_i, p_m will be in the fast memory of PAGE-ALG. From Claim 3.3, we know that if p_m is in fast memory, then, in PAGE-STAR, there is a server at m upon serving r_i. Therefore, PAGE-STAR is complete and correct for STAR. □

Lemma 3.5. *PAGE-STAR*$(\sigma_{STAR}) \leq$ *PAGE-ALG*$(\sigma_{PAGE}) + \frac{1}{2}kw$ *where PAGE-ALG is any algorithm used for PAGE and* w *is the page with the largest weight.*

Proof. Let A be the set of pages initially in fast memory that will be evicted before the end of σ and are not part of the final configuration in PAGE-ALG, let B be the multiset of pages that enter and exit fast memory over σ_{PAGE}, and the pages that are evicted from the initial configuration and are part of the final configuration in PAGE-ALG. Let C be the set of pages in fast memory at the end of the request sequence that are not part of the initial configuration in PAGE-ALG. Let $\text{wt}(p)$ be the weight of page p.

We can express $\text{PAGE-ALG}(\sigma_{\text{PAGE}}) = \sum_{b\in B} \text{wt}(b) + \sum_{c\in C} \text{wt}(c)$. Let $\beta = \frac{1}{2}\sum_{a\in A} \text{wt}(a)$. As $|A| \leq k$, then $\beta \leq \frac{1}{2}kw$ where w is the page with the largest weight. By the construction of the weighted paging instance, w is equal to twice the distance of the longest branch of the star graph.

For every $b \in B$, the cost to PAGE-ALG is $\text{wt}(b)$ when it is brought into fast memory and nothing when b is evicted. For every $b \in B$, PAGE-STAR will move a server from the centre to m_b and move a server from m_b to the centre according to Lemma 3.3 where m_b is the node that b represents in PAGE-STAR. The cost to PAGE-STAR will be equivalent to $\text{wt}(b)$ for every $b \in B$ which is the same cost to PAGE-ALG.

Evicting a page, p_m, has no cost to PAGE-ALG, but the equivalent action in PAGE-STAR is to move a server to the centre which costs $\frac{1}{2}\text{wt}(p_m)$. So, for the pages in the initial configuration that are evicted, the cost to PAGE-STAR is more than the cost to PAGE-ALG by β.

For every $c \in C$, PAGE-ALG pays $\text{wt}(c)$ when c is brought into fast memory. As the page is never evicted, PAGE-STAR mimics bringing c into fast memory by moving a server from

the centre to m_c. The cost to PAGE-STAR is $\frac{1}{2}\text{wt}(c)$. Let $\gamma = \frac{1}{2}\sum_{c \in C} \text{wt}(c)$. As $|C| \leq k$, then $\gamma \leq kw$.

Therefore, PAGE-STAR(σ_{STAR}) $\leq$ PAGE-ALG(σ_{PAGE}) $+ \beta - \gamma$. As γ is positive and $\beta \leq \frac{1}{2}kw$, PAGE-STAR(σ_{STAR}) $\leq$ PAGE-ALG(σ_{PAGE}) $+ \frac{1}{2}kw$. □

3.4 Responses from STAR to PAGE

In this section, we will present an algorithm that converts the responses of a k-server algorithm, STAR-ALG, to σ_{STAR} on STAR to a series of response for σ_{PAGE} on PAGE and show the costs differ by a constant not dependant on σ. The reduction from an instance of the weighted paging problem, PAGE, to an instance of the k-server problem on the star graph, STAR, is the inverse of the reduction from STAR to PAGE in Section 3.2. It is informally described in [3]. Without loss of generality, we will assume that the responses by STAR are lazy.

Algorithm STAR-PAGE: Whenever STAR-ALG moves a server from node e to the centre, if p_e is in fast memory, evict page p_e otherwise do nothing.

Whenever STAR-ALG moves a server from the centre to node m, if p_m is not in fast memory, bring p_m into fast memory otherwise do nothing.

We will show that in the case where STAR-PAGE does nothing while STAR-ALG moves a server cannot happen.

3.4.1 Analysis

Claim 3.6. *Let p_m be a page in STAR-PAGE and let m be the node in STAR-ALG that p_m represents. If there is a server at node m in STAR-ALG, then p_m is in fast memory in STAR-PAGE.*

Proof. This is true for the initial configuration of both STAR-PAGE and STAR-ALG. Assume that it is true from operation 1 to $i-1$ of STAR-ALG. At operation i, STAR-ALG will do one of two operations. They are:

1. STAR-ALG moves a server from node e to the centre. As the claim was true for the previous action, p_e is in fast memory where p_e is the page represented by e in STAR-PAGE. So, STAR-PAGE will evict p_e
2. STAR-ALG moves a server from the centre to m. As the claim was true for the previous action, p_m is not in fast memory where p_m is the page represented by m in STAR-PAGE. So, STAR-PAGE will bring p_m into fast memory.

Therefore, after performing operation i, the claim remains true. □

Lemma 3.7. *The response algorithm, STAR-PAGE, is complete and correct for PAGE.*

Proof. Let m be the node requested at r_i of σ_{STAR}. By definition of the request sequences, p_m will be the page requested at r_i of σ_{PAGE}. Upon serving r_i, there will be a server at m in STAR-ALG. From Claim 3.6, we know that if there is a server at m, then p_m will be in fast memory in STAR-PAGE upon serving r_i. Therefore, STAR-PAGE is complete and correct for STAR. □

Lemma 3.8. *STAR-PAGE(σ_{PAGE}) $\leq$ STAR-ALG(σ_{STAR}) + 2kl where STAR-ALG is any algorithm used for STAR and l is length of the longest branch.*

Proof. For this analysis, we will assume, without the loss of generality, that there is an additional branch with a length of 0 used to represent any requests to the centre of the star.

Let A be the set of nodes covered at the initial configuration but not in the final configuration of STAR-ALG, let B be the multiset of nodes that are covered and uncovered over σ_{STAR}, and the nodes that are covered in both the initial configuration and the final configuration of STAR-ALG. Let C be the nodes that are covered in the final configuration that are not part of the initial configuration of STAR-ALG. Let $\text{dist}(x, y)$ be the distance between the positions of x and y on the star and let c be the centre of the star.

We can express $\text{STAR-ALG}(\sigma_{\text{STAR}}) = \sum_{a\in A} \text{dist}(a, c)+2\sum_{b\in B} \text{dist}(b, c)+\sum_{e\in C} \text{dist}(e, c)$. Let $\beta = \sum_{a\in A} \text{dist}(a, c)$. As $|A| \leq k$, then $\beta \leq kl$ where l is the length of the longest branch. By the construction of the STAR instance, $2l$ is equal to the weight of the heaviest page.

For every $b \in B$, the cost to STAR-ALG is $2\text{dist}(b, c)$. The equivalent actions in page would be to bring p_b into fast memory and to evict it. So, the cost to STAR-PAGE will be equivalent to $2\text{dist}(b, c)$ for every $b \in B$.

Moving a server to the centre in STAR-ALG is equivalent to evicting a page in STAR-PAGE which has no cost to STAR-PAGE. So, for the servers in the initial configuration that moved, the cost to STAR-ALG is more than the cost to STAR-PAGE by β.

For every $e \in C$, STAR-ALG pays $\text{dist}(e, c)$ when e is brought into fast memory. STAR-PAGE mimics this by bringing p_e into fast memory. The cost to STAR-PAGE is equivalent to $2\text{dist}(e, c)$. Let $\gamma = 2\sum_{e\in C} \text{dist}(e, c)$. As $|C| \leq k$, then $\gamma \leq 2kl$.

Therefore, $\text{STAR-PAGE}(\sigma_{\text{PAGE}}) \leq \text{STAR-ALG}(\sigma_{\text{STAR}}) - \beta + \gamma$. As β is positive and $\gamma \leq 2kl$, $\text{STAR-PAGE}(\sigma_{\text{PAGE}}) \leq \text{STAR-ALG}(\sigma_{\text{STAR}}) + 2kl$. □

3.5 WEIGHTED-MARK applied to the k-server problem with advice

Theorem 3.9. *PAGE-STAR is 1-competitive for the k-server problem with advice when PAGE-ALG is WEIGHTED-MARK.*

Proof. In Lemma 3.5, we show that

$$\text{PAGE-STAR}(\sigma_{\text{STAR}}) \leq \text{WEIGHTED-MARK}(\sigma_{\text{PAGE}}) + \frac{1}{2}kw \tag{1}$$

and from Theorem 3.2, we know that WEIGHTED-MARK is 1-competitive. Specifically, we show that

$$\text{WEIGHTED-MARK}(\sigma_{\text{PAGE}}) \leq \text{PAGE}_{\text{OPT}}(\sigma_{\text{PAGE}}) + kw \tag{2}$$

where PAGE_{OPT} be the optimum for PAGE and w is the page with the largest weight.

Combining the Equations 1 and 2, gives:

$$\begin{aligned}\text{PAGE-STAR}(\sigma_{\text{STAR}}) &\leq \text{WEIGHTED-MARK}(\sigma_{\text{PAGE}}) + \frac{1}{2}kw \\ &\leq \text{PAGE}_{\text{OPT}}(\sigma_{\text{PAGE}}) + \frac{3}{2}kw. \end{aligned}\tag{3}$$

Let STAR_{OPT} be the optimal algorithm for σ_{STAR} and let STAR-PAGE be the response for an instance of PAGE from the algorithm STAR-PAGE based on STAR_{OPT} responses.

From Lemma 3.8, we get STAR-PAGE(σ_{PAGE}) $\leq$ STAR$_{\text{OPT}}$(σ_{STAR}) + $2kl$ where l is length of the longest branch. Since PAGE$_{\text{OPT}}$(σ_{PAGE}) $\leq$ STAR-PAGE(σ_{PAGE}),

$$\text{PAGE}_{\text{OPT}}(\sigma_{\text{PAGE}}) \leq \text{STAR}_{\text{OPT}}(\sigma_{\text{STAR}}) + 2kl \tag{4}$$

Applying Equations 4 to Equation 3 and the fact that $w = 2l$, we get:

$$\text{PAGE-STAR}(\sigma_{\text{STAR}}) \leq \text{STAR}_{\text{OPT}}(\sigma_{\text{STAR}}) + 5kl.$$

□

4 k-server problem with advice on the line

There is an algorithm, DOUBLE-COVERAGE, for the k-server problem on the line that is k-competitive [4]. For every request, r_i, the algorithm moves both adjacent servers towards r_i until one of the servers reaches the request. Inspired by this algorithm without advice, we present an optimal algorithm, COMPLIANT, with 1 bit of advice that serves each request with an adjacent server.

4.1 Minimum weighted matching on the line

Our algorithm for the line relies on a minimum weighted matching between the servers of OPT and ALG such that the server used by OPT for a request is matched to an adjacent server of ALG. We will show that there always exists such a minimum weighted matching.

Lemma 4.1. *Let A and B be two sets of points of equal size on the line. For any $a \in A$ there is a minimum weighted matching between the points of A and the points of B such that a is matched to a point $b \in B$ that is adjacent to a on the line.*

Proof sketch. An exhaustive proof of the various configurations of 4 points on the line where a matched to $e \in B$ such that e in not adjacent to a and b is matched to $c \in A$. See Appendix A for the full proof.

□

4.2 Potential Function

Our proof that COMPLIANT is optimal makes use of positive potential functions and the following lemma.

Lemma 4.2. *Let Φ'_i and Φ_i be two functions that map the configurations of ALG and OPT to a real number where $\Phi'_i \geq 0$ and $\Phi_i \geq 0$ such that Φ'_i is calculated at the end of an operation, that is not the final operation, performed by ALG or OPT to serve r_i while Φ_i is calculated at the end of the final operation performed by ALG or OPT for r_i. Given the two following conditions for each request, r_i, in σ:*

- $OPT(r_i) \geq \Phi'_i - \Phi_{i-1}$
- $-ALG(r_i) \geq \Phi_i - \Phi'_i$

then $ALG(\sigma) \leq OPT(\sigma) + \Phi_0$

Proof sketch. Add the two conditions and sum over the entire request sequence. See Appendix B for the full proof. □

4.3 Optimal algorithm with 1 bit of advice on the line

The algorithm COMPLIANT always serves a request by an adjacent server. There is at most 2 adjacent servers on the line. The bit of advice will indicate if the server to use is the one to the right or the left of the request. The server indicated by the advice is based on the minimum weighted matching between the servers of OPT and COMPLIANT.

Algorithm COMPLIANT: Serve the request as directed by the advice.

4.3.1 Advice

1 bit of advice will be given with each request. Let s be the server used by OPT to serve request r_i. The advice with r_i will be based on the minimum weighted matching between the servers of OPT and ALG immediately after OPT serves r_i where s is matched to an adjacent server of ALG. Lemma 4.1 shows that such a matching always exists. The one bit of advice will be as follows:

1, if s is matched to the server of ALG immediately to the right of r_i.

0, otherwise.

4.3.2 Analysis

For the analysis of our algorithm, we will break up the process of serving a request, r_i, by OPT and ALG into two operations. They are:

1. OPT serves r_i
2. ALG serves r_i

Lemma 4.3. *Let Φ_i be the weight of the minimum weighted matching between the servers of ALG and OPT after operation 2 and let Φ'_i be the weight of the minimum weighted matching between the servers of ALG and OPT after operation 1. For each request of σ, COMPLIANT meets the two conditions of Lemma 4.2.*

Proof. Without loss of generality, we assume that OPT is lazy. Let s be the server used by OPT to serve r_i and let x be the distance that s is moved. Let q be the server that s is matched to when calculating Φ_{i-1}. Let $\Phi_i^{'*}$ be the weight of the matching after operation 1 using the same matching as was used to calculate Φ_{i-1}.

If s moves closer to q, then $\Phi_i^{'*} - \Phi_{i-1} = -x$ and if s moves away from q, then $\Phi_i^{'*} - \Phi_{i-1} = x$. $\Phi'_i \leq \Phi_i^{'*}$ since Φ'_i is the weight of the minimum weighted matching. Therefore, $\Phi'_i - \Phi_{i-1}$ is at most x which satisfies the first condition of Lemma 4.2.

Let ALG use server t as indicated by the advice and pay y to serve request r_i.

For the matching used to calculate Φ'_i, s is matched to t and t is adjacent to r_i. Let Φ_i^* be the weight of matching after operation 2 using the same matching as was used to calculate Φ'_i. So, $\Phi_i^* - \Phi'_i = -y$. $\Phi_i \leq \Phi_i^*$ since Φ_i is the weight of the minimum weighted matching. Therefore, $\Phi_i - \Phi'_i \leq -y$ which satisfies the second condition of Lemma 4.2. □

Theorem 4.4. *COMPLIANT is optimal.*

Proof. Immediately from Lemmas 4.2 and 4.3, and that $\Phi_0 = 0$ as OPT and ALG start from the same configuration. □

4.4 k-server problem with advice on the cycle

In this section, we show that it is possible to extend the algorithm COMPLIANT to the cycle and that it remains optimal with 1 bit of advice.

The algorithm and the analysis for the cycle are essentially the same for the line and the cycle. However, there are certain terms in the algorithm and the analysis on the line that do not directly translate to the cycle such as "right", "left", "to the left", "to the right", "between" and "move towards". In order to define these concepts with respect to the cycle, we will first choose an arbitrary node to be 12 o'clock. Then, "right" can be defined as the clockwise direction and "left" can be defined as the counterclockwise direction. If a node x is "to the right" of a node y on the line, this would translate to x being closer to y in the clockwise direction starting from y than the counterclockwise direction starting from y and vice versa for "to the left" on the cycle. If a node x is "between" two nodes y and z line, this would translate to x being on the shortest path between y and z on the cycle. If a server is "moving towards" node x on the cycle, this would mean that a server is approaching x along the shortest path between the server's position and x. If a server is "moving away" from node x on the cycle, this would mean that a server is approaching x along the longest path between the server's position and x. Finally, $\text{dist}(x, y)$ is the distance of the shortest path between x and y on the cycle. With these definitions, the algorithm and the analysis translate directly from the line to cycle.

5 k-server with advice on the spider

In this section, we will take our algorithm for the line and extend it to the spider graph. A spider graph is a graph that has 1 vertex with a degree greater than 2. We call this vertex the centre. A line formed from the centre to a leaf will be referred to as an arm.

Our algorithm, COMPLIANT-MOVE, has 2 bits of advice and is 1-competitive.

5.1 Non-lazy optimum

The advice and the competitive analysis will be based on a non-lazy optimum. The definition of the non-lazy optimum will be based on the lazy optimum. Let OPT_l be the lazy optimum and let OPT_{nl} be a non-lazy optimum. Before serving the first request, OPT_{nl} will move to the centre all the servers that will pass through the centre to serve the first request served by that server. OPT_{nl} will only move one server per request and that server is the same server as OPT_l. Let s be the server that OPT_{nl} uses for a request r_i and let r_j be the subsequent request that s serves. If s will pass through the centre to serve r_j in OPT_l, then OPT_{nl} will move it to the centre immediately after serving r_i.

Lemma 5.1. *On the spider graph, the cost to OPT_{nl} is equal to the cost OPT_l over any σ.*

Proof. Let s be the server used by OPT_{nl} and OPT_l to serve request r_i. If s and r_i are on the same arm, then both OPT_{nl} and OPT_l move s to r_i. As s did not pass through the centre, s is at the same position in OPT_{nl} and OPT_l before serving r_i.

If s and r_i are not on the same arm in OPT_l, then the cost for OPT_l to serve r_i is the distance from s to the centre and the distance from centre to r_i. The cost to OPT_{nl} is the distance from centre to r_i. However, as OPT_l is lazy, s is at the position of the previous request it served or at its initial position before serving r_i and, according to the definition

of OPT_{nl}, it would have moved s to the centre either after serving a prior request or before serving the first request. The distance OPT_{nl} moved s prior to r_i is the same as the distance between s and the centre that OPT_l moves s in order to serve r_i. Therefore, over the entire request sequence the cost to OPT_{nl} is equivalent to OPT_l. □

5.2 Potential Function

As with the the line, our proof that COMPLIANT-MOVE is 1-competitive makes use of positive potential functions. The following lemma is slightly modified from Lemma 4.2.

Lemma 5.2. *Let Φ'_i and Φ_i be two functions that map the configurations of ALG and OPT to a real number where $\Phi'_i \geq 0$ and $\Phi_i \geq 0$ such that Φ'_i is calculated at the end of an operation, that is not the final operation, performed by ALG or OPT to serve r_i while Φ_i is calculated at the end of the final operation performed by ALG or OPT for r_i. Given the three following conditions for each request, r_i, in σ where $OPT(r_i) = OPT(r_i)' + OPT(r_i)''$ and $ALG(r_i) = ALG(r_i)' + ALG(r_i)''$:*

- $OPT(r_i)' \geq \Phi'_i - \Phi_{i-1}$
- $-ALG(r_i)' \geq \Phi_i - \Phi'_i$
- $OPT(r_i)'' = ALG(r_i)''$

then $ALG(\sigma) \leq OPT(\sigma) + \Phi_0$

Proof sketch. Add the first two conditions, apply the third condition and sum over the entire request sequence. See Appendix C for the full proof. □

5.3 1-competitive algorithm with 2 bits of advice

For the algorithm, an additional arm will be added to the spider with a single node at a distance of 0 from the centre to the end of the arm. This arm will be used to represent requests that occur on the centre.

There will be two stages to the algorithm. The initial stage will be the first k requests and it is used to match the configuration of ALG to OPT_{nl} after r_k. The cost to ALG for the initial stage can be bounded by $2kd$ where d is the diameter of the spider graph. The second stage makes use of minimum weighted matching between the servers of ALG and OPT_{nl}. The cost to ALG over the second stage is the same as the cost to OPT_{nl}.

5.3.1 Advice and algorithm from $r_1, \ldots, r_k$

From r_1 to r_k, two bits of advice will be sent along with each request. The two bits of advice for ith request, r_i, during the first k requests are:

Bit 1: 1, if, after the kth request, the position of r_i is occupied by a server in OPT_{nl}.
0, otherwise

Bit 2: 1, if, after the kth request, the position of ith server at the start of the request sequence is occupied by a server in OPT_{nl}.
0, otherwise

From r_1 to r_k, COMPLIANT-MOVE will serve each request with the closest server. Immediately after serving r_k, COMPLIANT-MOVE will match the configuration of OPT_{nl}.

The two bits of advice over the first k requests are used to provide the algorithm with the configuration of OPT_{nl}. As OPT_{nl} only moves one server, s, per request, the k servers are either at their initial position, the position of a request or the centre of the spider. The two bits of advice indicate which requests and which initial positions are still occupied immediately after r_k. The difference between k and the positions occupied by servers as per the 2 bits of advice is the number of servers present at the centre of the spider.

5.3.2 Advice and algorithm from $r_{k+1}, \ldots, r_n$

Following the kth request, the advice will be based on a minimum weighted matching between the servers of ALG and ORG_{nl}. Only the arm where the request is located will be considered in the minimum weighted matching. Any servers at the centre will be ignored for the minimum matching on the arm. It is possible that OPT_{nl} would have more servers on the branch than ALG or vice versa. In that case, a number of dummy servers equal to the difference between the servers of OPT_{nl} and ALG will be placed at the position of the centre for the minimum weighted matching on the arm.

Let s be the server used by OPT_{nl} for request r_i. Similarly to the line, the advice will be based on the configuration of OPT_{nl} immediately after serving r_i with s and the configuration of ALG before it serves r_i where s is matched to an adjacent server of ALG.

Bit 1 indicates the direction of the server to use. Bit 2 indicates whether or not s is moved to the centre after serving the request. The two bits of advice for r_i are defined as follows:

Bit 1: 1, if s is matched to the server of ALG adjacent to r_i towards the centre.
0, otherwise.

Bit 2: 1, if, after serving the r_i with s, OPT_{nl} will move s to the centre.
0, otherwise

From r_{k+1} to r_n where n is the length of the request sequence, σ, COMPLIANT-MOVE will serve each request from an adjacent server as directed by the first bit of advice. If the second bit is a 1, COMPLIANT-MOVE will then move the server used for r_i to the centre.

5.3.3 Algorithm: COMPLIANT-MOVE

Algorithm COMPLIANT-MARK(r_i, adv) where r_i is the request at time i and adv is the advice.

---

if $i \leq k$ **then**
  Use the nearest server.
  if $i = k$ **then**
    Match the configuration of the OPT_{nl} based on the 2 bits advice from r_1 to r_k.
  end if
else
  if the first bit of advice $= 0$ **then**

Use t, where t is the adjacent server to r_i in the direction away from the centre.
else
Use t, where t is the adjacent server to r_i in the direction towards the centre.
end if
if the second bit of advice $= 1$ **then**
Move t to the centre
end if
end if

5.3.4 Analysis

For the analysis of the COMPLAINT-MOVE algorithm, we will break up the process of serving a request, r_i, by OPT_{nl} and ALG into three operations. They are:

1. OPT_{nl} serves r_i with s.
2. ALG serves r_i with t.
3. OPT_{nl} moves s and ALG moves t to the centre if required.

As with the algorithm for the line, we will make use of potential functions that use a minimum weighted matching between the servers of OPT_{nl} and the servers of ALG. In the case of the spider graph, we will calculate a minimum weighted matching per arm. The minimum weighted matching is the same as described in Section 5.3.2.

Claim 5.3. *Let $srv_{ALG}(a, r_i)$ and $srv_{OPT_{nl}}(a, r_i)$ be the number of servers that ALG respectively OPT_{nl} has on an arm $a \in \alpha$ after serving request r_i where α is the set of all the arms of the spider graph. For any $a \in \alpha$, $srv_{OPT_{nl}}(a, r_l) \geq srv_{ALG}(a, r_l)$ where $l \geq k$.*

Proof. The proof is by induction on the request sequence σ from r_k to the end of the request sequence.

After serving r_k, ALG matches the configuration of OPT_{nl}. As they have the same configuration, $\text{srv}_{\text{OPT}_{nl}}(a, r_k) = \text{srv}_{\text{ALG}}(a, r_k)$ for all $a \in \alpha$.

Assume that $\text{srv}_{\text{OPT}_{nl}}(a, r_j) \geq \text{srv}_{\text{ALG}}(a, r_j)$ for all $a \in \alpha$ where r_j is from r_k to r_{l-1}. Let s be the server used by OPT_{nl} to serve r_l, let t be the server used by ALG and let $b \in \alpha$ be the arm where r_l is located.

If s and t are both located at the centre before serving r_l, the number of servers on b increase by 1 for both ALG and OPT_{nl}. Therefore, $\text{srv}_{\text{OPT}_{nl}}(b, r_l) \geq \text{srv}_{\text{ALG}}(b, r_l)$.

If s is located at the centre and t is not before serving r_l, then the number of servers on b increase by 1 for OPT_{nl}. Therefore, $\text{srv}_{\text{OPT}_{nl}}(b, r_l) \geq \text{srv}_{\text{ALG}}(b, r_l)$.

If s and t are both located on b before serving r_l, then the number of servers on b does not change for either ALG or OPT_{nl}. Therefore, $\text{srv}_{\text{OPT}_{nl}}(b, r_l) \geq \text{srv}_{\text{ALG}}(b, r_l)$.

If s is on b and t is at the centre before serving r_l, then the number of servers on b increase by 1 for ALG. The first bit of advice is based on the minimum weighted matching of the servers of OPT and ALG on the arm of r_l that matches s to an adjacent server, t. The servers at the centre are not used in the minimum weighted matching. If OPT has more servers than ALG on the arm, the difference will be made up by dummy servers of ALG being placed at the centre. So, if t is at the centre before serving r_l and s is not, this implies that $\text{srv}_{\text{OPT}_{nl}}(b, r_{l-1}) > \text{srv}_{\text{ALG}}(b, r_{l-1})$ as this is the only manner in which s can be matched to a dummy server of ALG. Therefore, $\text{srv}_{\text{OPT}_{nl}}(b, r_l) \geq \text{srv}_{\text{ALG}}(b, r_l)$.

Finally, after serving r_l, if OPT_{nl} moves s to the centre, then ALG will move t to the centre. The number of servers on b decreases by 1 for both OPT_{nl} and ALG. Therefore, $\text{srv}_{\text{OPT}_{nl}}(b, r_l) \geq \text{srv}_{\text{ALG}}(b, r_l)$. □

Lemma 5.4. *Let r_l be a request of σ where $l > k$. Let $\phi_{a,l}$ be the weight of the minimum weighted matching between the servers of ALG and OPT_{nl} of an arm $a \in \alpha$ after operation 3 where α is the set of all the arms of the spider graph. Let $\phi'_{a,l}$ be the weight of the minimum weighted matching between the servers of ALG and OPT_{nl} of an arm $a \in \alpha$ after operation 1. Let $\Phi_l = \sum_{a\in\alpha} \phi_{a,l}$ and $\Phi'_l = \sum_{a\in\alpha} \phi'_{a,l}$. For each r_l, COMPLIANT-MOVE meets the three conditions of Lemma 5.2.*

Proof. Let $\text{dist}(x, y)$ be the distance between two points x and y, and let c be the centre. Let s be the server used by OPT_{nl} to serve request r_l and let t be the server used by ALG to serve request r_l.

The cost to OPT_{nl} to serve r_l can be broken into the cost for operation 1 and the cost for operation 3 to OPT_{nl}. The cost to ALG to serve r_l can be broken into the cost for operation 2 and the cost for operation 3 to ALG. So, $\text{OPT}_{nl}(r_i) = \text{OPT}_{nl}(r_i)' + \text{OPT}_{nl}(r_i)''$ and $\text{ALG}(r_i) = \text{ALG}(r_i)' + \text{ALG}(r_i)''$ where $\text{OPT}_{nl}(r_i)'$ is the cost for operation 1, $\text{OPT}_{nl}(r_i)''$ is the cost for operation 3 to OPT, $\text{ALG}(r_i)'$ is the cost for operation 2 and $\text{ALG}(r_i)''$ is the cost for operation 3 to ALG.

Let p be the server of ALG to which s is matched for the minimum weighted matching used to calculate Φ_{l-1} and let a be the arm where r_l is positioned. If s is at the centre, then we could take it to be matched to a dummy server of ALG at the centre for the minimum weighted matching used to calculate Φ_{l-1} as Claim 5.3 shows that OPT_{nl} has more servers on a than ALG and, if both s and p are at the centre, the weight between s and p would be 0. As the only server moved between the configuration of the servers of ALG and OPT used to calculate Φ_{l-1} and Φ'_l is s, $\Phi'_l - \Phi_{l-1} \leq \text{dist}(r_l, s) = \text{OPT}_{nl}(r_i)'$, meeting condition 1 of Lemma 5.2 as the greatest increase would result in s moving away from p on a.

By Lemma 4.1 and the definition of the first bit of advice, there is a minimum weighted matching for the configurations of the servers of ALG and OPT used to calculate Φ'_l such that t and s are matched. ALG moves t to r_l at operation 2. If s and t remain at r_l after operation 3, then the only change in the configurations of the servers of ALG and OPT used to calculate the minimum weighted matching at Φ'_l and Φ_l is t. Since, t moves to s, $\Phi_l - \Phi'_l = -\text{dist}(t, r_l) = \text{ALG}(r_i)'$, meeting condition 2 of Lemma 5.2.

If s and t move to the centre at operation 3, then they will not be included in the minimum weighted matching of Φ_l. However, as shown in the previous paragraph, when s and t remain, the weight of the minimum weighted matching decreases by $-\text{dist}(t, r_l)$. This decrease is when s is matched to t and, since they are at the same position, contribute a weight of 0 to Φ_l. Therefore, if s and t are no longer on the arm, then $\Phi_l - \Phi'_l$ must be less than or equal to $-\text{dist}(t, r_l) = \text{ALG}(r_i)'$, meeting condition 2 of Lemma 5.2.

For operation 3, ALG will only move t to the centre if s is moved to the centre as per the definition of the second bit of advice. Since both s and t served r_l, if they move to the centre, the cost will be $\text{dist}(r_l, c)$ to both ALG and OPT_{nl}. In that case, $\text{OPT}_{nl}(r_i)'' = \text{ALG}(r_i)'' = \text{dist}(r_l, c)$. If s and t do not move to the centre, then $\text{OPT}_{nl}(r_i)'' = \text{ALG}(r_i)'' = 0$. Therefore, condition 3 of Lemma 5.2 is met. □

Claim 5.5. *Let r_l be a request where $l > k$. If the advice for request r_l indicates to ALG to use a dummy server, then there exists a server of ALG at the centre.*

Proof. Let a be the arm where r_l is positioned. From Claim 5.3, OPT_{nl} will have the same number of servers or more servers on each arm of the spider graph than ALG and, as per the definition of the first bit of advice, if the advice indicates to use a dummy server of ALG then OPT_{nl} has more servers on a than ALG. Therefore, as ALG and OPT_{nl} have the same number of servers, there must be a server of ALG at the centre. □

Corollary 5.6. *COMPLIANT-MOVE is correct.*

Proof. Follows immediately from Claim 5.5. □

Theorem 5.7. *COMPLIANT-MOVE is 1-competitive.*

Proof. From r_1 to r_k, all requests are served by the closest server. Therefore, $\text{ALG}(r_1, \ldots, r_k) \leq kd$ where d is the diameter of the spider graph. Immediately after serving the kth request, ALG matches the configuration of the OPT_{nl}. This can be bounded by kd.

From r_{k+1} to r_n, $\text{ALG}(r_{k+1}, \ldots, r_n) = \text{OPT}(r_{k+1}, \ldots, r_n) + \Phi_k$. This follows immediately from Lemmas 5.4 and 5.2.

For the entire request sequence, $\text{ALG}(\sigma) \leq \text{OPT}(\sigma) + \Phi_k + 2kd$. Since the configurations of ALG and OPT_{nl} match after serving r_k, $\Phi_k = 0$. Therefore, $\text{ALG}(\sigma) \leq \text{OPT}(\sigma) + 2kd$ □

6 Conclusions

To date, there is no lower bound established for the k-server problem with advice. In [7], the authors present a deterministic algorithm that is $k^{O(\frac{1}{b})}$ competitive. All of the results presented here are 1-competitive or better with 2 or less bits of advice and they are the best solution to date for the particular problem. Specifically, our algorithms for the k-server problem on the line and the cycle take 1 bit of advice and are optimal which is clearly the best results possible.

The work presented here could be continued by extending the algorithm for the k-server problem with advice on the spider graph to the tree and, following that, establish a lower bound for the k-server problem with advice on the tree. Ideally, a lower bound for the k-server problem with advice on the tree could be useful in establishing a lower bound for the k-server problem with advice on a general graph.

In a more general context, it would be interesting to continue exploring online algorithms with advice and to establish results, ideally lower bounds as a function of the bits of advice, for those algorithms. As online algorithms often fall into the general category of approximation algorithms, it would be interesting to compare the results of online algorithms with and without advice to that of the approximability results of the offline version to see if any correlations can be established.

References

[1] Shai Ben-David, Allan Borodin, Richard M. Karp, Gábor Tardos, and Avi Wigderson. On the power of randomization in online algorithms (extended abstract). In *STOC*, pages 379–386. ACM, 1990.

[2] Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královic, Richard Královic, and Tobias Mömke. On the advice complexity of online problems. In Yingfei Dong, Ding-Zhu Du, and Oscar H. Ibarra, editors, *ISAAC*, volume 5878 of *Lecture Notes in Computer Science*, pages 331–340. Springer, 2009.

[3] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, New York, NY, USA, 1998.

[4] Marek Chrobak, Howard J. Karloff, Thomas H. Payne, and Sundar Vishwanathan. New results on server problems. In *SODA*, pages 291–300, 1990.

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

[6] Stefan Dobrev, Rastislav Královič, and Dana Pardubská. 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*, pages 247–258, Berlin, Heidelberg, 2008. Springer-Verlag.

[7] Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. In *ICALP '09: Proceedings of the 36th International Colloquium on Automata, Languages and Programming*, pages 427–438, Berlin, Heidelberg, 2009. Springer-Verlag.

[8] Amos Fiat, Yuval Rabani, Yiftach Ravid, and Baruch Schieber. A deterministic $o(k^3)$-competitive k-server algorithm for the circle. *Algorithmica*, 11(6):572–578, 1994.

[9] Sandy Irani and Anna R. Karlin. On online computation. In *Approximation Algorithms for NP-Hard Problems, chapter 13*, pages 521–564. PWS Publishing Company, 1997.

[10] Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel Dominic Sleator. Competitive snoopy caching. *Algorithmica*, 3:77–119, 1988.

[11] Elias Koutsoupias and Christos H. Papadimitriou. Beyond competitive analysis. In *FOCS*, pages 394–400. IEEE, 1994.

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

[13] Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for on-line problems. In *STOC*, pages 322–333. ACM, 1988.

[14] Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for server problems. *J. Algorithms*, 11(2):208–230, 1990.

[15] Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

[16] Eric Torng. A unified analysis of paging and caching. *Algorithmica*, 20(2):175–200, 1998.

[17] Neal E. Young. *Competitive Paging and Dual-Guided Algorithms for Weighted Caching and Matching*. PhD thesis, Princeton University, Department of Computer Science, 1991.

A Full Proof for Lemma 4.1

Lemma A.1. *Let A and B be two sets of points of equal size on the line. For any $a \in A$ there is a minimum weighted matching between the points of A and the points of B such that a is matched to a point $b \in B$ that is adjacent to a on the line.*

Proof. Given a minimum weighted matching between the sets A and B on the line, let $a \in A$ be matched to a server $e \in B$ such that e is not adjacent to a and let $b \in B$ be a point that is adjacent to a and between a and e on the line. Let $c \in A$ be the point to which b is matched. Let $\text{dist}(x, y)$ be the distance between the points x and y on the line. Without loss of generality, assume that b and e are to the right of a.

If c is between b and e, then $\text{dist}(a, b) + \text{dist}(c, e) \leq \text{dist}(a, e)$, so a can be matched to b and c can be matched to e without increasing the cost of the matching.

If c is to the right of e, then $\text{dist}(a, b) + \text{dist}(e, c) \leq \text{dist}(a, e) + \text{dist}(b, c)$ since $\text{dist}(a, b) + 2\text{dist}(b, e) + \text{dist}(e, c) = \text{dist}(a, e) + \text{dist}(b, c)$. So, a can be matched to b and c can be matched to e without increasing the cost of the matching.

If c is to the left of a, then $\text{dist}(a, b) + \text{dist}(c, e) = \text{dist}(c, b) + \text{dist}(a, e)$ since $\text{dist}(a, b) + \text{dist}(c, e) = \text{dist}(a, e) + 2\text{dist}(a, b) + \text{dist}(c, b) = \text{dist}(a, b) + \text{dist}(c, e)$. So, a can be matched to b and c can be matched to e without increasing the cost of the matching. □

B Full Proof for Lemma 4.2

Lemma B.1. *Let Φ'_i and Φ_i be two functions that map the configurations of ALG and OPT to a real number where $\Phi'_i \geq 0$ and $\Phi_i \geq 0$ such that Φ'_i is calculated at the end of an operation, that is not the final operation, performed by ALG or OPT to serve r_i while Φ_i is calculated at the end of the final operation performed by ALG or OPT for r_i. Given the two following conditions for each request, r_i, in σ:*

- $OPT(r_i) \geq \Phi'_i - \Phi_{i-1}$
- $-ALG(r_i) \geq \Phi_i - \Phi'_i$

then $ALG(\sigma) \leq OPT(\sigma) + \Phi_0$

Proof. Adding the two conditions gives

$$\text{OPT}(r_i) - \text{ALG}(r_i) \geq \Phi_i - \Phi_{i-1}.$$

Therefore,

$$\text{ALG}(r_i) \leq \text{OPT}(r_i) + \Phi_{i-1} - \Phi_i.$$

Summing over all the requests gives

$$\sum_{r_i \in \sigma} \text{ALG}(r_i) \leq \sum_{r_i \in \sigma} \text{OPT}(r_i) + \sum_{r_i \in \sigma} (\Phi_{i-1} - \Phi_i).$$

The last sum is a telescoping sum. So,

$$\text{ALG}(\sigma) \leq \text{OPT}(\sigma) + \Phi_0 - \Phi_n$$

where n is the length of σ. Since $\Phi_n \geq 0$,

$$\text{ALG}(\sigma) \leq \text{OPT}(\sigma) + \Phi_0.$$

□

C Full Proof for Lemma 5.2

Lemma C.1. *Let Φ'_i and Φ_i be two functions that map the configurations of ALG and OPT to a real number where $\Phi'_i \geq 0$ and $\Phi_i \geq 0$ such that Φ'_i is calculated at the end of an operation, that is not the final operation, performed by ALG or OPT to serve r_i while Φ_i is calculated at the end of the final operation performed by ALG or OPT for r_i. Given the three following conditions for each request, r_i, in σ where $OPT(r_i) = OPT(r_i)' + OPT(r_i)''$ and $ALG(r_i) = ALG(r_i)' + ALG(r_i)''$:*

- $OPT(r_i)' \geq \Phi'_i - \Phi_{i-1}$
- $-ALG(r_i)' \geq \Phi_i - \Phi'_i$
- $OPT(r_i)'' = ALG(r_i)''$

then $ALG(\sigma) \leq OPT(\sigma) + \Phi_0$

Proof. Adding the first two conditions gives

$$\mathrm{OPT}(r_i)' - \mathrm{ALG}(r_i)' \geq \Phi_i - \Phi_{i-1}.$$

Therefore,

$$\mathrm{ALG}(r_i)' \leq \mathrm{OPT}(r_i)' + \Phi_{i-1} - \Phi_i.$$

Using condition 3,

$$\mathrm{ALG}(r_i)' + \mathrm{ALG}(r_i)'' \leq \mathrm{OPT}(r_i)' + \mathrm{OPT}(r_i)'' + \Phi_{i-1} - \Phi_i$$

which is equivalent to

$$\mathrm{ALG}(r_i) \leq \mathrm{OPT}(r_i) + \Phi_{i-1} - \Phi_i.$$

Summing over all the requests gives

$$\sum_{r_i \in \sigma} \mathrm{ALG}(r_i) \leq \sum_{r_i \in \sigma} \mathrm{OPT}(r_i) + \sum_{r_i \in \sigma} (\Phi_{i-1} - \Phi_i).$$

The last sum is a telescoping sum. So,

$$\mathrm{ALG}(\sigma) \leq \mathrm{OPT}(\sigma) + \Phi_0 - \Phi_n$$

where n is the length of σ. Since $\Phi_n \geq 0$,

$$\mathrm{ALG}(\sigma) \leq \mathrm{OPT}(\sigma) + \Phi_0.$$

□