The k-client Problem*†

Houman Alborzi‡

Department of Computer Science
University of Maryland

Eric Torng, Patchrawat Uthaisombut, and Stephen Wagner

Department of Computer Science and Engineering
Michigan State University

*A preliminary version of this paper appeared in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, 1997.

†This work was supported in part by NSF CAREER grant CCR-9701679, a grant from IBM, and an MSU Research Initiation grant.

‡This author was a Masters student at Michigan State University when this research was conducted.

The k-client problem

Eric Torng
Department of Computer Science and Engineering
3115 Engineering Building
Michigan State University
East Lansing, MI 48824
torng@cse.msu.edu

Abstract

Virtually all previous research in on-line algorithms has focused on *single-threaded systems* where only a *single* sequence of requests compete for system resources. To model *multi-threaded on-line systems*, we define and analyze the k-client problem, a dual of the well-studied k-server problem. In the basic k-client problem, there is a single server and k clients, each of which generates a sequence of requests for service in a metric space. The crux of the problem is deciding which client's request the single server should service rather than which server should be used to service the current request. We also consider variations where requests have non-zero processing times and where there are multiple servers as well as multiple clients.

We evaluate the performance of algorithms using several cost functions including maximum completion time and average completion time. Two of the main results we derive are tight bounds on the performance of several commonly studied disk scheduling algorithms and lower bounds of $\frac{\lg k}{2}+1$ on the competitive ratio of any on-line algorithm for the maximum completion time and average completion time cost functions when k is a power of 2. Most of our results are essentially identical for the maximum completion time and average completion time cost functions.
Keywords: on-line algorithms, multi-threaded systems, disk scheduling, competitive analysis, maximum response time, average completion time

1 Introduction

In most previous studies of on-line algorithms, researchers have assumed that the system or algorithm must cope with a single request sequence; in particular, at any given time, there is at most one outstanding request in the system and future requests will not arrive until the current request is serviced (in some problems, the system is allowed to choose to not service the current request). Because of this assumption, the underlying problem addressed by most previous work in on-line algorithms has been deciding which system resource(s), if any, should be allocated to service the current request. Two of the many examples of interesting problems which fall into this single request sequence model are the paging problem [28, 23, 15] and its generalizations, the k-server and generic task system problems [23, 21, 5].

While the single request sequence model captures many important problems, there are many others which do not fall into this category, such as some operating system scheduling problems [18, 12, 13, 24] and some real-time scheduling problems [20, 4, 11]. In a typical problem, there is a single system resource such as a processor and, at any given time, there are multiple requests in the system waiting to be serviced. As a result, the underlying problem is deciding which current request should the system service rather than which system resource to use. Note that in many cases there are multiple resources and multiple requests so the system needs to decide which requests to service as well as which system resources to use.

While this multiple request model captures the scheduling aspect of many systems, it loses the thread-based or transaction-based nature of the single request model where the arrival of requests is dependent on whether or not the system processed previous requests. For example, in practice, for relatively long stretches of time there is a fixed number of users on an operating system making requests to the disk. Furthermore, each user or client generates a sequence of requests for service where each request is only generated after the previous request of the client has been serviced. This thread-based client model also describes database systems where users perform transactions which constitute a series of atomic operations in the database.

1.1 Basic Problem Definitions and Notation.

In order to capture (i) the multiple client nature and (ii) the thread-based client nature of problems such as disk scheduling, we define the k-client problem as follows.

In the basic k-client problem, there is one server, k clients, and a metric space (e.g. a plane or a line). Each client generates a sequence of requests in the metric space, and a request is serviced the moment the server, which moves at constant speed, moves to the location of the request; that is, we assume zero processing time for all requests. We define a move of the server to be a non-zero distance movement that takes it from one request to a second request with no intervening requests. In later sections, we will consider two variations of this basic problem: (i) requests which have non-zero processing times and (ii) multiple servers as well as multiple clients. We will update our definitions and notation when needed in those sections.

Let I denote an input instance to the basic k-client problem. We use the following notation to describe the number of requests in I.

- Let $n_i(I)$ denote the number of requests for client i where $1 \leq i \leq k$.
- Let $n_{max}(I) = \max_{1 \leq i \leq k} n_i(I)$.
- Let $n(I) = \sum_{1 \leq i \leq k} n_i(I)$.

We will typically omit the symbol I when the input instance referred to by I is not ambiguous.

We refer to the requests using the following notation.

- Request $r_{i,j}$ is the jth request of client i for $1 \leq i \leq k$ and $0 \leq j \leq n_i$
- For notational convenience, we assume that each client has a dummy request $r_{i,0}$ located at the initial server position.
- Input instance I can be completely described by the set of requests $\{r_{i,j}\}$.

At any time, each client has at most one request in the system; more specifically, request $r_{i,j+1}$ arrives exactly when $r_{i,j}$ has been serviced for $1 \leq i \leq k$ and $0 \leq j \leq n_i - 1$. Thus, for $1 \leq i \leq k$, $r_{i,0}$, the dummy request of each client, is serviced at time 0 and $r_{i,1}$, the first real request of each client, arrives at time 0. Because of our zero processing time assumption, we can assume without loss of generality that consecutive requests of a client are not located at the same point.

An algorithm $\mathcal{A}$ solves a specific input instance I by computing a schedule $\mathcal{A}(I)$ of server movement. We will evaluate the quality of $\mathcal{A}(I)$ using several different cost functions. We first consider the total distance cost function, a *server* oriented cost function which measures the total distance moved by the server (when we have multiple servers, this measures the total distance moved by *all* the servers). This is the cost function used in the k-server problem. We will also consider two *client* oriented cost functions which focus on the quality of service provided to each client:

1. The average completion time cost function measures the average completion time of any *entire* client; that is, we take the average of the completion times of requests r_{i,n_i} for $1 \leq i \leq k$.
2. The maximum response time cost function measures the maximum response time of any *individual request* of any client; that is, the maximum time between the servicing of any two consecutive requests of any client.

Note, the average completion time cost function is distinct from a request oriented variant which measures the average time of all individual requests of clients. We do not consider this request oriented cost function in this paper.

We use the following notation to represent the performance of an algorithm with respect to a given cost function.

- For any algorithm $\mathcal{A}$ and any input instance I
  - $\mathcal{A}^D(I)$ denotes the total distance moved by the server in schedule $\mathcal{A}(I)$.
  - $\mathcal{A}^{\mathrm{ACT}}(I)$ denotes the average completion time of all clients in schedule $\mathcal{A}(I)$.
  - $\mathcal{A}^{\mathrm{MRT}}(I)$ denotes the maximum response time of any request in schedule $\mathcal{A}(I)$.
- For any cost function CF, the competitive ratio $c_{\mathcal{A}}^{\mathrm{CF}}$ of an on-line algorithm $\mathcal{A}$ is defined as
$$c_{\mathcal{A}}^{\mathrm{CF}} = \sup_I \frac{\mathcal{A}^{\mathrm{CF}}(I)}{\mathcal{OPT}^{\mathrm{CF}}(I)}$$
  where $\mathcal{OPT}$ denotes the optimal offline algorithm for cost function CF. Note, we are overloading the $\mathcal{OPT}$ notation to represent the optimal algorithm for each cost function.

When the cost function is not ambiguous, we will drop the CF superscript. That is, we will abuse notation by using $\mathcal{A}(I)$ to represent both the schedule produced by $\mathcal{A}$ as well as its cost for the given cost function, and we will use $c_{\mathcal{A}}$ to represent its competitive ratio for the given cost function.

To simplify later proofs, we define the following notation to represent some natural characteristics of input instance I.

- Let $I_i \subseteq I$ denote the input instance consisting only of requests from client i of I.
- Let $\delta(r_{i,j}, r_{i',j'})(I)$ be the distance between requests $r_{i,j}$ and $r_{i',j'}$ of I. Since we work with a metric space, δ is symmetric.
- Let $\delta_{max}(I)$ be the maximum distance between any two requests of I, not necessarily from the same client.
- Let $d_{i,j}(I) = \delta(r_{i,j}, r_{i,j+1})(I)$ denote the distance between the jth and $j+1^{st}$ requests of client i of I.
- Let $D_i(I) = \sum_{j=0}^{n_i-1} d_{i,j}(I) = \mathcal{OPT}^D(I_i)$; that is, the minimum distance the server must move to service only client i.
- Let $D(I) = \sum_{i=1}^{k} D_i(I)$.
- Let $D_{max}(I) = \max_{1\le i\le k} D_i(I)$.

For all these items, we will typically omit I when the input instance referred to by I is not ambiguous.

Most of our upper bound results apply to any metric space unless otherwise specified. Any metric space has the following three properties: it satisfies the triangle inequality, it is symmetric, and the distance from any point to itself is 0. For lower bound proofs, we consider three primary metric spaces:

1. The $line(\infty)$ metric space represents an unbounded continuous line.
2. The $line(\Delta)$ metric space represents a discrete line segment of length Δ, a natural number, in which the distance between two consecutive points is 1.
3. The K_i metric space is a clique with i nodes where each node is distance 1 from any other node.

Note that we can replace Δ with $\delta_{max}(I)$ in all the upper and lower bound results using the $line(\Delta)$ metric space. In particular, lower bounds based on Δ in the $line(\Delta)$ metric space translate into unbounded lower bounds on the continuous line metric space $line(\infty)$. Our results for the $line(\infty)$ and general metric spaces are essentially identical.

1.2 Disk Scheduling Algorithms.

One of the goals of this work is to study the performance of several commonly used disk scheduling algorithms in the context of multithreaded systems. In particular, we wish to determine how the performance of these algorithms is related to the number of threads in the system. There are two main types of algorithms: greedy algorithms which seek to optimize some short term objective and "fair" algorithms which seek to insure all threads receive service in a reasonably timely fashion. Throughout this section, we work with the basic k-client problem which has only a single server.

We first define three commonly studied greedy algorithms. The Shortest Distance First ($\mathcal{SDF}$) algorithm moves the server to the nearest request. In disk scheduling, this algorithm is often referred to as Shortest Seek Time First. The Sequential ($\mathcal{SEQ}$) algorithm services all the requests for one client, then services all the requests of a second client, and so on. Note $\mathcal{SEQ}$ will service requests from other clients if it happens to pass over them while servicing the current client. $\mathcal{SEQ}$ is sometimes referred to as Run To Completion. The Minimum Path ($\mathcal{MP}$) algorithm examines all outstanding requests, computes the optimal path for servicing these requests which minimizes the objective cost function, and moves the server to the first request along this path. It recomputes the optimal path after every request is serviced. Three observations about $\mathcal{MP}$ are notable. First, it is significantly more complicated than either $\mathcal{SDF}$ or $\mathcal{SEQ}$. Second, $\mathcal{MP}$ denotes several different algorithms, one for each cost function we consider. That is $\mathcal{MP}$ which strives to minimize total distance behaves differently than $\mathcal{MP}$ which strives to minimize average completion time. Third, we shall show that $\mathcal{MP}$'s competitive ratio is no better than the competitive ratio of $\mathcal{SDF}$ or $\mathcal{SEQ}$ for any cost function. Thus, it is not clear that it is worthwile to implement this more complex algorithm in practice.

We now define two algorithms which strive to guarantee fairness in their service. The first, Elevator ($\mathcal{EL}$), is only defined for a line metric space. It moves the server in one direction on the line until there are no more requests in that direction. The second, First Come First Serve ($\mathcal{FCFS}$), prioritizes requests by their time of arrival. The server will move to the *oldest* currently available request. Because of our zero processing time assumption, we

assume the server will process younger requests that it encounters en route. Note, $\mathcal{FCFS}$ is essentially identical to the round robin algorithm except for the above caveat. In the case of a tie, we assume $\mathcal{FCFS}$ prioritizes the nearest oldest request first.

Our results show that the elevator algorithm has a competitive ratio which is independent of the number of threads in the system. On the other hand, all the greedy algorithms have competitive ratios which are proportional to the number of threads in the system. This result helps explain why elevator type algorithms have superior performance in heavily loaded environments.

1.3 Related Research.

We know of only two previous papers which have studied on-line problems with similar input formats. Feuerstein [14] independently developed the multiple client input model and used it to define the multi-threaded paging problem. This problem is a generalization of the traditional paging problem where each client represents an independent thread or process generating memory requests. The on-line algorithm must decide which outstanding request to service as well as which fast memory pages should be used to service each request. Feldmann et al. studied the problem of on-line scheduling of parallel jobs where the jobs arrive dynamically according to the dependencies between them [12]. These dependencies can form an arbitrary directed acyclic graph, not just a set of chains. Each job requests a certain number of processors with a specific communication configuration. The cost function they used was the maximum completion time of the schedule. In some sense, they studied a single client problem with a directed acyclic dependency graph whereas we study a multiple client problem with chain dependency graphs. Interestingly, they were able to show that any algorithm that can solve this problem with tree dependency graphs can be converted into an algorithm which solves the general problem.

There has been some study of on-line traveling salesman problems. Kalyanasundaram and Pruhs studied the setting where the nodes of a graph were only revealed when a neighbor of that node was visited, and the goal of the algorithm was to construct a tour of the graph of minimum possible length. For the case of a planar graph, they were able to develop a competitive algorithm [17]. Ausiello et al. studied the on-line traveling salesman problem where requests arrive over time independent of how the server processed previous requests [2, 3]. That is, they did not assume that requests were threaded. For this problem, they were able to develop on-line algorithms with small constant competitive ratios for general metric spaces and lines.

There is a large body of work analyzing disk scheduling algorithms [8, 29, 25, 7, 16, 27, 31, 6, 1]. This work assumes a single-threaded rather than a multithreaded environment. Most early work focused, as we do, on minimizing seek time and essentially ignored, as we do, rotational latency. Recent advances in disk technology imply that rotational latency is becoming more important, so several papers have studied disk scheduling modeling rotational latency as well [27, 1]. We focus on line graphs and general metric spaces, but in future work, we plan to expand our consideration of metric spaces to consider rotational latency as well.

Finally, on-line navigation and on-line searching problems such as the cow-path problem bear some relation to our problem [26, 10, 19, 22]. However, the goal function and the requirements on the optimal algorithm are quite different for these problems than for our problem. In most on-line searching problems, the typical goal is to find a specific unknown target. The offline algorithm which knows the location of this target can move directly to it while the on-line algorithm must probe the space for it. In our problem, the offline algorithm must visit every point the on-line algorithm does, though it may be able to do so much more efficiently.

1.4 Results and Outline.

The rest of this paper is organized as follows. In section 2, we focus on the basic k-client problem and the total distance and average completion time cost functions. Nearly all of our results are identical for these two different cost functions. In particular, we prove an upper bound of $2k-1$ for both the total distance and average completion time cost functions, and we prove lower bounds of $\frac{\lg k}{2}+1$ for both of these cost functions when k is a power of 2. These lower bounds drop to $\frac{\lg k}{2}$ when k is not a power of 2. When $k=2$, these lower bounds improve to $\frac{25}{9}$ and 3 for the total distance and average completion time cost functions, respectively. We also prove a randomized lower bound of $\Theta(\lg\lg k)$ for these cost functions.

In section 3, we analyze in detail how several commonly used disk scheduling algorithms perform on the $line(\infty)$ and $line(\Delta)$ metric spaces. We show that the competitive ratios of the $\mathcal{SDF}$ and $\mathcal{SEQ}$ algorithms are exactly $2k-1$ for both the total distance and average completion time cost functions. Furthermore, we show that the $\mathcal{MP}$ algorithm is exactly $(2k-1)$-competitive for the total distance cost function and that its competitive ratio lies between $2k-1$ and $k(2k-1)$ for the average completion time cost function. In an interesting contrast, we show that the elevator algorithm has a competitive ratio for these two cost functions that depends only on the length of the line; that is, it is independent of the number of clients. Because disk systems represent relatively short lines, particularly when processing time is factored in, this result helps explain why elevator type algorithms work well in practice, particularly when the disk is heavily loaded.

In section 4, we continue our study of the basic k-client problem but we focus on the maximum response time cost function. Our results are dramatically different. We prove a general lower bound of $\Omega(\sqrt[3]{\Delta})$ that applies to any algorithm which faces only two clients on $line(\Delta)$. This implies that no algorithm has a bounded competitive ratio on the continuous line metric space, $line(\infty)$. We also show that several greedy algorithms such as $\mathcal{SDF}$ and $\mathcal{SEQ}$ have unbounded competitive ratios, even on $line(\Delta)$. Finally, we show that algorithms such as $\mathcal{EL}$ and $\mathcal{FCFS}$ do have bounded competitive ratios on $line(\Delta)$. These results for sections 2 and 4 are summarized in Figure 1.

In section 5, we consider a generalized version of the k-client problem where requests may have non-zero processing times. With non-zero processing times, we introduce a new cost function, maximum completion time, which is now distinct from the total distance

k-clients

Cost Function	Upper Bound	Lower Bound
total distance	$2k - 1$	$\frac{\lg k}{2} + 1$
average completion time	$2k - 1$	$\frac{\lg k}{2} + 1$
maximum response time	$\Theta(\Delta)$	$\Omega(\sqrt[3]{\Delta})$

2-clients

Cost Function	Upper Bound	Lower Bound
total distance	3	$\frac{25}{9}$
average completion time	3	3
maximum response time	$\Theta(\Delta)$	$\Omega(\sqrt[3]{\Delta})$

Figure 1: Summary of Results for the basic k-client problem. The lower bounds of $\frac{\lg k}{2} + 1$ assume that k is a power of 2. These drop to $\frac{\lg k}{2}$ when k is not a power of 2. Also, the results for maximum response time assume the $line(\Delta)$ metric space.

cost function, and we focus on the maximum completion time and average completion time cost functions. We are able to show that processing times do not make the problem more complex. More specifically, for most algorithms, the worst case occurs when requests have zero processing times. Thus, the basic k-client problem which focuses on server movement captures most of the complexity of this problem.

Finally, in section 6, we consider the k-client l-server problem, a generalized version of the k-client problem where there are both multiple servers and multiple clients. We prove a general result which shows that any algorithm for the l-server problem can be combined with the $\mathcal{SEQ}$ k-client algorithm to produce a competitive k-client l-server algorithm.

In section 7, we summarize our work and describe some open problems.

2 The Basic k-client Problem

In this section, we consider the basic k-client problem and the total distance and average completion time cost functions. We prove that the greedy $\mathcal{SDF}$ algorithm is $(2k-1)$-competitive for both total distance and average completion time in any metric space, and we prove lower bounds of $\frac{\lg k}{2} + 1$ for any on-line algorithm in clique metric spaces for both total distance and average completion time when k is a power of 2. We improve these lower bounds for $k = 2$ clients.

We begin by proving some basic facts about the optimal offline algorithm and the relationships between these two different cost functions.

Fact 2.1. *For all input instances I, $\mathcal{OPT}^D(I) \geq D_{max}(I)$ and $\mathcal{OPT}^D(I) \geq \delta_{max}(I)$.*

Proof. We observe that for all input instances I, $\mathcal{OPT}^D(I) \geq D_{max}(I)$ since the cost of servicing every client cannot be less than the cost of servicing a single client; likewise,

$\mathcal{OPT}^D(I) \geq \delta_{max}(I)$ since $\mathcal{OPT}$ must move between the farthest two requests in I. □

Fact 2.2. *For all input instances I, $\mathcal{OPT}^{ACT}(I) \geq \frac{D(I)}{k}$.*

Proof. Consider any input instance I with k clients. For $1 \leq i \leq k$, $\mathcal{OPT}$ clearly cannot complete client i before $D_i(I)$, the time it takes to complete client i if client i is the only client. Thus $\mathcal{OPT}^{\mathrm{ACT}}(I)$, the average completion time incurred by $\mathcal{OPT}$, is lower bounded by $\frac{1}{k}\sum_{i=1}^{k} D_i(I) = \frac{D(I)}{k}$. □

Fact 2.3. *For all input instances I with k clients and any algorithm $\mathcal{A}$, $\frac{\mathcal{A}^D(I)}{k} \leq \mathcal{A}^{ACT}(I) \leq \mathcal{A}^D(I)$.*

Proof. The lower bound follows from the fact that at least one client must have a completion time of exactly $\mathcal{A}^D(I)$. The upper bound follows from the fact that every client can be completed in at most $\mathcal{A}^D(I)$ time units. □

2.1 Upper Bounds

Theorem 2.1. *For the total distance cost function, $c_{\mathcal{SDF}} = 2k - 1$.*

Proof. Without loss of generality, we assume the clients are labeled according to the order that $\mathcal{SDF}$ will service their last requests. We bound $\mathcal{SDF}^D(I)$ by bounding the cost of each move of $\mathcal{SDF}$. Every move begins from a request $r_{i,j}$ for $1 \leq i \leq k$ and $0 \leq j \leq n_i$. If a move starts from request $r_{i,j}$ where $j < n_i$, then the cost of the move can be upper bounded by $d_{i,j} = \delta(r_{i,j}, r_{i,j+1})$. This is true because either the server moves to request $r_{i,j+1}$, or it moves to a closer request from another client. The cost of moves starting from the final request of a client, r_{i,n_i}, can be upper bounded by δ_{max}. Note that there are only $k-1$ moves that start from the final request of a client because the server halts after the last request of the kth client is serviced. Thus,

$$\begin{aligned}\mathcal{SDF}^D(I) &\leq \sum_{i=1}^{k}\sum_{j=0}^{n_i-1} d_{i,j} + (k-1)\delta_{max} \\ &\leq \sum_{i=1}^{k} D_i + (k-1)\delta_{max}\end{aligned}$$

From $D_i \leq D_{max}$ and Fact 2.1, we conclude that

$$\mathcal{SDF}^D(I) \leq (2k-1)\mathcal{OPT}^D(I)$$

Note this upper bound is true for any metric space.

We now show that this bound is tight in any metric space that includes $line(\infty)$. More specifically, we show that $c_{SDF} > 2k - 1 - \epsilon$ for all $\epsilon > 0$ on $line(\infty)$. The lower bound instance is illustrated in Figure 2. The optimal solution for this instance is to move the

server to the far right to point $k-1$ and then to the far left to point $-n$ resulting in a cost of $n+(2k-2)$. Assuming ties are broken to $\mathcal{SDF}$'s detriment, $\mathcal{SDF}$ will move left to service all of the requests of client 1, then return to the right and service all of the requests of client 2, etc., resulting in a total cost of $(2k-1)n+O(k^2)$. For any $\epsilon>0$, there exists an n such that the competitive ratio will exceed $2k-1-\epsilon$. Note, this input instance can easily be adjusted to eliminate ties. $\square$

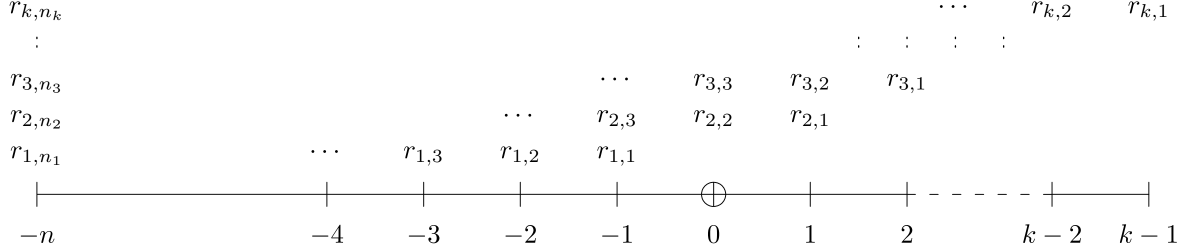

Figure 2: Lower bound instance for $\mathcal{SDF}$ when the cost function is total distance.

Theorem 2.2. *For the average completion time cost function,* $c_{\mathcal{SDF}}=2k-1$.

Proof. Without loss of generality, we assume the clients are labeled according to the order that $\mathcal{SDF}$ will service their last requests. That is, the completion time of client i is no greater than the completion time of client j in $\mathcal{SDF}(I)$ for $1\le i<j\le k$.

As in Theorem 2.1, we bound $\mathcal{SDF}^{\text{ACT}}(I)$ by bounding the cost of each move of $\mathcal{SDF}$. Every move begins from a request $r_{i,j}$ for $1\le i\le k$ and $0\le j\le n_i$.

If a move starts from request $r_{i,j}$ where $j<n_i$, i.e. $r_{i,j}$ is not a terminal request of client i, then this move adds a cost of at most $d_{i,j}$ to each unfinished client's completion time. This is true because either the server moves to request $r_{i,j+1}$, or it moves to a closer request from another client. Clearly such a move adds a cost of at most $d_{i,j}$ to the average completion time. The total cost of moves starting from nonterminal requests of all clients is at most

$$\sum_{i=1}^{k}\sum_{j=0}^{n_i-1} d_{i,j}=\sum_{i=1}^{k} D_i = D$$

If a move starts from request r_{i,n_i}, the terminal request of client i, then the cost of the move can be upper bounded by $\left(\frac{k-i}{k}\right)(D_i+\min_{i<j\le k} D_j)$. This is true for the following reasons. Clients 1 to i have finished. There are only $(k-i)$ unfinished clients. Consider the unfinished client $j>i$ such that D_j is minimized. Consider the distance between request r_{i,n_i} and the current request of client j. This distance is no more than D_i+D_j because in the worst case, the server must move from r_{i,n_i} back to $r_{i,0}=r_{j,0}$ and then to the current request of client j. Since SDF moves to the closest request, the distance it moves in this case can be no larger than D_i+D_j. Thus the completion time of the at most $(k-i)$ unfinished clients is increased by at most D_i+D_j and the average completion time is increased by at

most $\left(\frac{k-i}{k}\right)(D_i + D_j)$. The total cost of moves starting from terminal requests of all clients is at most

$$\begin{aligned}
\sum_{1\le i\le k}\left(\frac{k-i}{k}\right)(D_i + \min_{i<j\le k} D_j) &\le \sum_{1\le i\le k}\left(\left(\frac{k-i}{k}\right)D_i + \sum_{i<j\le k}\frac{D_j}{k}\right)\\
&= \sum_{1\le i\le k}\left(\frac{k-i}{k}\right)D_i + \sum_{1\le i\le k}\sum_{i<j\le k}\frac{D_j}{k}\\
&= \sum_{1\le i\le k}\left(\frac{k-i}{k}\right)D_i + \sum_{1\le j\le k}\sum_{1\le i<j}\frac{D_j}{k}\\
&= \sum_{1\le j\le k}\left(\frac{k-j}{k}\right)D_j + \sum_{1\le j\le k}\left(\frac{j-1}{k}\right)D_j\\
&= \sum_{1\le j\le k}\left(\frac{k-1}{k}\right)D_j\\
&= \left(\frac{k-1}{k}\right)D
\end{aligned}$$

The cost of moves of $\mathcal{SDF}$ starting from nonterminal requests is at most D. The cost of moves of $\mathcal{SDF}$ starting from terminal requests is at most $\left(\frac{k-1}{k}\right)D$. The combined cost is at most $\left(\frac{2k-1}{k}\right)D$. Thus, from Fact 2.2, $\mathcal{SDF}^{\text{ACT}}(I) \le (2k-1)\mathcal{OPT}^{\text{ACT}}(I)$. Note this upper bound is true for any metric space.

This bound is tight in any metric space that includes $line(\infty)$. More specifically, $c_{SDF} > 2k-1-\epsilon$ for all $\epsilon > 0$ on $line(\infty)$. The lower bound instance is illustrated in Figure 3. Note that client 1 has several requests while all other clients have exactly one request. Again assuming that ties are broken to $\mathcal{SDF}$'s detriment, $\mathcal{SDF}$ will first move to the left and service all requests of client 1. It will then service the requests of the remaining clients. The completion time of client 1 will be n_1. The completion time of client j will be $2n_1 + j - 1$. The average completion time is $(2k-1)n_1 + O(k^2)$. The optimal solution is to move all the way to the right first. The optimal average completion time is $n_1 + O(k)$. For any $\epsilon > 0$, there exists an n_1 such that the competitive ratio will exceed $2k-1-\epsilon$. Again, this input instance can easily be adjusted to eliminate ties. □

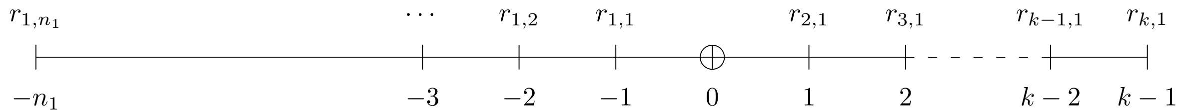

Figure 3: Lower bound instance for $\mathcal{SDF}$ when the cost function is average completion time.

2.2 General Lower Bounds

We now prove that no on-line algorithm has a competitive ratio better than $\frac{\lg k}{2}+1$ for either the total distance cost function or for the average completion time cost function when k is a power of 2. These lower bounds drop to $\frac{\lg k}{2}$ when k is not a power of 2. These results apply in clique metric spaces.

This section is organized as follows. We first define a restricted adversary strategy that will be used in the proof of both lower bounds. We then prove some basic properties about this adversary in Section 2.2.2. We then prove the actual lower bound results in Sections 2.2.3 and 2.2.4. In section 2.2.5, we show how these lower bounds can be translated to line metric spaces.

2.2.1 Definition of Adversary Strategy $A(N, k)$

The k-client problem can be described as a game between the adversary and the on-line algorithm as follows. The adversary begins the game by generating a single request for each of the k clients in the metric space. The on-line algorithm then responds by moving the server to a location where at least one request resides and servicing all requests at that position. Consider any client whose request has just been serviced. The adversary must respond by either generating another request for this client or informing the on-line algorithm that this client has no more requests. The game continues in this fashion until the on-line algorithm has been informed that all clients have no more requests.

We define an adversary strategy $A(N, k)$ that is parameterized by two integers N and k, both of which must be at least 1. It will utilize a clique with $Nk + 1$ nodes partitioned into N subcliques of size k plus an extra node which is the initial server position. We number the subcliques from 1 to N. On a first reading, it is helpful to focus on adversary strategy $A(1, k)$ which uses only subclique 1. Before we can define adversary $A(N, k)$, we first define the following notation and concepts. Also, for the remainder of this section, when we say "at any time during the game" or "at the end of the game", it will be understood that the game is taking place between adversary strategy $A(N, k)$ and any on-line algorithm $\mathcal{A}$.

Definition 2.1. *At any time during the game, let σ_i be the sequence of positions occupied by the requests of client i in the clique of size Nk. Define $\sigma_i(h)$ to be the sequence of positions occupied by the requests of client i in subclique h. Define $I(h)$ to be the restricted input instance where client i has only the sequence of requests $\sigma_i(h)$ for $1 \leq i \leq k$.*

Definition 2.2. *At any time during the game, for $1 \leq i \leq k$ and $1 \leq h \leq N$, client i has* h-length n *if $\sigma_i(h)$ currently has length n. Note the h-length of a client may increase over time as the adversary continues to generate requests for client i in subclique h.*

Definition 2.3. *At any time during the game, client i* subsumes *client j in subclique h if $\sigma_j(h)$ is a proper subsequence of $\sigma_i(h)$.*

Definition 2.4. *Let $I_\phi = \bigcup_{i\in\phi} I_i$ where $\phi \subseteq \{1, \ldots, k\}$.*

Thus, $I(h)_\phi$ is the restricted input instance consisting only of the clients in ϕ and their requests that occur in subclique h.

Fact 2.4. *At any time during the game,* $\mathcal{OPT}^D(I(h)_\phi) = D_i(I(h))$ *if* $i \in \phi$ *and* $\forall j \in \phi$, *i subsumes j in subclique h*

Proof. The cost of servicing all of the requests of client i in subclique h is $D_i(I(h))$. In order to service client i in subclique h, the server must visit all the locations in $\sigma_i(h)$ in order. It follows that the server will also visit all of locations of $\sigma_j(h)$ in order for any client j subsumed by i in subclique h. □

Definition 2.5. *At any time during the game, client i* covers *client j in subclique h if client i subsumes client j in subclique h and there is no client l that is simultaneously subsumed by client i in subclique h and subsumes client j in subclique h. A client j is* uncovered *in subclique h if there is no client i such that i covers j in subclique h.*

We now define the notion of a client i being dead/alive/critically alive in subclique h.

Definition 2.6. *At any time during the game, for $1 \leq i \leq k$, and for $1 \leq h \leq N$,*

- *Let l_i be the most recent request of client i that has appeared. Since the game begins with the adversary generating one request for each client, l_i must be defined.*
- *Let c_i be the subclique in which l_i appears.*
- *Client i is* dead *if l_i has been serviced,* **and** *the adversary has made known to the on-line algorithm that client i has no more requests.*
- *Client i is* alive *if it is not dead. An alive client* lives *in subclique h if $c_i = h$. Note that if $c_i = h$, it is not always the case that client i lives in subclique h because client i may be dead.*
- *Client i is* critically alive *if it is alive, request l_i has just been serviced, and the adversary has not yet responded.*

Definition 2.7. *At any time during the game, client i is* graftable *in subclique h if it has requests in subclique h, it is uncovered in subclique h, and it no longer lives in subclique h.*

Definition 2.8. *At any time during the game, if we focus on a single subclique h, our adversary is restricted to making only the following three types of moves.*

- *The adversary can* plant *client i in subclique h by placing the next request of client i on a node in subclique h that has not yet been the location of any other request.*
- *The adversary can* terminate *a critically alive client i which lives in subclique h by informing the on-line algorithm that client i has no more requests in subclique h. Note, the adversary may or may not choose to plant the next request of client i in subclique $h+1$ for $1 \leq h \leq N-1$.*

- *The adversary can* concatenate *a graftable client j in subclique h with h-length n to a critically alive client i in subclique h with the same h-length by making the $n + l^{th}$ request of client i in subclique h be on the same node as the lth request of client j in subclique h for $l = 1, ..., n$. Note that j is no longer graftable in subclique h.*

Definition 2.9. *Adversary strategy $A(N, k)$ is defined as follows.*

1. *The adversary begins the game by planting clients 1 through k in subclique 1.*
2. *In each later turn, the adversary responds only if there is a critically alive client i. If there is such a client i, let h be the subclique client i lives in, and let n be the h-length of client i. The adversary responds as follows.*
   - *(a) If there is a graftable client j in subclique h with h-length n, the adversary* concatenates *client j to client i.*
   - *(b) If there is no graftable client j in subclique h with h-length n, $A(N, k)$ responds as follows.*
     - *i. The adversary* terminates *client i in subclique h (making client i graftable in subclique h).*
     - *ii. If $\mathcal{A}$ has serviced at most $N - 1$ requests, the adversary* plants *client i into subclique $h + 1$. Otherwise, client i is dead.*

2.2.2 Properties of the Adversary Strategy

We now prove some properties about the adversary strategy $A(N, k)$.

Lemma 2.1. *For $1 \leq i \leq k$, $1 \leq h \leq N$, and at any time during the game, the h-length of client i is 0 or is 2^l for some integer $l \geq 0$.*

Proof. We first observe that the adversary can only increase the h-length of a client using the planting or concatenation operations. After planting, client i has h-length $1 = 2^0$. Assuming the lemma holds before a concatenation occurs, it must also hold after concatenation since concatentation can only occur between two clients of equal length. □

Lemma 2.2. *For $1 \leq i \leq k$, $1 \leq h \leq N$, and any time during the game, client i is covered by at most one other client in subclique h.*

Proof. We first observe that when a client is planted in subclique h, it is not covered since its single request is placed on an otherwise unoccupied node $v(i)$ of subclique h. Afterwards, no other client will contain request $v(i)$ unless client i is concatenated to the end of some other client j. Thus, the only client which can cover client i at this time is client j, and client j may not exist. Assuming such a client j does exist, client i is no longer graftable which means that the only clients which can contain node $v(i)$ are those which subsequently subsume client j. However, by the definition of cover, these clients will not cover client i and the result follows. □

To help understand the adversary strategy, it is useful to consider the graphical representation of the covers relation in each of the subcliques.

Definition 2.10. *For $1 \leq h \leq N$ and any time during the game, the* cover graph H_h *of subclique h is a directed graph with k nodes labeled 1 through k. An arc exists from node i to node j in H_h if client i covers client j in subclique h.*

In a general case, the N cover graphs may not help convey the structure of the adversary strategy and the input instance, but for $A(N, k)$, the N cover graphs do help. In particular, our adversary $A(N, k)$ will construct an input instance such that at any time during the game, each H_h will be a collection of directed binomial trees; furthermore, at the end of the game, each H_h will be a directed binomial heap [30, 9].

Corollary 2.1. *For $1 \leq h \leq N$ and any time during the game, the cover graph H_h generated by adversary $A(N, k)$ will consist of a set of directed trees.*

Proof. This follows immediately from Lemma 2.2. □

Definition 2.11. *At any time during the game, let T be a tree in H_h. Define $C(T)$ to be the set of clients which have nodes in tree T, and define the root client $r(T)$ to be the client corresponding to the root node of T.*

Lemma 2.3. *For $1 \leq h \leq N$, $1 \leq i \leq k$, and any time during the game, the following statements are true.*

1. *H_h consists of a collection of directed binomial trees [30, 9].*
2. *If client i has h-length $2^l > 0$ for some integer l, then the maximal subtree in H_h rooted at node i is a directed binomial tree with height l containing 2^l nodes.*
3. *For any tree $T \in H_h$, there is at most 1 client in $C(T)$ which lives in subclique h. Furthermore, if there is such a client, it is the root client $r(T)$.*
4. *If client i is in $C(T)$ and client j is in $C(T')$ where $T \neq T'$ are both trees in H_h, then $\sigma_i(h)$ and $\sigma_j(h)$ occupy disjoint sets of nodes in subclique h.*

Proof. Most of these properties follow trivially from the definition of our adversary, the three operations our adversary can perform, the k-client game, and the definition of binomial trees. The key observation is that when client j is concatenated to client i in subclique h, the only change to H_h is the addition of an arc from node i to node j. □

Corollary 2.2. *At any time during the game, the on-line algorithm $\mathcal{A}$ can service at most one request per turn.*

Proof. The proof follows from properties 3 and 4 of Lemma 2.3. □

Lemma 2.4. *For $1 \leq h \leq N$, $1 \leq i \leq k$, and any time during the game, there is at most 1 graftable client with h-length 2^j in subclique h for $j \geq 0$.*

Proof. Fix j and h. Initially, there are no graftable clients in subclique h so there are no graftable clients with h-length 2^j in subclique h. The termination step of the adversary is the only place where the number of graftable clients in subclique h can increase. However, for the termination step to be executed, the number of graftable clients with h-length 2^j must be 0. Otherwise, a concatenation would have been performed instead. Furthermore, after the execution of this step, the number of graftable clients with h-length 2^j is exactly 1. Therefore, the result follows. □

Definition 2.12. *Let $B(n)$ be the set of 1-valued bits of the binary representation of n where bit 0 is the least significant bit.*

For example,
$$B(15) = B(1111_2) = \{3, 2, 1, 0\}$$
$$B(16) = B(10000_2) = \{4\}.$$

Note that
$$\lfloor n/2^b \rfloor \mod 2 = \begin{cases} 0 & \text{if } b \notin B(n) \\ 1 & \text{if } b \in B(n). \end{cases}$$

For example,
$$\lfloor 1000100_2/2^2 \rfloor \mod 2 = 1 \quad \text{since} \quad B(1000100_2) = \{6, 2\} \ni 2$$
$$\lfloor 1101111_2/2^4 \rfloor \mod 2 = 0 \quad \text{since} \quad B(1101111_2) = \{6, 5, 3, 2, 1, 0\} \not\ni 4.$$

Lemma 2.5. *For any subclique h and any $i \geq 1$, if there are exactly i clients with h-length greater than 0 at the end of the game, then*
(1) there is exactly 1 uncovered client with h-length 2^b in subclique h for each $b \in B(i)$, and
(2) there are no uncovered clients with h-length b in subclique h for $b \notin B(i)$.
That is, the underlying undirected graph of H_h is a binomial heap [30, 9] with i nodes.

Proof. Fix i and h. Let x_b be the number of alive clients with h-length 2^b *created* by the adversary during the course of its operation. Let y_b be the number of uncovered clients with h-length 2^b *left* when the adversary terminates its operation. From the operation of the adversary, in particular steps (2.a) and (2.b), it follows that that for any nonnegative integer b,

$$x_b = \lfloor i/2^b \rfloor \quad \text{and}$$
$$y_b = x_b \mod 2 = \begin{cases} 0 & \text{if } b \notin B(i) \\ 1 & \text{if } b \in B(i). \end{cases}$$

Thus, the result follows. □

Lemma 2.6. *For any subclique h and $1 \le i \le k$, if there are exactly i clients with h-length greater than 0 at the end of the game, then the number of requests in subclique h generated by $A(N,k)$ is*

$$\sum_{b \in B(i)} 2^{b-1}(b+2) \ge \begin{cases} \frac{i}{2}\lg i + i & \text{if } i \text{ is a power of 2} \\ \frac{i}{2}\lg i & \text{otherwise.} \end{cases}$$

Proof. From Lemma 2.3, for $1 \le h \le N$, we can determine the number of requests generated in subclique h at the end of the game by summing the number of nodes in the binomial subtree rooted at each node in the final cover graph H_h. From Lemma 2.5, H_h will contain exactly $|B(i)|$ disjoint binomial trees, one of size 2^b for each $b \in B(i)$.

Fix b in $B(i)$. Applying well known properties of binomial trees, the binomial subtree of size 2^b in H_h will contain $2^{b-1}/2^j$ nodes which are the roots of binomial trees of size 2^j for $j = 0, \ldots, b-1$. Thus, there are a total of $2^{b-1}(b+2)$ requests represented by this binomial tree. Therefore, the number of requests represented by all binomial trees in H_h is $\sum_{b \in B(i)} 2^{b-1}(b+2)$.

If i is a power of 2, then $i = 2^g$ for some integer $g \ge 0$. Thus, there are $2^{g-1}(g+2)$ requests which is $\frac{i}{2}\lg i + i$. If i is not a power of 2, then from Lemma A.2, the number of requests is lower bounded by $\frac{i}{2}\lg i$. □

We will be particularly interested in the input instances generated by adversary $A(1,k)$ where k is a power of 2. These input instances have the following characteristics. The initial request of each of the k clients is on a different node, no client has more than one request on any node, and there are no requests on node 0. There is a single client with k requests, and for each i, $0 \le i < (\lg k) - 1$, there are $k/2^{i+1}$ clients with 2^i requests. Figure 4 shows a possible input instance and its corresponding cover graph for $k = 8$.

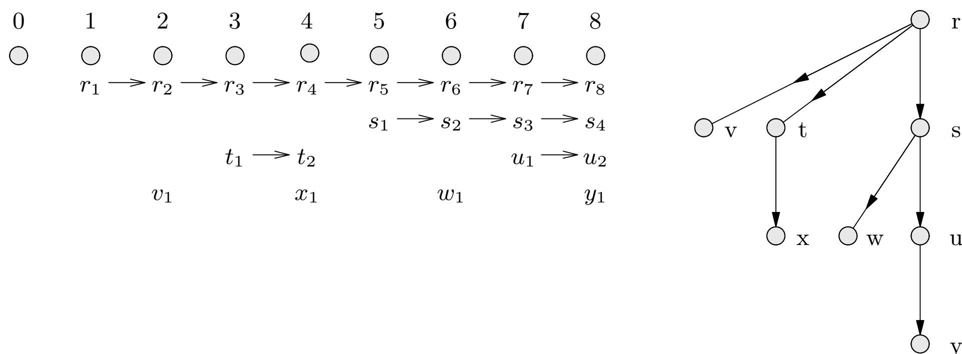

Figure 4: An example $A(1,8)$ lower bound instance and its cover graph. Note, all edges in the clique graph of size 9 have been removed. Also, the nodes have been displayed in a linear fashion to emphasize the structure of the clients.

2.2.3 General lower bound for the total distance cost function

Theorem 2.3. *For the total distance cost function and any on-line algorithm $\mathcal{A}$,*

$$c_{\mathcal{A}}^{D} \geq \begin{cases} \frac{1}{2}\lg k + 1 & \text{if } k \text{ is a power of 2} \\ \frac{1}{2}\lg k & \text{otherwise} \end{cases}$$

on the K_{k+1} metric space.

Proof. We use adversary $A(1,k)$ to prove this theorem. In particular, we use K_{k+1} which contains only one subclique. In this case, the adversary description can be simplified by making clause (2.b) simply terminate the client with no option to plant the client in the next subclique.

We first bound the on-line cost incurred by $\mathcal{A}$. From Corollary 2.2, the cost incurred by $\mathcal{A}$ will be the number of requests in the input instance. From Lemma 2.6, there are a total of $\frac{k}{2}\lg k + k$ requests if k is a power of 2. If k is not a power of 2, the number of requests is lower bounded by $\frac{k}{2}\lg k$.

We now show the optimal offline cost is k. At the end of the game, H_1 consists of a collection of disjoint binomial trees. Let T be a tree in H_1 at the end of the game. The root client $r(T)$ subsumes all clients in $C(T)$. Thus, from Fact 2.4, the optimal offline algorithm can service all requests of all clients in $C(T)$ by servicing the requests of $r(T)$ in order. If the optimal offline algorithm does this for each tree in H_1, it will service all requests by visiting each node of subclique 1 exactly once for a total cost of k. No algorithm can do better than this since there are k distinct positions containing requests, so the offline cost follows.

Dividing the on-line cost by the offline cost, we get the final result. □

2.2.4 General lower bound for the average completion time cost function

Theorem 2.4. *For the average completion time cost function, any on-line algorithm $\mathcal{A}$, and any integer $N \geq 1$,*

$$c_{\mathcal{A}}^{ACT} \geq \begin{cases} (\frac{\lg k}{2} + 1)(\frac{2N+k-1}{2N+2k^2-2}) & \text{if } k \text{ is a power of 2} \\ (\frac{\lg k}{2})(\frac{2N+k-1}{2N+2k^2-2}) & \text{otherwise} \end{cases}$$

on the K_{Nk+1} metric space.

Proof. Adversary $A(1,k)$ used in Section 2.2.3 was concerned with forcing the on-line algorithm $\mathcal{A}$ to traverse nodes as many times as possible while still allowing the optimal offline algorithm to service all requests by traversing each node only once. This adversary strategy is not appropriate for the average completion time cost function because this strategy allows $\mathcal{A}$ to complete many clients in a short amount of time. We need an adversary strategy which

- forces the on-line algorithm $\mathcal{A}$ to traverse the nodes many times while the offline algorithm can service all requests by traversing each node only once, and

- prevents the on-line algorithm $\mathcal{A}$ from completing any client too quickly.

Thus, we use $A(N, k)$ which may terminate a client in a subclique but keeps the client alive by planting it in the next subclique.

We first bound the on-line cost incurred by $\mathcal{A}$. From Corollary 2.2, $\mathcal{A}$ services at most one request when it visits a node. From step (2.b.ii), after the first $N-1$ requests have been serviced, no client has been completed. Servicing the first $N-1$ requests requires at least $N-1$ time steps, and this contributes $N-1$ to the average completion time. After the first $N-1$ requests are serviced, each client has at least one more request. The cost of servicing these requests contributes at least $\frac{1}{k}\sum_{i=1}^{k} i = \frac{k+1}{2}$ to the average completion time. Thus, the cost incurred by $\mathcal{A}$ is lower bounded by $N-1+\frac{k+1}{2}$.

Next, we compute an upper bound of the optimal cost. We will calculate the average completion time incurred by the obvious algorithm which services the subcliques in order. Clearly this is an upper bound on the cost of the optimal solution.

In an analogous fashion to the previous section, there exists a server path which visits each occupied node of a subclique exactly once and services all the requests on that subclique. The server simply services the requests of the root of each tree in H_h in order. Thus, if H_h has i nodes, then the time required to service all requests in subclique h is at most i. During this time, only these i clients can still have requests waiting to be serviced, so the average completion time increases by at most $\frac{i^2}{k}$. Thus, we can upper bound the cost of the optimal algorithm as follows:

$$\mathcal{OPT}(I) \leq \frac{1}{k}\sum_{i=1}^{k} i^2 p_i \tag{1}$$

where p_i is the number of cover graphs with exactly i clients with h-length greater than 0 for $1 \leq h \leq N$.

We will upper bound inequality (1) by upper bounding p_i for $i = 1, ..., k$. First, we give an upper bound of the total number of requests. Consider the time just after the last planting operation. At most $N-1$ requests have been serviced by the on-line algorithm $\mathcal{A}$ at that time. Each client never changes subclique after that. Hence, each client has at most k unserviced requests. Therefore, there are at most k^2 unserviced requests. Thus, there are at most $N-1+k^2$ requests.

Let R_i be the total number of requests in a subclique with i clients. The value of R_i is given in Lemma 2.6. Since there are at most $N-1+k^2$ requests in I, this implies $\sum_{i=1}^{k} R_i p_i \leq N-1+k^2$. Thus $\mathcal{OPT}$(I) is upper bounded by the maximization of

$$\frac{1}{k}\sum_{i=1}^{k} i^2 p_i \qquad \text{subject to} \qquad \sum_{i=1}^{k} R_i p_i \leq N-1+k^2.$$

Since i^2 grows faster than $\frac{i}{2}\lg i + i$, and from Lemma A.2, $\frac{i}{2}\lg i + i$ grows at least as fast as

R_i, then our upper bound on $\mathcal{OPT}$(I) is maximized when $p_i = 0$ for $i = 1, ..., k-1$ and

$$p_k = \frac{N-1+k^2}{R_k}.$$

$$\text{Thus,} \qquad \mathcal{OPT}(I) \leq \frac{1}{k}\sum_{i=1}^{k} i^2 p_i = \frac{1}{k} k^2 p_k = k\,\frac{N-1+k^2}{R_k}$$

$$\leq \begin{cases} \frac{N-1+k^2}{\frac{1}{2}\lg k+1} & \text{if } k \text{ is a power of 2} \\ \frac{N-1+k^2}{\frac{1}{2}\lg k} & \text{otherwise.} \end{cases}$$

Dividing the on-line cost by the offline cost, we get the final result. □

Corollary 2.3. *For the average completion time cost function, any on-line algorithm $\mathcal{A}$, and any integer $N \geq 2$,*

$$c_{\mathcal{A}}^{ACT} \geq \begin{cases} (\frac{\lg k}{2}+1)(\frac{2N+k-1}{2N+2k^2-2}) & \textit{if } k \textit{ is a power of 2} \\ (\frac{\lg k}{2})(\frac{2N+k-1}{2N+2k^2-2}) & \textit{otherwise} \end{cases}$$

on the K_{N+k-1} metric space.

Proof. Nodes are assigned to the subcliques on demand. A new node is needed when the adversary plants a client. The adversary plants k clients in the initial step. The only other place the adversary plants a client is in step (2.b.ii). The adversary can plant at most 1 client after each request is serviced. The adversary plants no more clients when $\mathcal{A}$ has serviced more than $N-1$ requests. Therefore, the adversary does at most $k+N-1$ plantings. Hence, $k+N-1$ nodes are needed for placing requests.

Since $N \geq 2$, then $k+N-1 \geq k+1$. Thus, other than the k nodes on which the adversary makes k initial plantings, there is 1 other node that can be used as the initial position of the server. □

Corollary 2.4. *For the average completion time cost function, any on-line algorithm $\mathcal{A}$, and any $\varepsilon > 0$, there exists an integer M such that,*

$$c_{\mathcal{A}}^{ACT} \geq \begin{cases} \frac{\lg k}{2}+1-\varepsilon & \textit{if } k \textit{ is a power of 2} \\ \frac{\lg k}{2}-\varepsilon & \textit{otherwise} \end{cases}$$

on the K_M metric space.

Proof. From Corollary 2.3, by choosing N large enough and setting $M = N+k-1$, the result follows. □

2.2.5 General lower bound on the line

In this section, we show how most lower bound results on cliques can be transformed into lower bound results on lines. In particular, any lower bound result on K_l using a finite adversary strategy can be extended to hold on $line(2l-1)$ where a finite adversary strategy is one in which the maximum number of requests ever generated by the adversary is upper bounded by some known constant.

Theorem 2.5. *Suppose we have a lower bound of c for either the total distance or average completion time cost functions on the K_l metric space for $l \geq 1$ as a result of a finite adversary strategy. Then this implies there exists a $c - \varepsilon$ lower bound for the same cost function on the $line(2l-1)$ metric space as the result of another finite adversary strategy.*

Proof. Let A denote an arbitrary finite adversary strategy on K_l that never has a client repeat a request in the same position. Let M denote the known upper bound on the number of requests A might generate. We show how to create a new finite adversary strategy A' on $line(2l-1)$ with essentially the same lower bound. We assume that the nodes in K_l are numbered from 0 to $l-1$ and that the left end of $line(2l-1)$ is node 0 and the right end of $line(2l-1)$ is node $2l-1$. The key idea is that whenever adversary strategy A requests node i in K_l, adversary strategy A' will request a cluster of m requests alternating between nodes $2i$ and $2i+1$ on $line(2l-1)$ where $m \gg M$. We will define m more precisely later. This leads to the following behavior for adversary A'. Whenever the im^{th} request of any client is serviced where $i \geq 0$, adversary A' reveals that this client either has no more requests or A' reveals the next m requests of this client. Whenever any of the other requests of any client are serviced, the adversary reveals no new information about this client.

The first key observation is that when considering input instance I', we can without loss of generality consider only algorithms which essentially service requests of clients in clusters of m. More precisely, we can consider algorithms which, after servicing a cluster of m requests from one client, move directly to service a cluster of m requests from a second client in uninterrupted fashion. The reason follows from a standard interchange argument. No new information will be revealed to the on-line algorithm until it services an entire cluster of m requests from a client. An algorithm which services some but not all requests in a cluster from one client and then moves on to service a cluster of m requests from a second client can be improved by moving directly to the second client and servicing its cluster of m requests first.

The only exception to this observation is the servicing of requests that occurs when moving from one client's cluster of m requests to another client's cluster of m requests. For example, if we service node 0 and then node $l-1$ of K_l, on $line(2l-1)$, this corresponds to alternately servicing nodes 0 and 1 a total of m times and then moving on to alternately servicing nodes $2l-2$ and $2l-1$ a total of m times. However, in the process of moving from node 1 to node $2l-2$, the algorithm will incidentally service any requests on nodes 2 through $2l-3$ once each. Since $m \gg M$ which is the maximum number of requests that the adversary will generate, these incidental servicings of individual requests will never result in a cluster of m requests being incidentally serviced. Thus, the interchange argument still

holds and we can focus on on-line algorithms which, after servicing a cluster of m requests from one client, move directly to another client's cluster of m requests and services that cluster to completion.

We now focus on the cost of servicing a cluster of m requests of some client in I'. There are two components to this cost. The first is the transition cost of moving from the previous location to the location of the first request of this block of m requests. This cost is lower bounded by 1 (the requests cannot be colocated) and upper bounded by $2l-1$. The second component is the cost of actually servicing this block of m requests. This cost is lower bounded by $m-1-M$ and upper bounded by $m-1$. The lower bound results from the fact that at most M requests of a cluster of m requests may have been incidentally served. Thus the total cost of servicing this cluster of m requests ranges between $m-M$ and $m+2l-2$. The key observation is that if $m \gg l$ and $m \gg M$, then this cost is essentially m. This returns us to the clique metric space where the cost to service any request is always 1. Since we can make m arbitrarily large with respect to l and M, the result follows. □

Corollary 2.5. *For the total distance cost function, any on-line algorithm $\mathcal{A}$, and any $\varepsilon > 0$,*

$$c_{\mathcal{A}}^{D} \geq \begin{cases} \frac{\lg k}{2} + 1 - \varepsilon & \text{if } k \text{ is a power of 2} \\ \frac{\lg k}{2} - \varepsilon & \text{otherwise} \end{cases}$$

on the line$(2k-1)$ metric space.

Proof. This follows from Theorems 2.3 and 2.5. □

Corollary 2.6. *For the average completion time cost function, any on-line algorithm $\mathcal{A}$, any integer $N \geq 2$, and any $\varepsilon > 0$,*

$$c_{\mathcal{A}}^{ACT} \geq \begin{cases} (\frac{\lg k}{2} + 1)(\frac{2N+k-1}{2N+2k^2-2}) - \varepsilon & \text{if } k \text{ is a power of 2} \\ (\frac{\lg k}{2})(\frac{2N+k-1}{2N+2k^2-2}) - \varepsilon & \text{otherwise} \end{cases}$$

on the line$(2N+2k-3)$ metric space.

Proof. This follows from Corollary 2.3 and Theorem 2.5. □

Corollary 2.7. *For the average completion time cost function, any on-line algorithm $\mathcal{A}$, and any $\varepsilon' > 0$, there exists integer N' such that,*

$$c_{\mathcal{A}}^{ACT} \geq \begin{cases} \frac{\lg k}{2} + 1 - \varepsilon' & \text{if } k \text{ is a power of 2} \\ \frac{\lg k}{2} - \varepsilon' & \text{otherwise} \end{cases}$$

on the line(N') metric space.

Proof. From Corollary 2.6, by choosing $\varepsilon < \varepsilon'$, choosing N large enough, and setting $N' = 2N+2k-3$, the result follows. □

2.3 Randomized Lower Bounds

Theorem 2.6. *For the total distance cost function, no randomized on-line algorithm $\mathcal{R}$ has a competitive ratio lower than $1 + \Theta(\lg \lg k)$ on the K_{k+1} metric space.*

Proof. We use Yao's [32] method of deriving a lower bound on the performance of any randomized algorithm on a worst-case input instance by deriving a lower bound on the expected performance of any deterministic algorithm against a specific probability distribution of input instances. Let the input distribution ρ be the uniform distribution over the set of all input instances generated by the adversary $A(1, k)$ defined in Section 2.2.1 where k is a power of 2.

Let $\mathcal{A}$ be any deterministic on-line algorithm. For any input instance I, we divide the cost $\mathcal{A}^D(I)$ among the k clients as follows. Whenever the server visits a node, the cost of the visit is charged to the longest client with an outstanding request on that node. We denote the cost charged to client i as $X_i(I)$. The cost of servicing the longest client is always k since no client subsumes it.

We now calculate $\mathbf{E}[X_i(\rho)]$. As noted previously, if i is the longest client, then $\mathbf{E}[X_i(\rho)] = k$. Otherwise, consider a client i with l requests that is subsumed by c clients. By definition, each of the c clients that subsume i must have at least l requests. Let $F(i, l)$ denote the event that the lth request of client i is served before the lth request of any of the c clients that subsume i. If $F(i, l)$ occurs, the cost of servicing client i is l. We lower bound $\mathbf{E}[X_i(\rho)]$ by l times the probability that $F(i, l)$ occurs. This probability is $\frac{1}{c+1}$ because these $c + 1$ clients are indistinguishable to the on-line algorithm until the lth request of any of these clients is serviced. Thus, we get the following lower bound on the expected cost of serving client i.

$$\mathbf{E}[X_i(\rho)] \geq \frac{l}{c+1} \tag{2}$$

The number of clients of length l that are subsumed by exactly c clients is the number of nodes at depth c with exactly l descendants in a binomial tree of size k, and this number is $\binom{\lg k - 1 - \lg l}{c-1}$.

Combining the fact that the number of clients with length $l = 2^g$ and that are subsumed by c clients is $\binom{\lg k - 1 - g}{c-1}$ with Equation 2 and the fact that $A^D(I) = \sum_{i=1}^{k} X_i(I)$, we get the

following.

$$\begin{aligned}
\mathbf{E}[A^D(\rho)] &\geq k + \sum_{g=0}^{\lg k-1} 2^g \sum_{c=1}^{\lg k-g} \binom{\lg k - 1 - g}{c-1} \frac{1}{c+1} \\
&= k + \sum_{g=0}^{\lg k-1} 2^g \frac{(\lg k - 1 - g)2^{\lg k-g} + 1}{(\lg k - g)(\lg k - g + 1)} \\
&\geq k + k \sum_{g=0}^{\lg k-1} \frac{(\lg k - 1 - g)}{(\lg k - g)(\lg k - g + 1)} \\
&= k + k \sum_{g=1}^{\lg k-1} \frac{g}{(g+1)(g+2)} \\
&= k + \Theta(k \lg\lg k)
\end{aligned}$$

The optimal solution has cost k. Therefore it follows that no randomized on-line algorithm has a competitive ratio better than $1 + \Theta(\lg\lg k)$. □

This lower bound is optimal for this input distribution. Consider the algorithm $\mathcal{RSEQ}$, which randomly selects a client and services it to completion. The cost of servicing client i will be the length of i if $\mathcal{RSEQ}$ chooses i before it chooses any of the clients that subsume i, and 0 otherwise. If a client i is subsumed by c clients, the probability that i will be chosen before any of the clients that subsume it is $\frac{1}{c+1}$. It can be shown that the expected cost of $\mathcal{RSEQ}$ is

$$\begin{aligned}
\mathbf{E}[\mathcal{RSEQ}^D(\rho)] &= k + \sum_{i=0}^{\lg k-1} 2^i \frac{(\lg k - 1 - i)2^{\lg k-i} + 1}{(\lg k - i)(\lg k - i + 1)} \\
&= k + \Theta(k \lg\lg k)
\end{aligned}$$

2.4 Lower Bound On the Line when $k = 2$.

In this section, we improve our general lower bounds of $\frac{\lg k}{2}$ for total distance and average completion time for the case when $k = 2$ and the metric space is $line(\infty)$. We refer to the two clients as client A and client B, and we label the requests from client A as a_i and the requests from client B as b_i.

Theorem 2.7. *For the total distance cost function, no on-line algorithm is* $(\frac{25}{9} - \epsilon)$*-competitive for the 2-client problem on* $line(\infty)$ *for all* $\epsilon > 0$.

Proof. We consider an adversary $V(\delta)$ that constructs inputs of the following form for $\delta > 0$. The server is initially located at the origin. The first request from A is located one unit to

the left of the origin. All the remaining requests from A except the last request will each be δ to the left of the previous one. The requests from B will be arranged to the right of the origin in a similar fashion. The final requests from both clients will be located at the same point, either δ to the left of the leftmost request, or δ to the right of the rightmost request. The input sequence where the final requests are both on the left is shown in Figure 5.

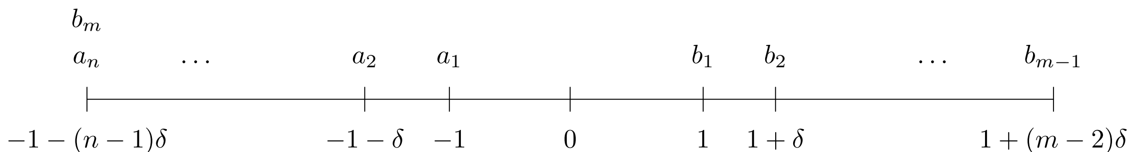

Figure 5: A lower bound example for total distance.

The optimal offline algorithm services these input sequences by first servicing all the requests of the client whose last request is on the opposite side of the origin of its previous requests, and then servicing the requests of the other client and the final request of the first client. In the example, the optimal offline algorithm first services B's requests and then services all of A's requests and B's final request.

First we show that any on-line algorithm which hopes to be better than 3-competitive against adversary $V(\delta)$ can be described by the following two infinite monotonically increasing sequences of finite numbers $\alpha_i(\delta)$ and $\beta_i(\delta)$ for $i \geq 1$ and $\delta > 0$. The term $\alpha_i(\delta)$ is the maximum distance the server is willing to move to the left after having previously changed directions $2(i-1)$ times when faced with adversary $V(\delta)$, and $\beta_i(\delta)$ is the maximum distance the server is willing to move to the right after having previously changed directions $2i-1$ times against adversary $V(\delta)$. Without loss of generality, we assume the server initially goes to the left to service client A. Suppose $\alpha_1(\delta)$ is unbounded; that is, the on-line algorithm will move the server to the left until there are no more requests to the left. In this case, the adversary will choose the input instance to be one where the final request of client A is at position $-x$ for a large value of x and client B will have only two requests at positions 1 and $-x$. The on-line cost will be $3x+2$ while the offline cost will be $x+2$ giving a competitive ratio of $3-\frac{4}{x+2}$. Therefore $\alpha_1(\delta)$ must be bounded to have a ratio better than 3. This argument generalizes which means that for an on-line algorithm which hopes to be better than 3-competitive, all $\alpha_i(\delta)$ and $\beta_i(\delta)$ must be bounded for all $i \geq 1, \delta > 0$. Note that $\alpha_1(\delta), \beta_1(\delta) \geq 1$ since the server must reach the first request of client A or B before changing direction. We will typically just use the notation α_i and β_i when δ is unambiguous.

We now define the set of input instances $I(n, A, \delta)$ for $n \geq 2$ and $I(n, B, \delta)$ for $n \geq 1$ based on an on-line algorithm's $\alpha_i(\delta)$ and $\beta_i(\delta)$ values. Input instance $I(n, A, \delta)$ for $n \geq 2$ is as follows: client A's requests appear progressively from position -1 to position $-\alpha_n$ while client B's requests appear progressively from position 1 to position $\beta_{n-1}+\delta$ followed by a final request at position $-\alpha_n$. Input instance $I(n, B, \delta)$ for $n \geq 1$ is as follows: client B's requests appear progressively from position 1 to position β_n while client A's requests appear progressively from position -1 to position $-\alpha_n-\delta$ followed by a final request at position

β_n. Figure 6 illustrates both input instance $I(n, A, \delta)$ as well as how the on-line algorithm behaves on $I(n, A, \delta)$. We will typically refer to these input instances as $I(n, A)$ and $I(n, B)$ when δ is unambiguous.

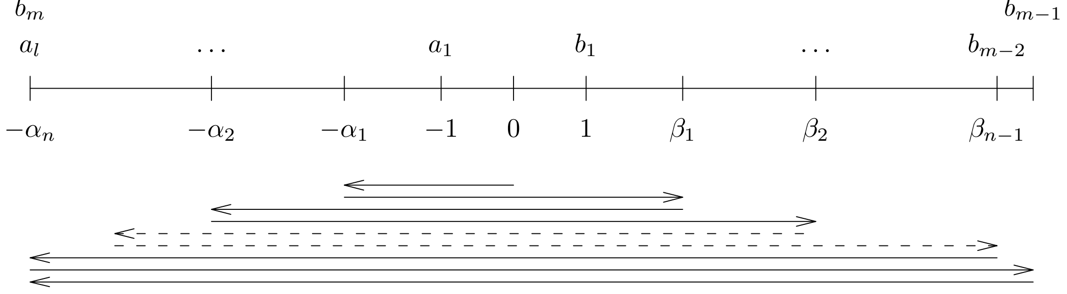

Figure 6: Input Instance $I(n, A, \delta)$

The optimal solution for input instance $I(n, A)$ for $n \geq 2$ is to first go to position $\beta_{n-1} + \delta$ and then to position $-\alpha_n$ for a total cost of $\alpha_n + 2\beta_{n-1} + 2\delta$. Meanwhile, the on-line cost is

$$\mathcal{A}^D = 2\sum_{i=1}^{n-1} \alpha_i + 3\alpha_n + 2\sum_{i=1}^{n-2} \beta_i + 4\beta_{n-1} + 2\delta.$$

Dividing these two terms gives us the following inequality.
(3)

$$c_{\mathcal{A}}^D \geq 2 + \frac{\alpha_n(\delta) + 2\sum_{i=1}^{n-1} \alpha_i(\delta) + 2\sum_{i=1}^{n-2} \beta_i(\delta)}{\alpha_n(\delta) + 2\beta_{n-1}(\delta) + 2\delta} - \frac{2\delta}{\alpha_n(\delta) + 2\beta_{n-1}(\delta) + 2\delta} \quad \text{for } n \geq 2, \delta > 0.$$

Since we can make δ arbitrarily close to 0 and since $\alpha_i(\delta), \beta_i(\delta) \geq 1$ independent of δ, we can simplify the above inequality as follows.

(4)
$$c_{\mathcal{A}}^D \geq 2 + \frac{\alpha_n + 2\sum_{i=1}^{n-1} \alpha_i + 2\sum_{i=1}^{n-2} \beta_i}{\alpha_n + 2\beta_{n-1}} \quad \text{for } n \geq 2.$$

Similarly, we can analyze input instance $I(n, B)$ for $n \geq 1$ to derive a lower bound of

(5)
$$c_{\mathcal{A}}^D \geq 2 + \frac{\beta_n + 2\sum_{i=1}^{n-1} (\alpha_i + \beta_i)}{\beta_n + 2\alpha_n} \quad \text{for } n \geq 1$$

on the competitive ratio of the on-line algorithm.

In order to show that equations 3 and 4 are equal, we define a set of values λ_i and γ_i as follows:

$$\begin{aligned} \text{For odd } i \geq 1, \lambda_i &= \alpha_{\lceil i/2 \rceil} \\ \text{For even } i \geq 2, \lambda_i &= \beta_{i/2} \\ \text{For } i \geq 0, \gamma_i &= \sum_{j=1}^{i} \lambda_j \end{aligned}$$

This allows us to combine inequalities (3) and (4) into the following inequality

$$c_{\mathcal{A}}^{D} \geq 2 + \frac{\lambda_n + 2\gamma_{n-2}}{\lambda_n + 2\lambda_{n-1}} \quad \text{for } n \geq 2$$

which can be rewritten as $2 + c_n$ where

$$c_n = \frac{\lambda_n + 2\gamma_{n-2}}{\lambda_n + 2\lambda_{n-1}} \quad \text{for } n \geq 2. \tag{6}$$

We now use the property that λ_i is finite for all $i \geq 1$ to define two important quantities of any sequence of finite values $\langle \lambda_i \rangle$. First, we define $X(\langle \lambda_i \rangle) = \sup_{n \geq 2} c_n$ where c_n is defined in equation (5). Next we define $R(\langle \lambda_i \rangle) = \inf_{n \geq 2} \langle \frac{\lambda_n}{\gamma_{n-1}} \rangle$. For the remainder of this proof, we will use X to denote $X(\langle \lambda_i \rangle)$, and we will use R to denote $R(\langle \lambda_i \rangle)$.

To derive the best possible lower bound on the competitive ratio of any on-line algorithm using this set of input instances, we need to find the set of values $\langle \lambda_i \rangle$ that minimize $2 + X$. We will show that the minimum possible value of X is 7/9 which is achieved when $\frac{\lambda_n}{\gamma_{n-1}} = 3$ for all $n \geq 2$; that is, when $R = 3$.

We first observe that

$$\forall n \geq 2 \quad \lambda_n \leq \left(\frac{2X}{1-X} \right) \lambda_{n-1} - \left(\frac{2}{1-X} \right) \gamma_{n-2}. \tag{7}$$

This follows from equation 5 and the definition of X.

We next observe that

$$\forall n \geq 3 \quad \lambda_n \geq R(R+1)\gamma_{n-2}. \tag{8}$$

This can be derived as follows given the definitions of R and γ_n for $n \geq 1$.

$$\lambda_n \geq R\gamma_{n-1} = R(\lambda_{n-1} + \gamma_{n-2}) \geq R(R\gamma_{n-2} + \gamma_{n-2}) = R(R+1)\gamma_{n-2}.$$

Applying the transitive property to inequalities (6) and (7), we derive the following inequality:

$$\forall n \geq 3 \quad R(R+1)\gamma_{n-2} \leq \left(\frac{2X}{1-X} \right) \lambda_{n-1} - \left(\frac{2}{1-X} \right) \gamma_{n-2}$$

which can be rewritten as

$$\forall n \geq 2 \quad \frac{\lambda_n}{\gamma_{n-1}} \geq \left(R(R+1) + \frac{2}{1-X} \right) \left(\frac{1-X}{2X} \right). \tag{9}$$

Combining inequality (8) and the definition of R as the infimum of $\langle \frac{\lambda_n}{\gamma_{n-1}} \rangle$ for $n \geq 2$, we obtain

$$R \geq \left(R(R+1) + \frac{2}{1-X} \right) \left(\frac{1-X}{2X} \right)$$

which can be rewritten as

$$R^2(1-X) + R(1-3X) + 2 \leq 0. \tag{10}$$

Clearly R must be real. We apply the quadratic equation to find the minimum value of X that results in R being real.

$$(1-3X)^2 - 4(1-X)2 \geq 0$$

The minimum value of $X = 7/9$ when $R = 3$. Therefore, we achieve the desired lower bound of $2 + 7/9 = 25/9$ on the competitive ratio of any on-line algorithm for the 2-client problem on *line*(∞). □

Theorem 2.8. *For the average completion time cost function, no on-line algorithm is* $(3-\epsilon)$-*competitive for the 2-client problem on line*(∞) *for all* $\epsilon > 0$.

Proof. The proof uses an adversary $V'(\delta)$ for $\delta > 0$ that constructs inputs similar to those used in the proof of Theorem 2.7. The only difference is that the final request of client A is always on the left and the final request of client B is always on the right.

As in Theorem 2.7, we can argue that any on-line algorithm which hopes to be $(3-\epsilon)$-*competitive* against $V'(\delta)$ can be described by two infinite monotonically increasing sequences of finite numbers $\alpha_i(\delta)$ and $\beta_i(\delta)$ for $i \geq 1$ and $\delta > 0$ where $\alpha_i(\delta)$ is the maximum distance the server is willing to move to the left after having changed directions $2(i-1)$ times and $\beta_i(\delta)$ is the maximum distance the server is willing to move to the right after having changed directions $2i-1$ times. We again use just α_i and β_i when δ is unambiguous.

The remainder of this proof, however, is much simpler as we focus only on an input instance I constructed by the adversary which is defined entirely by α_1 and β_1. We will show that the best an on-line algorithm can do is to make $\alpha_1 = \beta_1$ in which case the on-line algorithm will be 3-competitive at best. For any given α_1 and β_1, define I as follows. Requests from A appear progressively from position -1 to position $-\alpha_1 - \delta$. Requests from B appear progressively from position 1 to position $\beta + \delta$.

Clearly, the on-line algorithm will move left to position $-\alpha_1$, move right to position β_1, move left to position $-\alpha_1 - \delta$ (completing client A), and then move right to position $\beta_1 + \delta$ (completing client B). Thus, the average completion time incurred by the on-line algorithm on input instance I is $\frac{1}{2}(7\alpha_1 + 5\beta_1 + 4\delta)$. Meanwhile, $\mathcal{OPT}$ services the shorter client first which means $\mathcal{OPT}^{\text{ACT}}(I) = \frac{1}{2}\min(3\alpha_1 + \beta_1 + 4\delta, \alpha_1 + 3\beta_1 + 4\delta)$. Note, we can make δ arbitrarily small, so this implies a lower bound of $\frac{7\alpha_1+5\beta_1}{\min(3\alpha_1+\beta_1,\alpha_1+3\beta_1)}$ on the competitive ratio of the on-line algorithm.

We now show that this lower bound is minimized to be 3 when $\alpha_1 = \beta_1$. There are 2 cases to consider.

Case 1: $\alpha_1 \leq \beta_1$

In this case, $\mathcal{OPT}^{\text{ACT}}(I) = \frac{1}{2}(3\alpha_1 + \beta_1)$ which means that the lower bound on the competitive ratio of the on-line algorithm is $\frac{7\alpha_1+5\beta_1}{3\alpha_1+\beta_1}$ which is strictly larger than 3 when $\alpha_1 < \beta_1$ and which equals 3 when $\alpha_1 = \beta_1$.

Case 2: $\beta_1 \leq \alpha_1$
In this case, $\mathcal{OPT}^{\text{ACT}}(I) = \frac{1}{2}(\alpha_1 + 3\beta_1)$ which means that the lower bound on the competitive ratio of the on-line algorithm is $\frac{7\alpha_1+5\beta_1}{\alpha_1+3\beta_1}$ which is strictly larger than 3 when $\beta_1 < \alpha_1$ and which equals 3 when $\alpha_1 = \beta_1$.

Thus the result follows. □

2.5 Lower Bound On the Clique when $k = 2$.

In this section, we improve our general lower bounds of $\frac{\lg k}{2}$ for the case where $k = 2$ on K_∞.

Theorem 2.9. *For the total distance cost metric, no on-line algorithm is $(\frac{9}{5} - \epsilon)$-competitive for the 2-client problem on K_∞ for all $\epsilon > 0$.*

Proof. Label the vertices with positive integers. Let D be a large integer. The adversary operates in rounds. At the beginning of round 1, a client is called the *leader*, and its initial request is on node $D + 1$. The other client is called the *follower*, and its initial request is on node 1. Subsequent requests are generated according to the following rule. If the current request of both clients are on different nodes, then when the request, say on node x, of either of the clients is serviced by the on-line algorithm, the next request of that client is on node $x + 1$. If the current request of both clients are on the same nodes, say on node x, then after these two requests are serviced, the new round begins. In the new round, the leader becomes the follower, and its next request is on node 1. The follower becomes the leader, and its next request is on node $x + 1$.

Suppose the adversary stops just after round n. The optimal solution is to service the client that is the follower in round n picking up requests of the other client along the way.

We show that any on-line algorithm which hopes to be better than 2-competitive on this form of input sequence can be described by an infinite monotonically increasing sequence of finite numbers α_i for $i \geq 1$ where $\alpha_1 = x_1/D - 1$, $\alpha_i = x_i/D$ for $i \geq 2$, and x_i is the last node on which the clients place their last requests in round i. Suppose α_1 is unbounded; that is, at any time, if the current request of the follower and the leader are on node x and y respectively, then $x < y$. In this case, the adversary will choose the input instance to be one where the final request of both clients is on node xD for a large integer x. The on-line cost will be $xD + xD - D$ while the offline cost will be xD giving a competitive ratio of $2 - \frac{1}{x}$. Therefore, α_1 must be bounded to have a ratio better than 2. Clearly, this argument generalizes which means that all α_i must be bounded for all $i \geq 1$.

Definition 2.13. *For $n \geq 1$, let o_n be the sum of α_i where i is an odd number between 1 and n inclusively.*

Definition 2.14. *For $n \geq 1$, let e_n be the sum of α_i where i is an even number between 1 and n inclusively.*

Definition 2.15. *Let $\lambda_0 = 0$. For $n \geq 1$, let $\lambda_n = \alpha_n - \lambda_{n-1}$.*

Definition 2.16. *For $n \geq 1$, let*

$$\gamma_n = \begin{cases} o_n & n \text{ is odd} \\ e_n & n \text{ is even} \end{cases}$$

Fact 2.5. *For $n \geq 0, \lambda_n \geq 0$.*

This is true because α_n's is a non-decreasing sequence.

Fact 2.6. *For $n \geq 1$,*

$$\lambda_n = \begin{cases} o_n - e_n = o_n - e_{n-1} & n \text{ is odd} \\ e_n - o_n = e_n - o_{n-1} & n \text{ is even} \end{cases}$$

Fact 2.7. *For $n \geq 3, \gamma_n = \gamma_{n-2} + \alpha_n$.*

This is true by Definition 2.16, 2.13, and 2.14.

Fact 2.8. *For $n \geq 2, \gamma_n = \gamma_{n-1} + \lambda_n$.*

This is true by Definition 2.16 and Fact 2.6.

The cost of the on-line algorithm is,

$$\begin{aligned} \forall n \geq 2 \qquad A(I_n) &= (\gamma_n + \gamma_{n-1} + \alpha_n)D - n \\ &= (\gamma_n + \gamma_{n-1} + \lambda_{n-1} + \lambda_n)D - n \end{aligned}$$

The optimal cost is,

$$\begin{aligned} \forall n \geq 2 \qquad OPT(I_n) &= (\gamma_{n-1} + \alpha_n)D \\ &= (\gamma_{n-1} + \lambda_{n-1} + \lambda_n)D \end{aligned}$$

Thus the ratio of the cost of the on-line algorithm and the adversary is

$$\frac{(\gamma_{n-1} + \gamma_n + \lambda_{n-1} + \lambda_n)D - n}{(\gamma_{n-1} + \lambda_{n-1} + \lambda_n)D}.$$

If D is sufficiently larger, the ratio is arbitrarily close to

$$\frac{\gamma_{n-1} + \gamma_n + \lambda_{n-1} + \lambda_n}{\gamma_{n-1} + \lambda_{n-1} + \lambda_n} = \frac{2\gamma_{n-1} + \lambda_{n-1} + 2\lambda_n}{\gamma_{n-1} + \lambda_{n-1} + \lambda_n} = 2 - \frac{\lambda_{n-1}}{\gamma_{n-1} + \lambda_{n-1} + \lambda_n} \tag{11}$$

We can write the equation (10) as $2 - c_n$ where

$$c_n = \frac{\lambda_{n-1}}{\gamma_{n-1} + \lambda_{n-1} + \lambda_n} \tag{12}$$

We now use the property that λ_i is finite for all $i \geq 1$ to define two important quantities of any sequence of finite values $\langle \lambda_i \rangle$. First, we define $X(\langle \lambda_i \rangle) = \sup_{n \geq 2} c_n$ where c_n is defined

in equation (11). Note that $0 < X(\langle\lambda_i\rangle) < 1$. For the remainder of this proof, we will use X to denote $X(\langle\lambda_i\rangle)$. Next we define $R(\langle\lambda_i\rangle) = \inf_{n\geq 2} \frac{\lambda_n}{\gamma_{n-1}}$, and we will use R to denote $R(\langle\lambda_i\rangle)$.

To derive the best possible lower bound on the competitive ratio of any on-line algorithm against this adversary, we need to find the set of values $\langle\lambda_i\rangle$ that minimize $2 - X$. We will show that the maximum possible value of X is $1/5$ which is achieved when $\frac{\lambda_n}{\gamma_{n-1}} = 1$ for all $n \geq 2$; that is, when $R = 1$.

We first observe that

$$\forall n \geq 2 \quad \lambda_n \leq \frac{1-X}{X}\lambda_{n-1} - \gamma_{n-1}. \tag{13}$$

This follows from equation (11) and the definition of X.

We next observe that

$$\forall n \geq 3 \quad \lambda_n \geq R(R+1)\gamma_{n-2}. \tag{14}$$

This can be derived as follows given the definitions of R and γ_n for $n \geq 1$.

$$\lambda_n \geq R\gamma_{n-1} \geq R(\lambda_{n-1} + \gamma_{n-2}) \geq R(R\gamma_{n-2} + \gamma_{n-2}) = R(R+1)\gamma_{n-2}.$$

Applying the transitive property to inequalities (12) and (13), we derive that following inequality:

$$\forall n \geq 3 \qquad R(R+1)\gamma_{n-2} \leq \frac{1-X}{X}\lambda_{n-1} - \gamma_{n-1}$$

which can be rewritten as

$$\forall n \geq 2 \qquad \frac{\lambda_n}{\gamma_{n-1}} \geq \frac{X}{1-2X}(R^2 + R + 1) \tag{15}$$

Combining inequality (14) and the definition of R as the infimum of $\frac{\lambda_n}{\gamma_{n-1}}$ for $n \geq 2$, we obtain

$$R \geq \frac{X}{1-2X}(R^2 + R + 1)$$

which can be rewritten as

$$XR^2 + (3X-1)R + X \leq 0 \tag{16}$$

Since R must be real, we apply the quadratic equation and the constraint $0 < X < 1$ to find the maximum value of X that results in R being real.

$$(3X-1)^2 - 4 \cdot X \cdot X \geq 0$$

The maximum value of $X = 1/5$ when $R = 1$. Therefore, we achieve the desired lower bound of $2 - 1/5 = 9/5$ on the competitive ratio of any on-line algorithm for the 2-client problem on K_∞. □

Theorem 2.10. *For the average completion time cost metric, no on-line algorithm is $(\frac{9}{5}-\epsilon)$-competitive for the 2-client problem on K_∞ for all $\epsilon > 0$.*

Proof. The proof is the same as that of Theorem 2.9 because both clients complete at the same time in both the optimal offline schedule and the on-line schedule. □

3 Disk Scheduling Algorithms

In this section, we analyze in detail how commonly used disk scheduling algorithms perform on the $line(\infty)$ and $line(\Delta)$ metric spaces in a multithreaded environment. In particular, we study how the performance of these algorithms is related to the number of threads in the system. We prove that the elevator algorithm has a competitive ratio which is independent of the number of threads in the system. On the other hand, all the greedy algorithms have competitive ratios of $2k-1$. This helps to explain why elevator type algorithms have superior performance in heavily loaded systems. We close this section by describing how lower bound results on cliques can be translated into lower bound results on lines.

While we are mostly interested in the $line(\Delta)$ metric space since this most accurately represents the disk scheduling problem, we try to prove results which are as general as possible. Note that in the $line(\Delta)$ metric space, distinct requests of each client are at least distance 1 apart. Further note that any lower bounds of $\Omega(\Delta)$ for the $line(\Delta)$ metric space translate into unbounded lower bounds on the $line(\infty)$ metric space.

We first analyze $\mathcal{EL}$, and we only consider the $line(\Delta)$ and $line(\infty)$ metric spaces since $\mathcal{EL}$ is not well defined for other metric spaces.

Theorem 3.1. *On $line(\Delta)$, $\frac{\Delta}{2} \leq c_{\mathcal{EL}} \leq \Delta$ for the total distance cost function.*

Proof. To prove the upper bound on $\mathcal{EL}^D(I)$, we first observe that $\mathcal{OPT}^D(I) \geq n_{max}(I)$ because consecutive requests of each client are at least distance 1 apart. On the other hand, $\mathcal{EL}$ serves at least one request from each client on each sweep, and each sweep is no longer than Δ. Therefore $\mathcal{EL}^D(I) \leq n_{max}(I)\Delta$. Thus $c_{\mathcal{EL}} \leq \Delta$.

We now show that $c_{\mathcal{EL}} > \frac{\Delta}{2} - \epsilon$ for all $\epsilon > 0$. Consider the following input instance consisting of two clients where client A and client B both generate n requests where n is an even number. The server begins at position 0. The requests from client A alternate between points 1 and 0 with the first request at position 1. The requests from client B alternate between points Δ and $\Delta - 1$ with the first request at position Δ. $\mathcal{EL}$ will service one request from each client on each sweep. $\mathcal{OPT}$ will first service all requests from client A and then it will service the requests from client B. It is easy to see that $\mathcal{EL}^D(I) = n\Delta$ whereas $\mathcal{OPT}^D(I) = 2n - 1 + \Delta$. If we choose $n > \frac{\Delta^2-\Delta}{4\epsilon} - \frac{\Delta-1}{2}$, then $c_{\mathcal{EL}} > \frac{\Delta}{2} - \epsilon$. □

Theorem 3.2. *On $line(\Delta)$, $\frac{2\Delta}{3} \leq c_{\mathcal{EL}} \leq \Delta$ for the average completion time cost function.*

Proof. To prove the upper bound on $\mathcal{EL}^{\mathrm{ACT}}(I)$, we first observe that $\mathcal{OPT}^{\mathrm{ACT}}(I) \geq \frac{n(I)}{k}$ because consecutive requests of each client are at least 1 unit apart. On the other hand,

$\mathcal{EL}$ serves at least one request from client i on each sweep and there are $n_i(I)$ requests from client i. So the completion time of client i is at most $\Delta n_i(I)$. Thus,

$$\mathcal{EL}^{\text{ACT}}(I) \leq \frac{\sum_{i=1}^{k} \Delta n_i(I)}{k} \leq \Delta \frac{n(I)}{k} \leq \Delta \mathcal{OPT}^{\text{ACT}}(I).$$

The lower bound follows from considering the same lower bound input instance used for $\mathcal{EL}$ when the cost function is total distance. It is easy to see that $\mathcal{EL}^{\text{ACT}}(I) = \frac{n\Delta+(n-1)\Delta}{2}$ whereas $\mathcal{OPT}^{\text{ACT}}(I) = \frac{n+2n+\Delta}{2}$. If we choose $n > \frac{2\Delta^2+3\Delta}{9\epsilon} - \frac{\Delta}{3}$, then $c_{\mathcal{EL}} > \frac{2\Delta}{3} - \epsilon$. □

We now consider the three greedy algorithms, $\mathcal{SDF}$, $\mathcal{SEQ}$, and $\mathcal{MP}$, on any metric space and then focus on the $line(\Delta)$ metric space. For $\mathcal{SDF}$, Theorems 2.1 and 2.2 prove that the competitive ratio of $\mathcal{SDF}$ is exactly $2k-1$ for both cost functions for any metric space which contains $line(\infty)$. On $line(\Delta)$, $\mathcal{SDF}$ has only slightly better performance. We now prove similar results for both $\mathcal{SEQ}$ and $\mathcal{MP}$.

Theorem 3.3. *For the total distance cost function,* $c_{\mathcal{SEQ}} = 2k-1$.

Proof. We bound $\mathcal{SEQ}^D(I)$ by bounding the distance required to service each client of I. The cost of servicing client 1 is D_1. The cost of servicing client i for $2 \leq i \leq k$ is upperbounded by $\delta_{max} + D_i$ where the first term is an upper bound of the cost incurred in transitioning from client $i-1$ to client i. Therefore

$$\mathcal{SEQ}^D(I) \leq D_1 + \sum_{i=2}^{k} (\delta_{max} + D_i) \leq kD_{max} + (k-1)\delta_{max}.$$

Applying Fact 2.1, we observe that $\mathcal{SEQ}^D(I) \leq (2k-1)\mathcal{OPT}^D(I)$, so the competitive ratio of $\mathcal{SEQ}$ is at most $2k-1$. Again, this upper bound result extends to all metric spaces. Furthermore, this bound is tight for any metric space which includes $line(\infty)$ as can be seen by the same lower bound input instance for $\mathcal{SDF}$ (Figure 2). □

Theorem 3.4. *For the average completion time cost function,* $c_{\mathcal{SEQ}} = 2k-1$.

Proof. We bound $\mathcal{SEQ}^{\text{ACT}}(I)$ by bounding the completion time of each client of I. For $1 \leq i \leq k$, the completion time of client i is at most $D_i + \sum_{j=1}^{i-1} 2D_j$. This bound assumes a worst-case scenario where the server must service client 1, return to its initial position, then service client 2, return to its initial position, etc., before finally servicing client i. Thus, we

obtain the following upper bound on $\mathcal{SEQ}^{\text{ACT}}(I)$.

$$\begin{aligned}
\mathcal{SEQ}^{\text{ACT}}(I) &\leq \left(\frac{1}{k}\right)\left(\sum_{1\leq i\leq k}\left[\left(\sum_{1\leq j<i} 2D_j\right) + D_i\right]\right) \\
&= \left(\frac{1}{k}\right)\left(\left(\sum_{1\leq i\leq k}\sum_{1\leq j<i} 2D_j\right) + \sum_{1\leq i\leq k} D_i\right) \\
&= \left(\frac{1}{k}\right)\left(\left(\sum_{1\leq j\leq k}\sum_{j<i\leq k} 2D_j\right) + \sum_{1\leq i\leq k} D_i\right) \\
&= \left(\frac{1}{k}\right)\left(\sum_{1\leq j\leq k}(k-j)2D_j + \sum_{1\leq j\leq k} D_j\right) \\
&= \left(\frac{1}{k}\right)\left(\sum_{1\leq j\leq k}(2k-2j+1)D_j\right) \\
&\leq \left(\frac{1}{k}\right)\left(\sum_{1\leq j\leq k}(2k-1)D_j\right) \\
&= (2k-1)\left(\frac{D}{k}\right)
\end{aligned}$$

From Fact 2.2 we can conclude

$$\mathcal{SEQ}^{\text{ACT}}(I) \leq (2k-1)\mathcal{OPT}^{\text{ACT}}(I)$$

Again, this upper bound holds for any metric space, and this bound is tight in any metric space that includes $line(\infty)$ as can be seen by the same lower bound input instance for $\mathcal{SDF}$ (Figure 3). □

Theorem 3.5. *On $line(\infty)$, $c_{\mathcal{MP}} = 2k-1$ for the total distance cost function.*

Proof. We define the outlying requests to be the rightmost and leftmost requests in the system. On a line it is easy to see that $\mathcal{MP}$ will move the server towards the nearest outlying request.

Because the server's motion is determined only by the outlying requests, we will bound $\mathcal{MP}^D(I)$ by bounding the distance the server can move before the outlying requests change. The outlying requests will change whenever an outlying request is serviced or a new request appears that lies to the right or left of all other requests. Clearly this can only happen after some request $r_{i,j}$ has been serviced. There are three cases to consider when there is a change in the outlying request. The first case is that $j < n_i$ and $r_{i,j+1}$ is the new outlying request. The second case is $j < n_i$ and $r_{i,j+1}$ is not the new outlying request. The third case is that $j = n_i$, that is $r_{i,j}$ is a terminal request.

If $r_{i,j+1}$ is the new outlying request, then the nearest outlying request is at most a distance $d_{i,j}$ from the server, and the server cannot move farther than $d_{i,j}$ before the outlying requests change. This is true because if the server moves a distance of $d_{i,j}$ it will reach the nearest outlying request, at which time the outlying requests will change. It is worth noting that the server does not necessarily move towards $r_{i,j+1}$.

If $r_{i,j+1}$ is not the new outlying request, then $r_{i,j}$ must have been an outlying request, and the server is now to the right or the left of all requests. Therefore the nearest outlying request is the nearest request, and is at most a distance $d_{i,j}$ from the server. As in the first case, this means that the server cannot move farther than $d_{i,j}$ before the outlying requests change.

If $j = n_i$, then the new outlying request can be any outstanding request. However the nearest outlying request cannot be farther than δ_{max} from the server, and the server cannot move farther than δ_{max} before the outlying requests change again.

Because changes in the outlying requests happen at most once for each request that is serviced, the total cost from the first two cases is at most $\sum_{i=1}^{k} \sum_{j=0}^{n_i-1} d_{i,j} = \sum_{i=1}^{k} D_i$. The total cost from the third case is at most $(k-1)\delta_{max}$. Therefore

$$\begin{aligned} \mathcal{MP}^D(I) &\leq \sum_{i=1}^{k} D_i + (k-1)\delta_{max} \\ &\leq kD_{max} + (k-1)\delta_{max} \\ &\leq (2k-1)\mathcal{OPT}^D(I) \end{aligned}$$

We show that this bound is tight by observing that $\mathcal{MP}$ will service the requests of the input instance in Figure 2 in the same order that $\mathcal{SDF}$ would. □

We do not have a similar upper bound proof for $\mathcal{MP}$ in an arbitrary metric space, but we note that in an arbitrary general metric space, $\mathcal{MP}$ has to solve a Traveling Salesman Problem of size k to determine each move. Given that $\mathcal{MP}$ can do no better than the $2k-1$ ratio achievable by the simple $\mathcal{SDF}$ algorithm, it does not seem worthwhile to implement this more complex algorithm.

Theorem 3.6. *On line(∞), $2k-1 \leq c_{\mathcal{MP}} \leq (k-1)(2k-1)$ for the average completion time cost function.*

Proof. When the cost function is total distance, the behavior of $\mathcal{MP}$ is determined only by the distance between the server and the outlying requests. When the cost function is average completion time, the number of requests on either side of the server is also a factor. Consider the extreme case where the nearest request is d_l units to the left of the server, and the remaining $k-1$ requests are d_r units to the right of the server. The cost of moving to the nearest request first is $(kd_l + (k-1)(d_l + d_r))/k$. The cost of moving to the farther requests first is $(kd_r + (d_l + d_r))/k$. If $\mathcal{MP}$ moves to the farther requests, then $(kd_r + (d_l + d_r))/k \leq (kd_l + (k-1)(d_l + d_r))/k$. This implies that $d_r \leq (k-1)d_l$. It follows that, on each move, $\mathcal{MP}$ will move at most $k-1$ times as far as the the nearest request.

We will use this fact and the argument used in the proof of Theorem 2.2 to conclude that the average completion time of $\mathcal{MP}$ is at most $(k-1)(2k-1)$ times the average completion time of $\mathcal{OPT}$.

Assume that the clients are labeled according to the order that $\mathcal{MP}$ will services their last requests. If a move starts from request $r_{i,j}$ where $j < n_j$, then the distance to the *nearest* request is at most $d_{i,j}$. The distance that $\mathcal{MP}$ will move is at most $(k-1)d_{i,j}$. Using an argument similar to that in the proof of Theorem 2.2, we can prove an upper bound on the cost of a move from request r_{i,n_i} to the *nearest* request of $(\frac{k-i}{k})(D_i + \min_{i<j\leq k} D_j)$. The cost of the move from request r_{i,n_i} incurred by $\mathcal{MP}$ is at most $(k-1)$ times this amount. The average completion time of $\mathcal{MP}$ can be computed the same way as in Theorem 2.2, and the upper bound follows.

The lower bound input instance is the same as the lower bound input instance for $\mathcal{SDF}$ (Figure 3). □

In Section 2 we showed that on $line(\infty)$ for the greedy algorithms $\mathcal{SDF}$ and $\mathcal{SEQ}$, that if the the cost function is total distance or average completion time, then $c^{\mathrm{CF}}_{\mathcal{A}} > 2k-1-\epsilon$ for all $\epsilon > 0$. We can use the same arguments to show that on $line(\Delta)$, $c^{\mathrm{CF}}_{\mathcal{A}} > 2k-1-\epsilon$ where ϵ is a function of Δ and k.

Corollary 3.1. *For the greedy algorithms $\mathcal{SDF}$ and $\mathcal{SEQ}$ on $line(\Delta)$, $c^{D}_{\mathcal{A}} \geq 2k-1-\frac{3k^2-5k+2}{\Delta+k-1}$ and $c^{ACT}_{\mathcal{A}} \geq 2k-1-\frac{2k^3+4k^2-10k+4}{2\Delta+k^2+k-2}$.*

Proof. For total distance consider the input instance described in Figure 2. Let the server's initial position be $k-1$ instead of the origin and shift the positions of the requests accordingly. For this instance, $\mathcal{OPT}^D = \Delta + k - 1$ and the total distance is:

$$\mathcal{A}^D = \Delta + \left(\sum_{i=1}^{k-1} 2(\Delta - k + i) + 1 \right) = (2k-1)\mathcal{OPT}^D - (3k^2 - 5k + 2)$$

For average completion time consider the input instance described in Figure 3. Let the server's initial position be $k-1$ and shift the positions of the requests accordingly. $\mathcal{OPT}^{\mathrm{ACT}} = \frac{2\Delta+k^2+k-2}{2k}$ and the average completion time is:

$$\mathcal{A}^{\mathrm{ACT}} = \Delta - k + 1 + \left(\sum_{i=1}^{k-1} 2(\Delta - k + 1) + i \right) = (2k-1)\mathcal{OPT}^{\mathrm{ACT}} - \frac{(k^3 + 2k^2 - 5k + 2)}{k}$$

□

Finally, we consider the $\mathcal{FCFS}$ algorithm. The only upper bounds we have are trivial upper bounds that apply to any algorithm which does not idle the server when requests are available for service. Note by idling we include changing the server's direction before reaching an available request and moving the server away from all available requests. Our lower bounds for the $\mathcal{FCFS}$ algorithm are derived from the same input instance used to derive lower bounds for the $\mathcal{EL}$ algorithm.

Lemma 3.1. *For any input instance I and any non-idling algorithm $\mathcal{A}$ on line(Δ), $c_{\mathcal{A}} \leq k\Delta$ for the total distance cost function.*

Proof. For any non-idling algorithm $\mathcal{A}$, $\mathcal{A}^D(I) \leq n(I)\Delta \leq kn_{max}(I)\Delta$. Because consecutive requests of each client must be at least distance 1 apart, $\mathcal{OPT}^D(I) \geq n_{max}(I)$. Thus the result follows. □

Corollary 3.2. *On line(Δ), $c^D_{\mathcal{FCFS}} \leq k\Delta$ and $c^{ACT}_{\mathcal{FCFS}} \leq k^2\Delta$.*

Proof. The total distance upper bound follows immediately from Lemma 3.1. The average completion time upper bound follows from Lemma 3.1 and Fact 2.3. □

Corollary 3.3. *On line(Δ), $c^D_{\mathcal{FCFS}} \geq \frac{\Delta}{2}$ and $c^{ACT}_{\mathcal{FCFS}} \geq \frac{2\Delta}{3}$.*

Proof. Both lower bound results follow from observing that $\mathcal{FCFS}$ will service the requests of the lower bound instance described in Theorem 3.1 in the same order that $\mathcal{EL}$ would. □

4 Maximum response time on a Line

In this section we analyze the maximum response time cost function. We first prove a general lower bound of of $\Omega(\sqrt[3]{\Delta})$ that applies to any algorithm which faces only two clients on *line*(Δ). This implies that no algorithm has a bounded competitive ratio on the continuous line metric space, *line*(∞). We also show that several greedy algorithms such as $\mathcal{SDF}$ and $\mathcal{SEQ}$ have unbounded competitive ratios, even on *line*(Δ). Finally, we show that algorithms such as $\mathcal{EL}$ and $\mathcal{FCFS}$ do have bounded competitive ratios on *line*(Δ).

4.1 Lower Bounds

Theorem 4.1. *On line(Δ) for $\Delta \geq 20$, no on-line algorithm is better than $\Omega(\sqrt[3]{\Delta})$-competitive for the maximum response time cost function, even when the input has only two clients.*

Proof. Let m be the smallest integer such that $m\lceil\sqrt{m}\,\rceil \geq \Delta/2$ and $m\lceil\sqrt{m}\,\rceil + m \leq \Delta$. From lemma A.3, m exists for $\Delta \geq 20$. Our lower bound input uses the line segment $[-m\lceil\sqrt{m}\,\rceil, m]$.

Our lower bound argument utilizes two symmetric input instances I_1 and I_2. In both I_1 and I_2, we will refer to the two clients as client A and client B, and we refer to the requests from client A as a_i and the requests from client B as b_i.

In input I_1, client A has a total of $3m-1$ requests. For $1 \leq i \leq m$, a_i is at point i. The final $2m-1$ requests alternate between the origin and point m, starting and ending with the origin. Client B has a total of $m + \lceil\sqrt{m}\,\rceil - 1$ requests. For $1 \leq i \leq m$, b_i is at point $-i$. For $1 \leq i \leq \lceil\sqrt{m}\,\rceil - 1$, request b_{m+i} is at point $-(i+1)m$. Input instance I_1 is depicted in Figure 7. Input I_2 is identical to input I_1 except client A generates requests to the left of the origin and client B generates requests to the right of the origin.

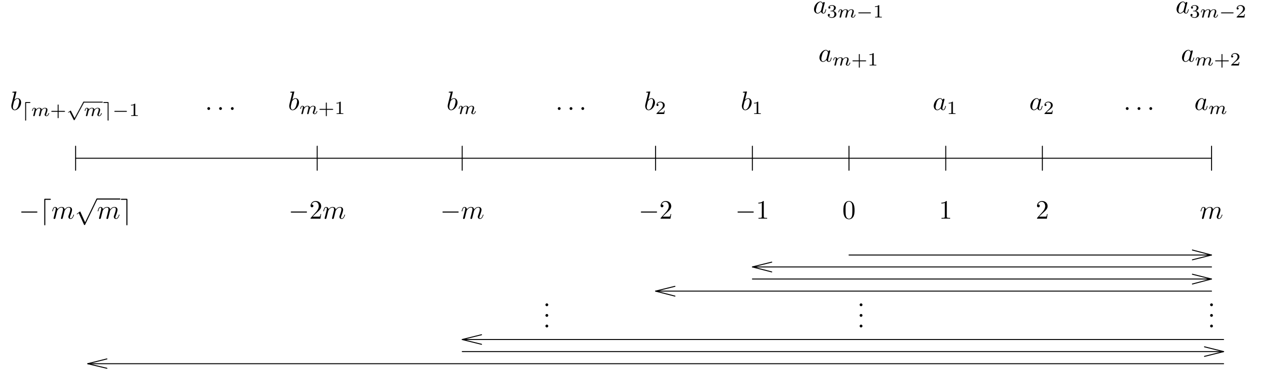

Figure 7: Lower bound input instance I_1

The optimal offline solution for input instance I_1 is as follows: the server moves right to point m and then moves left to service one request of client B. This pattern is repeated m times. Finally, the server moves left to point $-m\lceil\sqrt{m}\,\rceil$. For input instance I_2, the server does the same pattern of movement but moves left in place of moving right and vice versa. It is easy to verify that $\mathcal{OPT}(I_1) = \mathcal{OPT}(I_2) = 4m-1$ (request b_m). The arrows in Figure 7 depict the server's movement for input I_1.

We now show that for any on-line algorithm $\mathcal{A}$, $\max(\mathcal{A}(I_1), \mathcal{A}(I_2)) \geq 2m\sqrt{m}$. An on-line algorithm $\mathcal{A}$ cannot determine if an input instance is I_1 or I_2 until the server reaches point m or point $-m$. Without loss of generality, we assume $\mathcal{A}$'s server reaches point $-m$ before point m in which case the adversary chooses the input instance to be I_1 with client A on the right side of the origin. Thus request b_m is serviced before request a_m.

There are now two possibilities for $\mathcal{A}$. Either the final request of client B is serviced before the final request of client A or vice versa. In the first case, since all requests from A are at or to the right of the origin, then there must be a time, between servicing two consecutive requests from A (or before servicing the first one), when the server moves from the origin to point $-m\lceil\sqrt{m}\,\rceil$ and back to the origin. Thus, in this case, the maximum response time is at least $2m\lceil\sqrt{m}\,\rceil \geq 2m\sqrt{m}$. In the second case, note the server is currently at point $-m$ before request a_m has been serviced. Therefore, finishing client A will require the server to make m trips from the origin to point m and back. Furthermore, there are only $\lceil\sqrt{m}\,\rceil - 1 < \sqrt{m}$ requests of client B left to be serviced. Since client A finishes before client B, this means there must be some request of client B which must wait while the server makes at least $\sqrt{m}$ trips from the origin to point m and back to the origin. Thus, in this case, the maximum response time again is at least $2m\sqrt{m}$. Thus, $\mathcal{A}(I_1) \geq 2m\sqrt{m}$.

Combining these results, we get a lower bound on the competitive ratio of any on-line algorithm of $\frac{2m\sqrt{m}}{4m-1} > \frac{\sqrt{m}}{2}$. The theorem follows from observing that from the choice of m, $2(m-1)\lceil\sqrt{m-1}\,\rceil < \Delta \leq 2m\lceil\sqrt{m}\,\rceil$, *i.e.*, $\Delta = \Omega(\sqrt{m}^3)$. □

On the continuous line metric space $line(\infty)$, this result improves as follows.

Corollary 4.1. *On line(∞), no on-line algorithm is better than $\Omega(\sqrt{n_{max}(I)})$-competitive for the maximum response time cost function, even when the input has only 2 clients. Thus, no on-line algorithm is competitive on line(∞) for the maximum response time cost function.*

Proof. On $line(\infty)$, we can embed input instance I_1 and I_2 of the proof of Theorem 4.1 for arbitrarily large values of m. The specific value of the lower bound follows from observing that $n_{max}(I_1) = n_{max}(I_2) = 3m - 1$. □

4.2 Analysis of Disk Scheduling Algorithms.

Because of Corollary 4.1, we restrict our attention to the $line(\Delta)$ metric space for the rest of this section.

Theorem 4.2. *On line(Δ), $c_{\mathcal{SDF}}$ and $c_{\mathcal{SEQ}}$ are unbounded for the maximum response time cost function, even when the input has only two clients.*

Proof. Consider the following input instance I where $k = 2$. The second client generates a single request at point $d > 2$ and the first client generates a sequence of $2n$ requests that alternate between points 0 and 1. Assuming the server is initially at position 0, the $\mathcal{SDF}$ algorithm will service all of the first client's requests before servicing the second client's request. Therefore $\mathcal{SDF}(I) = 2n + d$. Meanwhile, $\mathcal{OPT}(I) = 2d - 1$. Since n can be arbitrarily large, the competitive ratio can be unbounded. The same example results in an unbounded competitive ratio for the $\mathcal{SEQ}$ algorithm. □

In order to analyze the remaining algorithms, it is helpful to make the following observation. A trivial input instance is one in which all of the requests are located at the server's original position. Clearly any algorithm can serve this instance with a maximum response time of 0. For all other input instances, the cost of any algorithm is at least 1. In particular, for any non-trivial input instance I, $\mathcal{OPT}(I) \geq 1$.

Theorem 4.3. *On line(Δ), $\frac{2\Delta-1}{3} \leq c_{\mathcal{EL}} \leq 2\Delta - 1$ for the maximum response time cost function.*

Proof. Clearly, the longest any request must wait to be served by $\mathcal{EL}$ is $2\Delta - 1$. This case can happen if a new request appears on an endpoint of the line just after the server has left that endpoint. Combining this with the fact that $\mathcal{OPT}(I) \geq 1$, it follows that $c_{\mathcal{EL}} \leq 2\Delta - 1$.

For the lower bound, consider the two client input instance I in Figure 8. Assuming that the server is at position 1 and going to the right, $\mathcal{EL}(I) = 2\Delta - 1$. Meanwhile $\mathcal{OPT}(I) = 3$ and the result follows. Note, this configuration can be reached even if we assume the server starts in the middle of the line. For example, if the server starts at position $\Delta/2$, we place the ith request of both clients at position $\Delta/2 - i$ for $1 \leq i \leq \Delta/2$ and the $\Delta/2 + 1^{st}$ request of both clients at position 1. After servicing these requests, the server will be at position 1 and moving to the right, so the lower bound example applies. □

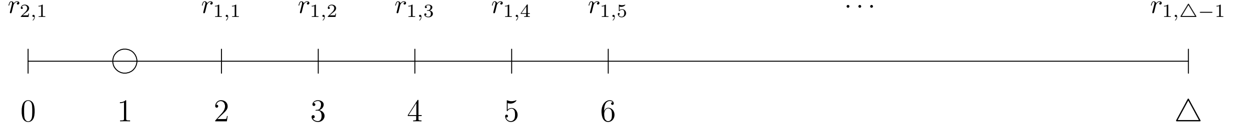

Figure 8: Worst Case Input for $\mathcal{EL}$

Corollary 4.2. *On line*(Δ), $c_{\mathcal{MP}} \geq \left(\frac{2\Delta-1}{3}\right)$ *for the maximum response time cost function.*

Proof. The lower bound input instance is the same as the one depicted in Figure 8 assuming that ties are broken to $\mathcal{MP}$'s detriment. If we wish to eliminate ties, then we can alter the input instance slightly by having the server begin at position 2, the first request of client 2 is still at position 0, and the ith request of client 1 is at position $i+2$ for $1 \leq i \leq \Delta - 2$ resulting in a lower bound of $\left(\frac{2\Delta-2}{5}\right)$. □

Theorem 4.4. *On line*(Δ), $\max(\Omega(\min(k,\Delta)), \Omega(\sqrt{\Delta})) \leq c_{\mathcal{FCFS}} \leq k\Delta$ *for the maximum response time cost function.*

Proof. We first show that $\mathcal{FCFS}(I) \leq k\Delta$. This follows from observing that the $\mathcal{FCFS}$ server will, at the latest, prioritize a request r after prioritizing the $k-1$ requests of other clients that are available when request r arrives. Furthermore, once a request is prioritized, it will take at most Δ time to service it. Therefore $c_{\mathcal{FCFS}} \leq k\Delta$.

To see the lower bound of $\Omega(\min(k,\Delta))$, consider the input instance I illustrated in Figure 9. Without loss of generality, we assume that k and Δ are even and that $k \leq \Delta$. The clients are numbered according to the priority assigned to them by $\mathcal{FCFS}$. The single requests for odd numbered clients are located at positions 0 through $\frac{k}{2}-1$ starting at position $\frac{k}{2}-1$, and the single requests for even numbered clients are located at positions $\Delta - \frac{k}{2} + 1$ through Δ starting at position $\Delta - \frac{k}{2} + 1$. Clearly, $\mathcal{FCFS}(I) = \Omega(k\Delta)$. Meanwhile, $\mathcal{OPT}(I) = 3\Delta/2$. Thus, the result follows. Note this lower bound input instance generalizes

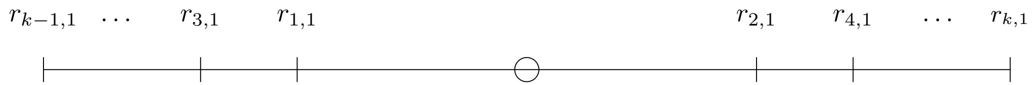

Figure 9: $\Omega(k)$ Lower Bound Instance for $\mathcal{FCFS}$

to clients with more than 1 request. For example, the first n requests of all clients could oscillate between the two middle points with the final requests as in the figure.

To see the lower bound of $\Omega(\sqrt{\Delta})$, consider the input instance I illustrated in Figure 10 with 2 clients. Let $n_1 = m$ and $n_2 = 2m-1$. The ith request of client 1, $r_{1,i}$, is at position $2 + (i-1)y$ for $1 \leq i \leq m$. Choose y and m such that $2 + (m-1)y = \Delta$ so that the mth request of client 1, $r_{1,m}$, is at position Δ. The requests from client 2 alternate between position 0 and position 1 starting at position 0. □

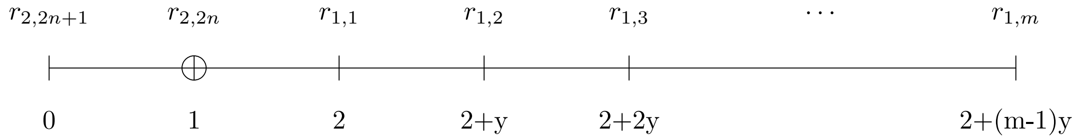

Figure 10: $\Omega(\sqrt{\Delta})$ Lower Bound Instance for $\mathcal{FCFS}$

Assume without loss of generality that $\mathcal{FCFS}$ initially services the first request of client 1 (otherwise we simply add 2 more requests to client 2). $\mathcal{FCFS}$ will behave as follows. Each time the server moves left, it is to service a client 2 request at position 0. It will then move right to pick up the next request of client 1, picking up one request of client 2 at position 1 for free. The maximum response time is the response time for the last request from client 2, which is $2\Delta - 1$. Therefore $\mathcal{FCFS}(I) = 2\Delta - 1$. The optimal solution is to service all of the requests from client 2 first, so $\mathcal{OPT}(I) = \max(2m + 1, y)$. Thus the competitive ratio of $\mathcal{FCFS}$ on this input instance is

$$\frac{2\Delta - 1}{\max(2m + 1, y)}$$

If we choose $y = 2m + 1$ and remember the fact that $2 + (m - 1)y = \Delta$, it follows that $c_{\mathcal{FCFS}} = \Omega(\sqrt{\Delta})$.

5 The k-client problem with non-zero processing times

Prior to this section, we have assumed that client requests can be processed in zero time. In this section, we consider what effects non-zero processing times have on this problem. We make the following assumptions about the processing of requests. First, we assume a non-preemptive environment. Once the server begins processing a request, it cannot stop until the request is completely processed. Second, we assume a nonclairvoyant model where the server does not know the processing time of a request until that request is completed. Finally, we assume that client requests cannot be processed in parallel, even if they are located at the same position.

Since we have processing times, we define a new cost function, maximum completion time. Without processing times, maximum completion time and total distance are identical assuming the server never idles unnecessarily. However, with processing times, these two cost functions are different, and maximum completion time is more relevant. In this section, we primarily focus on the maximum completion time and average completion time cost functions and mostly ignore the total distance cost function as processing times have no effect on the distance the server has moved.

Our main conclusions are the following. If an algorithm $\mathcal{A}$ does not skip over requests, and if $\mathcal{A}$ is c-competitive for the maximum completion time cost function where $c \geq k$ when there are no processing times, then $\mathcal{A}$ will be no worse than c-competitive for the maximum

completion time cost function when there are non-zero processing times. The same is also true of the average completion time cost function. One interesting point is that the $\mathcal{FCFS}$ algorithm becomes easier to analyze since it will no longer pick up intermediate requests for free. We take advantage of this by proving some strong lower bounds on the performance of the $\mathcal{FCFS}$ algorithm.

5.1 Definitions and Notation

Let I be an input instance of the k-client problem with non-zero processing times.

- Let $x_{i,j} \geq 0$ denote the processing time of request $r_{i,j} \in I$.
- Let $x_{max} = \max_{1 \leq i \leq k, 0 \leq j \leq n_i} x_{i,j}$, the maximum processing time of any request.
- Let $X_i(I) = \sum_{j=0}^{n_i} x_{i,j}$, the time required just to process all the requests of client i,
- Let $X(I) = \sum_{i=1}^{k} X_i(I)$, the time required to process all requests of all k clients.
- Let $X_{max}(I) = \max_{1 \leq i \leq k} X_i(I)$.
- Let $t_{i,j} = d_{i,j-1} + x_{i,j}$, the minimum time required to process request $r_{i,j}$ once request $r_{i,j-1}$ has been completed for $1 \leq j \leq n_i$.
- Let $t_{max}(I) = \max_{1 \leq i \leq k, 0 \leq j \leq n_i} t_{i,j}$.
- Let $T_i(I) = \sum_{j=0}^{n_i} t_{i,j} = D_i(I) + X_i(I)$. This term represents the minimum time required to service client i (ignoring all other clients).
- Let $T_{max}(I) = \max_{1 \leq i \leq k} T_i(I)$
- Let $T(I) = \sum_{i=1}^{k} T_i(I) = D(I) + X(I)$.
- Let I^0 be the equivalent input instance where $x_{i,j} = 0$ for all i, j. That is, I^0 is the equivalent input instance for the basic k-client problem.

We will typically omit I when the input instance referred to by I is not ambiguous.

Consider any algorithm $\mathcal{A}$ for the k-client problem with non-zero processing times and cost function CF. We let $c_{\mathcal{A}}^{\text{CF}}$ denote the competitive ratio of algorithm $\mathcal{A}$ for the basic k-client problem for cost function CF with all processing times equal to 0; that is, $c_{\mathcal{A}}^{\text{CF}} = \sup_{I^0} \frac{\mathcal{A}^{\text{CF}}(I^0)}{\mathcal{OPT}^{\text{CF}}(I^0)}$. We then define $c_{p,\mathcal{A}}^{\text{CF}}$ to be the competitive ratio of algorithm $\mathcal{A}$ for the k-client problem with non-zero processing times for cost function CF; that is, $c_{p,\mathcal{A}}^{\text{CF}} = \sup_I \frac{\mathcal{A}^{\text{CF}}(I)}{\mathcal{OPT}^{\text{CF}}(I)}$. When the cost function is not ambiguous, we will drop the CF superscript.

5.2 Maximum Completion Time cost function

We first observe that we can divide the time spent by any algorithm $\mathcal{A}$ into two components: times when the server is moving and times when the server is stationary and processing a request. This is captured in the following fact.

Fact 5.1. *For all algorithms $\mathcal{A}$ and all input instances I, $\mathcal{A}^T(I) = \mathcal{A}^D(I) + X(I)$.*

We define a non-skipping algorithm to be an algorithm which always processes a request it is in position to serve; that is, it never skips over a request and leaves it unprocessed. The algorithms $\mathcal{SDF}$, $\mathcal{EL}$, and the optimal maximum completion time algorithm are non-skipping algorithms, whereas $\mathcal{SEQ}$ and $\mathcal{FCFS}$ are not non-skipping algorithms. We observe the following fact about $\mathcal{A}^D(I)$ for any non-skipping algorithm $\mathcal{A}$.

Fact 5.2. *For all non-skipping algorithms $\mathcal{A}$ and all input instances I, $\mathcal{A}^D(I) = \mathcal{A}^D(I^0)$.*

This leads us to the following conclusion about $c_{p,\mathcal{A}}$ for any non-skipping algorithm $\mathcal{A}$.

Theorem 5.1. *For all non-skipping algorithms $\mathcal{A}$ $c^T_{p,\mathcal{A}} \leq c^D_{\mathcal{A}}$.*

Proof. For any input instance I, use Fact 5.1 and Fact 5.2 to conclude that that $\mathcal{A}^T(I) = \mathcal{A}^D(I^0) + X(I)$ and $\mathcal{OPT}^T(I) = \mathcal{OPT}^D(I^0) + X(I)$. Thus, for all input instances I, $\frac{\mathcal{A}^T(I)}{\mathcal{OPT}^T(I)} = \frac{\mathcal{A}^D(I^0)+X(I)}{\mathcal{OPT}^D(I^0)+X(I)} \leq \frac{\mathcal{A}^D(I^0)}{\mathcal{OPT}^D(I^0)}$ since $X(I) \geq 0$ and $\mathcal{A}^D(I^0) \geq \mathcal{OPT}^D(I^0)$. Thus the result follows. □

Corollary 5.1. *The $\mathcal{SDF}$ algorithm is $(2k-1)$-competitive for the maximum completion time cost function, even when non-zero processing times are allowed.*

In analyzing specific skipping algorithms, we see that the competitive ratio of the $\mathcal{SEQ}$ algorithm, $c_{p,\mathcal{SEQ}}$, is still $2k-1$ whereas the competitive ratio of the $\mathcal{FCFS}$ algorithm, $c_{p,\mathcal{FCFS}}$, is much worse depending on our assumptions about the relationship between distances and processing costs.

Theorem 5.2. *The $\mathcal{SEQ}$ algorithm is $(2k-1)$-competitive for the maximum completion time cost function, even when non-zero processing times are allowed.*

Proof. The proof is essentially identical to the proof of Theorem 3.3 replacing D_i with T_i. □

Theorem 5.3. *There exists an input instance I on line(Δ) with only 2 clients where $\frac{\mathcal{FCFS}^D(I)}{\mathcal{OPT}^D(I)}$ and $\frac{\mathcal{FCFS}^T(I)}{\mathcal{OPT}^T(I)}$ are unbounded given that consecutive requests of a client may be located in the same position and our restriction that requests cannot be serviced in parallel.*

Proof. Consider an input instance I on *line*(Δ) with two clients where each client has n requests. Request $r_{1,i}$ for $1 \leq i \leq n$ is located at position 1 while request $r_{2,i}$ for $1 \leq i \leq n$ is located at position Δ. Furthermore, $x_{i,j} = \epsilon$ for $1 \leq i \leq 2$ and $1 \leq j \leq n$. Note $\mathcal{FCFS}$ will process I alternating service between the 2 clients. Thus, $\mathcal{FCFS}^D(I) = 2n\Delta$ while $\mathcal{OPT}^D(I) = \Delta$. Since n can be arbitrarily large, the total distance cost function result follows. If we consider the maximum completion time cost function, we have $\mathcal{OPT}^T(I) = \Delta + 2n\epsilon$ whereas $\mathcal{FCFS}^T(I) = 2n\Delta + 2n\epsilon$. Since ϵ can be chosen so that $2n\epsilon = \Delta$ where n can be arbitrarily large, the maximum completion time cost function result follows. □

Finally, because of Fact 5.1 and the existence of non-skipping algorithms, we cannot improve our lower bound argument using input instances where requests have non-zero processing times. That is, if we consider adversary strategies which utilize some input instances $\{I_i\}$ with non-zero processing times to derive a lower bound, we can derive the same lower bound by replacing input instances $\{I_i\}$ with input instances $\{I_i^0\}$.

5.3 Average Completion Time cost function

We now consider the average completion time cost function. We again will analyze algorithms by breaking down time into two components: those times when the server is moving and those times when the server is processing a request. Let $\mathcal{A}_m^{\text{ACT}}(I)$ denote the component of $\mathcal{A}^{\text{ACT}}(I)$ that corresponds to times when the server is moving and $\mathcal{A}_x^{\text{ACT}}(I)$ denote the component of $\mathcal{A}^{\text{ACT}}(I)$ that corresponds to times when the server is processing a request. By definition then, we have the following fact.

Fact 5.3. *For all algorithms $\mathcal{A}$ and all input instances I, $\mathcal{A}^{ACT}(I) = \mathcal{A}_m^{ACT}(I) + \mathcal{A}_x^{ACT}(I)$.*

We first observe the following bounds on $\mathcal{A}_x^{\text{ACT}}(I)$ for any algorithm $\mathcal{A}$.

Fact 5.4. *For all algorithms $\mathcal{A}$ and all input instances I, $\frac{X(I)}{k} \leq \mathcal{A}_x^{ACT}(I) \leq X(I)$.*

We then observe the following for any non-skipping algorithm $\mathcal{A}$.

Fact 5.5. $\mathcal{A}_m^{ACT}(I) = \mathcal{A}^{ACT}(I^0)$.

We also observe the following fact for $\mathcal{OPT}$, the optimal average completion time algorithm. Note that the optimal average completion time algorithm may actually be a skipping algorithm and may be different than the optimal maximum completion time algorithm.

Fact 5.6. $\mathcal{OPT}_m^{ACT}(I) \geq \mathcal{OPT}^{ACT}(I^0)$.

Putting together the previous series of facts, we can conclude the following.

Theorem 5.4. *For any non-skipping algorithm $\mathcal{A}$ with a competitive ratio without processing times $c_{\mathcal{A}} \geq k$ for the average completion time cost function, its competitive ratio $c_{p,\mathcal{A}}$ with non-zero processing times for the average completion time cost function is equal to $c_{\mathcal{A}}$.*

Proof. For any input instance I, using Facts 5.3, 5.4, and 5.5 we find the following upper bound on $\mathcal{A}^{\text{ACT}}(I)$.

$$\begin{aligned} \mathcal{A}^{\text{ACT}}(I) &= \mathcal{A}_m^{\text{ACT}}(I) + \mathcal{A}_x^{\text{ACT}}(I) \\ &= \mathcal{A}^{\text{ACT}}(I^0) + \mathcal{A}_x^{\text{ACT}}(I) \\ &\leq c_{\mathcal{A}}\mathcal{OPT}^{\text{ACT}}(I^0) + X(I) \end{aligned}$$

Using Facts 5.3, 5.4, and 5.6, we find the following lower bound on $\mathcal{OPT}^{\text{ACT}}(I)$.

$$\begin{aligned}\mathcal{OPT}^{\text{ACT}}(I) &= \mathcal{OPT}_m^{\text{ACT}}(I) + \mathcal{OPT}_x^{\text{ACT}}(I) \\ &\geq \mathcal{OPT}^{\text{ACT}}(I^0) + \frac{X(I)}{k}\end{aligned}$$

Clearly if $c_{\mathcal{A}} \geq k$, then $\frac{\mathcal{A}^{\text{ACT}}(I)}{\mathcal{OPT}^{\text{ACT}}(I)} \leq c_{\mathcal{A}}$ for all input instances I, and the result follows.. □

Corollary 5.2. *The $\mathcal{SDF}$ algorithm is $(2k-1)$-competitive for the average completion time cost function, even when non-zero processing times are allowed.*

Theorem 5.5. *The $\mathcal{SEQ}$ algorithm is $(2k-1)$-competitive for the average completion time cost function, even when non-zero processing times are allowed.*

Proof. The proof is essentially identical to the proof of Theorem 3.4 replacing D_i with T_i. □

Corollary 5.3. *There exists an input instance I on line(∞) with only 2 clients where $\frac{\mathcal{FCFS}^{ACT}(I)}{\mathcal{OPT}^{ACT}(I)}$ is unbounded given that consecutive requests of a client may be located in the same position and our restriction that requests cannot be serviced in parallel.*

Proof. The result follows from considering the same input instance I used in Theorem 5.3. □

When considering the maximum completion time cost function, we could not prove a better lower bound with non-zero processing times. However, the situation is quite different with the average completion time cost function. We can prove a much stronger lower bound than we were able to in the zero processing time setting.

Theorem 5.6. *For any on-line algorithm $\mathcal{A}$, $c_{p,\mathcal{A}}^{ACT} \geq k$.*

Proof. Suppose all k clients have only a single request located at the original server position where the request of one client has length n and the other $k-1$ requests have length 1. We can assume the on-line algorithm $\mathcal{A}$ first processes the request of length n. Since we have assumed a non-preemptive scheduling model, this request must be completely processed before any of the other requests begin. Thus, $\mathcal{A}^{\text{ACT}}(I) > kn$ whereas $\mathcal{OPT}$ first processes the $k-1$ requests of length 1 so $\mathcal{OPT}^{\text{ACT}}(I) \leq k^2 + n$. Since n can be arbitrarily large, $c_{p,\mathcal{A}}$ asymptotically approaches k. □

6 Multiple Servers

We now consider the *k-client l-server problem* where there are l servers instead of only a single server. We consider two models. In the first model, all servers reside in the same metric space, and any server can service any request. This model is the natural generalization of the classic l-server problem to include multiple clients. We also consider the case where the

servers reside in separate metric spaces and client requests can appear in any metric space. This model is suitable for modeling a multiple surface disk system where each head can serve a single surface.

Our main result is a general technique for combining any algorithm for the l-server problem with the k-client $\mathcal{SEQ}$ algorithm to produce a competitive k-client l-server algorithm. This result applies to both multiple server models. Note, we return to assuming that all requests have zero processing time and thus can be serviced for free. With respect to lower bounds, we cannot prove any better lower bounds for this generalized problem than simply those which hold for the k-client problem and the l-server problem.

6.1 The Total Distance and Maximum Completion Time cost functions with multiple servers

In this multiple server context, as in the non-zero processing time context, the total distance cost function and the maximum completion time cost function are not equivalent. The total distance function corresponds to the sum of the distances moved by each of the l servers. On the other hand, the maximum completion time cost function corresponds to the time required to process all k clients. These may differ in the multiple server setting because more than one server may be moving at the same time.

The following lemmas capture useful relationships between the performance of algorithms with respect to the maximum completion time and total distance cost functions.

Lemma 6.1. *For any algorithm $\mathcal{A}$ and any input instance I,*

$$\mathcal{A}^T(I) \leq \mathcal{A}^D(I) \leq l\mathcal{A}^T(I)$$

Proof. The lower bound follows from the observation that no time elapses unless at least one of the servers is moving. The upper bound follows from the fact that at most l servers can be moving at any time. Again, we are assuming that there is at least one request in the system at all times until all requests have been serviced and that the algorithm never idles all its servers when outstanding requests exist. □

Lemma 6.2. *Any algorithm $\mathcal{A}$ that is a strongly c-competitive algorithm with respect to the total distance cost function is at worst lc-competitive and at best $\frac{1}{l}c$-competitive with respect to the maximum completion time cost function.*

Proof. The upper bound result, $c^T_{\mathcal{A}} \leq lc^D_{\mathcal{A}}$, follows from applying Lemma 6.1 to observe that for any input instance I, $\mathcal{A}^T(I) \leq \mathcal{A}^D(I)$ and $l\mathcal{OPT}^T(I) \geq \mathcal{OPT}^D(I)$. The lower bound result, $c^T_{\mathcal{A}} \geq \frac{1}{l}c^D_{\mathcal{A}}$, follows from applying Lemma 6.1 to observe that for any input instance I, $l\mathcal{A}^T(I) \geq \mathcal{A}^D(I)$ and $\mathcal{OPT}^T(I) \leq \mathcal{OPT}^D(I)$. □

6.2 The $\mathcal{A}$-$\mathcal{SEQ}$ Algorithm

We now describe how we compose any competitive algorithm $\mathcal{A}$ for the l-server with the $\mathcal{SEQ}$ k-client algorithm to produce $\mathcal{A}$-$\mathcal{SEQ}$, a competitive algorithm for the k-client l-server algorithm.

Definition 6.1. *The $\mathcal{A}$-$\mathcal{SEQ}$ algorithm arbitrarily orders the clients from 1 to k and devotes all l servers to servicing the current client using the l-server algorithm $\mathcal{A}$.*

For analysis purposes, we assume all l servers move back to their initial positions immediately after each client has been completely served. We derive the following results on the competitive ratio of $\mathcal{A}$-$\mathcal{SEQ}$ for the total distance cost function.

Theorem 6.1. *If $\mathcal{A}$ is c-competitive for the l-server problem, then $\mathcal{A}$-$\mathcal{SEQ}$ is $(2k-1)c$-competitive for the k-client l-server problem for the total distance cost function.*

Proof. For any input instance I, let the clients be numbered according to the order that $\mathcal{A}$-$\mathcal{SEQ}$ prioritizes them. Furthermore, remember that I_i is the set of requests of client i for $1 \leq i \leq k$.

We first observe that $\mathcal{A}$-$\mathcal{SEQ}^D(I) \leq \mathcal{A}^D(I_k)+\sum_{i=1}^{k-1} 2\mathcal{A}^D(I_i) = \sum_{i=1}^{k} \mathcal{A}^D(I_i)+\sum_{i=1}^{k-1} \mathcal{A}^D(I_i)$. This follows from noting that the cost for $\mathcal{A}$-$\mathcal{SEQ}$ to return the servers to their initial positions after completing client i for $1 \leq i \leq k-1$ is at most $\mathcal{A}^D(I_i)$.

We next observe that $\mathcal{A}^D(I_i) \leq c\mathcal{OPT}^D(I_i) = cD_i(I) \leq cD_{max}(I)$ for $1 \leq i \leq k$. This follows from the fact that $\mathcal{A}$ is c-competitive for the l-server problem and the definitions of $D_i(I)$ for $1 \leq i \leq k$ and $D_{max}(I)$.

Combining the two previous observations, we get $\mathcal{A}$-$\mathcal{SEQ}^D(I) \leq (2k-1)cD_{max}(I)$. Since $\mathcal{OPT}^D(I) \geq D_{max}(I)$, the result follows. □

Corollary 6.1. *If $\mathcal{A}$ is c-competitive for the l-server problem, then $\mathcal{A}$-$\mathcal{SEQ}$ is $(2k-1)lc$-competitive for the k-client l-server problem for the maximum completion time cost function.*

Proof. This follows immediately from Theorem 6.1 and Lemma 6.2. □

Theorem 6.2. *If $\mathcal{A}$ is c-competitive for the l-server problem, then $\mathcal{A}$-$\mathcal{SEQ}$ is $(2k-1)lc$-competitive for the k-client l-server problem for the average completion time cost function.*

Proof. For any input instance I, we assume the clients are numbered according to the order in which $\mathcal{A}$-$\mathcal{SEQ}$ prioritizes them.

We first observe that the completion time of client i is at most $2\sum_{j=1}^{i-1} \mathcal{A}^D(I_j) + \mathcal{A}^D(I_i)$ for $1 \leq i \leq k$. We next observe that $\mathcal{A}^D(I_i) \leq c\mathcal{OPT}^D(I_i) = cD_i(I)$ for $1 \leq i \leq k$. This follows from the fact that $\mathcal{A}$ is c-competitive for the l-server problem and the definition of $D_i(I)$ for $1 \leq i \leq k$. Combining the previous two observations, we obtain the fact that the completion time of client i is at most $c\left(2\sum_{j=1}^{i-1} D_j(I) + D_i(I)\right)$.

Thus, we derive the following upper bound on $\mathcal{A}\text{-}\mathcal{SEQ}^{\text{ACT}}(I)$.

$$\begin{aligned}
\mathcal{A}\text{-}\mathcal{SEQ}^{\text{ACT}}(I) &\leq \left(\frac{1}{k}\right)\sum_{i=1}^{k} c\left(2\sum_{j=1}^{i-1} D_j(I) + D_i(I)\right) \\
&\leq \left(\frac{1}{k}\right) c \sum_{i=1}^{k} (2k-2i+1) D_i(I) \\
&\leq \left(\frac{1}{k}\right) c \sum_{i=1}^{k} (2k-1) D_i(I) \\
&\leq c(2k-1)\left(\frac{1}{k}\right)\sum_{i=1}^{k} D_i(I) \\
&= c(2k-1)\left(\frac{D(I)}{k}\right)
\end{aligned}$$

Meanwhile, the optimal algorithm cannot complete the requests of client i sooner than $\mathcal{OPT}^{T}(I_i) \geq \frac{D_i(I)}{l}$ for $1 \leq i \leq k$. Therefore

$$\mathcal{OPT}^{\text{ACT}}(I) \geq \left(\frac{1}{k}\right)\sum_{i=1}^{k}\frac{D_i(I)}{l} = \left(\frac{1}{l}\right)\left(\frac{D(I)}{k}\right).$$

Combining the upper bound on $\mathcal{A}\text{-}\mathcal{SEQ}^{\text{ACT}}(I)$ and the lower bound on $\mathcal{OPT}^{\text{ACT}}(I)$ leads to the result. □

6.3 Implications

We first apply the above results to the work function algorithm which is $(2l-1)$-competitive for the l-server problem [21].

Corollary 6.2. *For the k-client l-server problem, the work function-$\mathcal{SEQ}$ algorithm is $(2k-1)(2l-1)$-competitive for the total distance cost function, $(2k-1)(2l-1)l$-competitive for the maximum completion time cost function, and $(2k-1)(2l-1)l$-competitive for the average completion time cost function.*

Proof. This result follows from the result of [21], Theorems 6.1 and 6.2, and Corollary 6.1. □

We now consider the problem where there are l distinct metric spaces, each containing a single server. Any request of any client can appear in any metric space. Figure 11 is a possible input sequence for 4 clients and 3 servers.

We define the $\mathcal{SEQ}$ algorithm for the k-client l-server problem with l separate metric spaces to be the simple generalization of the $\mathcal{SEQ}$ algorithm for the k-client problem in a single metric space where a server does not move unless a request from the current client is

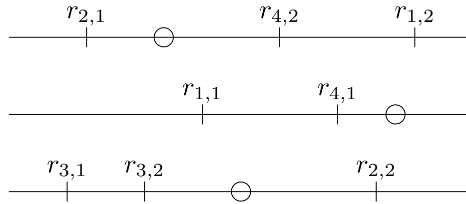

Figure 11: Multiple Servers in Separate Metric Spaces

in its metric space. Again, for analysis purposes, we assume that the servers move back to their initial positions whenever the current client has been completely processed.

Lemma 6.3. *For the total distance cost function, the $\mathcal{SEQ}$ algorithm is optimal for the 1-client l-server problem with l separate metric spaces.*

Proof. The key observation is that at any time, there is only one request in the system, and only one server can service that request. Since a server does not move when there are no requests in its metric space, that server will move no more than it must to service the client. □

This lemma and the results from the previous subsections leads to the following corollary.

Corollary 6.3. *For the k-client l-server problem with l separate metric spaces, the $\mathcal{SEQ}$ algorithm is $(2k-1)$-competitive for the total distance cost function, $(2k-1)l$-competitive for the maximum completion time cost function, and $(2k-1)l$-competitive for the average completion time cost function.*

Proof. This follows from Lemmas 6.3, Theorems 6.1 and 6.2, and Corollary 6.1. □

7 Conclusion

We have defined the k-client problem to model multi-threaded systems where multiple processes compete for scarce system resources. When we restrict our attention to the line metric space, this problem models the disk scheduling problem in a multi-threaded environment. We have tight results on the performance of many commonly studied disk scheduling algorithms which help explain why elevator type algorithms perform well in practice, particularly when the system is heavily loaded. Our results are essentially identical for the total distance, maximum completion time, and average completion time cost functions. Several open problems remain including closing the gap between the $2k-1$ upper bound and the $\frac{\lg k}{2}+1$ lower bound for the total distance and average completion time cost functions when $k \geq 3$.

8 Acknowledgments

The authors thank Abdol-Hossein Esfahanian, Ferit Kivanc, and Samik Sengupta for their many helpful comments and suggestions. The authors would also like to thank the anonymous referees for their many helpful suggestions.

References

[1] M. Andrews, M. Bender, and L. Zhang. New algorithms for the disk scheduling problem. In *Proceedings of IEEE FOCS*, 1996.

[2] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Serving requests with on-line routing. In *Proceedings of the 4th Scandinavian Workshop on Algorithms Theory (SWAT 94)*, pages 37–48, 1994.

[3] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Competitive algorithms for the on-line traveling-salesman. In *Proceedings of the 4th Workshop on Algorithms and Data Structures (WADS 95)*, pages 206–217, 1995.

[4] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha. On-line scheduling in the presence of overload. In *Proc. 32nd IEEE Symp. on the Foundations of Computer Science*, pages 101–110, 1991.

[5] A. Borodin, N. Linial, and M. Saks. An optimal online algorithm for metrical task system. *Journal of the ACM*, 39:745–763, 1992.

[6] Tung-Shou Chen. SIMPLE: An optimal disk system with two restricted heads. *Information Processing Letters*, 55:273–277, 1995.

[7] E. Coffman and M. Hofri. On the expected performance of scanning disks. *SIAM Journal on Computing*, 11:60–70, 1982.

[8] E. Coffman, L. Klimko, and B. Ryan. Analysis of scanning policies for reducing disk seek times. *SIAM Journal of Computing*, 1(3):269–279, 1972.

[9] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[10] X. Deng and C.H. Papadimitriou. Exploring an unknown graph. In *Proc. 31st IEEE Symp. on Foundations of Computer Science*, pages 355–361, 1990.

[11] M. Dertouzos and A. Mok. Multiprocessor on-line scheduling of hard-real-time tasks. *IEEE Transactions on Software Engineering*, 15:1497–1506, 1989.

[12] A. Feldmann, M.-Y. Kao, J. Sgall, and S-H. Teng. Optimal online scheduling of parallel jobs with dependencies. *Journal of Combinatorial Optimization*, 1:393–411, 1998.

[13] A. Feldmann, J. Sgall, and S-H. Teng. Dynamic scheduling on parallel machines. In *Proc. of 32nd IEEE Symp. on Foundations of Computer Science*, pages 111–120, 1991.

[14] E. Feuerstein. *On-line Paging of Structured Data and Multi-threaded Paging*. PhD thesis, University of Rome, 1995.

[15] A. Fiat, R. Karp, M. Luby, L. McGeoch, D. Sleator, and N. Young. Competitive paging algorithms. *Journal of Algorithms*, 12:685–699, 1991.

[16] R. Geist and S. Daniel. A continuum of disk scheduling algorithms. *ACM Transactions on Computer Scheduling*, 5(1):77–92, 1987.

[17] B. Kalyanasundaram and K. Pruhs. Constructing competitive tours from local information. *Theoretical Computer Science*, 130:125–138, 1994.

[18] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. In *Proceedings of the 34th Annual IEEE Foundations of Computer Science*, pages 214–221, 1995. To appear in *Journal of the ACM*.

[19] M-Y. Kao, J.H. Reif, and S.R Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. *Information and Computation*, 131:63–80, 1997.

[20] G. Koren and D. Shasha. MOCA: A multiprocessor on-line competitive algorithm for real-time system scheduling. *Theoretical Computer Science*, 128:75–97, 1994.

[21] E. Koutsoupias and C. Papadimitriou. On the k–server conjecture. *Journal of the ACM*, 42(5):971–983, 1995.

[22] E. Koutsoupias, P. Papadimitriou, and M. Yannakakis. Searching a fixed graph. In *Int. Colloq. on Automata, Languages, and Programming (ICALP 96)*, 1996.

[23] M.S. Manasse, McGeoch L.A., and D.D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990.

[24] R. Motwani, S. Phillips, and E. Torng. Non-clairvoyant scheduling. *Theoretical Computer Science*, 130(1):17–47, 1994.

[25] W. Oney. Queuing analysis of the scan policy for moving-head disks. *Journal of the ACM*, 22(3):397–412, 1975.

[26] P. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84:127–150, 1991.

[27] M. Seltzer, P. Chen, and J. Ousterhout. Disk scheduling revisited. In *USENIX*, pages 313–324, 1990.

[28] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *CACM*, 28:202–208, 1985.

[29] T. Teorey and T. Pinkerton. A comparative analysis of disk scheduling policies. *Communication of the ACM*, 15(3):177–184, 1972.

[30] Jean Vuillemin. A data structure for manipulating priority queues. *Communications of the ACM*, 24:309–315, 1978.

[31] B. Worthington, G. Ganger, and Y. Patt. Scheduling algorithms for modern disk drives. In *SIGMETRICS*, pages 241–251, 1994.

[32] A.C.-C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proc. of 18th IEEE Symp. on Foundations of Computer Science*, pages 222–227, 1977.

A Appendix

Definition A.1. *$B(n)$ is the set of non-negative integers such that $\sum_{b\in B(n)} 2^b = n$.*

Lemma A.1. *$\sum_{b\in B(n)} 2^b(b'-b) < n$ where $b' = \max B(n)$.*

Proof. Let $B_n = |B(n)|$, and let $b_1 < b_2 < ... < b_{B_n}$ be the elements of $B(n)$.

$$\begin{aligned}
\sum_{b\in B(n)} 2^b(b'-b) &= \sum_{1\le i\le B_n} 2^{b_i}(b_{B_n}-b_i) \\
&= \sum_{1\le i\le B_n-1} 2^{b_i}(b_{B_n}-b_i) \\
&= \sum_{1\le i\le B_n-1} 2^{b_i} \sum_{i+1\le j\le B_n} (b_j - b_{j-1}) \\
&= \sum_{1\le i\le B_n-1} \sum_{i+1\le j\le B_n} 2^{b_i}(b_j - b_{j-1}) \\
&= \sum_{1\le i<j\le B_n} 2^{b_i}(b_j - b_{j-1}) \\
&= \sum_{2\le j\le B_n} \sum_{1\le i\le j-1} 2^{b_i}(b_j - b_{j-1}) \\
&\le \sum_{2\le j\le B_n} \sum_{1\le i\le j-1} 2^{b_i} 2^{(b_j - b_{j-1}-1)} \quad \text{because } x \le 2^{x-1} \text{ for all integers } x \\
&= \sum_{2\le j\le B_n} \sum_{1\le i\le j-1} 2^{b_j + b_i - b_{j-1} - 1} \\
&= \sum_{2\le j\le B_n} \left(2^{b_j} \cdot 2^{-(b_{j-1}+1)} \cdot \sum_{1\le i\le j-1} 2^{b_i} \right)
\end{aligned}$$

$$
\begin{aligned}
&< \sum_{2\le j\le B_n} 2^{b_j}\cdot 2^{-(b_{j-1}+1)}\cdot 2^{b_{j-1}+1} \\
&= \sum_{2\le j\le B_n} 2^{b_j} \\
&< \sum_{1\le j\le B_n} 2^{b_j} \\
&= \sum_{b\in B(n)} 2^{b} \\
&= n
\end{aligned}
$$

□

Lemma A.2. $n\lg n \ < \ \sum_{b\in B(n)} 2^b(b+2) \ \le \ n\lg n + 2n.$

Proof.

$$
\begin{aligned}
n\lg n &= (\sum_{b\in B(n)} 2^b)(\lg \sum_{b\in B(n)} 2^b) \\
&< (\sum_{b\in B(n)} 2^b)(\lg 2^{b'+1}) \qquad \text{where } b' = \max B(n) \\
&= \sum_{b\in B(n)} 2^b(b'+1) \\
&= \sum_{b\in B(n)} 2^b(b+1+b'-b) \\
&= \sum_{b\in B(n)} 2^b(b+1) + \sum_{b\in B(n)} 2^b(b'-b) \\
&< \sum_{b\in B(n)} 2^b(b+1) + \sum_{b\in B(n)} 2^b \qquad \text{by Lemma A.1} \\
&= \sum_{b\in B(n)} 2^b(b+2) \\
\sum_{b\in B(n)} 2^b(b+2) &\le \sum_{b\in B(n)} 2^b(b'+2) \qquad \text{where } b' = \max B(n) \\
&= (b'+2)\sum_{b\in B(n)} 2^b \\
&= (b'+2)n \\
&\le (\lg n + 2)n \qquad \text{because } 2^{b'} \le \sum_{b\in B(n)} 2^b = n \\
&= n\lg n + 2n
\end{aligned}
$$

□

Lemma A.3. *For any integer $\Delta \geq 20$, there exists an integer m such that $g(m) \leq \Delta \leq 2f(m)$ where $f(m) = m\lceil\sqrt{m}\,\rceil$ and $g(m) = m\lceil\sqrt{m}\,\rceil + m$.*

Proof. First, observe that $g(5) = 20 < 30 = 2f(5)$. Given an integer $\Delta \geq 20$, let y be the real number such that $2f(y) = \Delta$. Let $m = \lceil y \rceil$. If we can show that $g(x+1) \leq 2f(x)$ for all real $x \geq 5$, then $g(m) = g(\lceil y \rceil) \leq g(y+1) \leq 2f(y) = \Delta = 2f(y) \leq 2f(\lceil y \rceil) \leq 2f(m)$ for all $\Delta \geq 20$, and the result follows. It remains to show that $g(x+1) \leq 2f(x)$ for all real $x \geq 5$.

$$
\begin{aligned}
\frac{g(x+1)}{f(x)} &= \frac{(x+1)(\lceil\sqrt{x+1}\,\rceil + 1)}{x\lceil\sqrt{x}\,\rceil} \\
&\leq \left(1+\frac{1}{x}\right)\frac{\lceil\sqrt{x+1}\,\rceil + 1}{\lceil\sqrt{x}\,\rceil} \\
&\leq \left(1+\frac{1}{x}\right)\left(1+\frac{1}{\lceil\sqrt{x}\,\rceil}+\frac{1}{\lceil\sqrt{x}\,\rceil}\right) \\
&\leq \left(1+\frac{1}{5}\right)\left(1+\frac{2}{\lceil\sqrt{5}\,\rceil}\right) \quad \text{for } x \geq 5 \\
&= \left(1+\frac{1}{5}\right)\left(1+\frac{2}{3}\right) \quad = \quad 2
\end{aligned}
$$

where the second inequality follows from the following.

For all $x > 0$,

$$
\begin{aligned}
x+1 &\leq x + 2\sqrt{x} + 1 \\
\sqrt{x+1} &\leq \sqrt{x} + 1 \\
\left\lceil\sqrt{x+1}\right\rceil &\leq \lceil\sqrt{x}\,\rceil + 1 \\
\frac{\lceil\sqrt{x+1}\,\rceil}{\lceil\sqrt{x}\,\rceil} &\leq 1 + \frac{1}{\lceil\sqrt{x}\,\rceil}.
\end{aligned}
$$

□