

# The $k$ -Server Problem and Fractional Analysis

Duru Türkoğlu \*

November 7, 2005

## Abstract

The  $k$ -server problem, introduced by Manasse, McGeoch and Sleator [29, 30] is a fundamental online problem where  $k$  mobile servers are required to serve a sequence of requests on a metric space, with the minimum possible movement cost. Despite the conjectured existence of a  $\Theta(\log k)$ -competitive randomized algorithm, the best known upper bound for arbitrary metric spaces is  $2k - 1$ . Even for one of the simplest cases, the Euclidean metric on the interval  $[0, 1]$ , nothing better than  $k$  is known. In this work, we first give a survey of the  $k$ -server problem, and then introduce a fractional version in order to provide a different perspective on randomized algorithms for this problem. In this version, servers can move fractionally instead of moving as a unit, and in the process, servicing a request requires providing one unit server at the point of request. We show that on the line and circle, the randomized version of the problem is equivalent to the fractional version. We also classify the cases for which these versions are not equivalent by presenting a fractional algorithm which cannot be simulated by any randomized algorithm. Furthermore, we investigate and analyze some algorithms on the line using the fractional setting.

## 1 Introduction

The  $k$ -server problem consists of a metric space  $\mathcal{M}$  and  $k$  mobile servers located at the points of this metric space. Given a sequence of requests  $\sigma$ , a list of points in  $\mathcal{M}$ , each request must be served by moving one of the  $k$  servers to the location requested, only if there is no server at that location. The cost of an algorithm for servicing a sequence  $\sigma$  is determined by the total distance traveled by these servers, based on the distance function  $d$  of the metric space  $\mathcal{M}$ . In an *online* setting, an algorithm has to serve the current request without the knowledge of the future requests, *i.e.* it has to decide which server to move based on the previous and current requests. In contrast, an *offline* algorithm has prior knowledge of the entire sequence.

*Competitive analysis* introduced by Sleator and Tarjan [35], has become the standard way to measure the performance of online algorithms. So, we can say that an online algorithm is *competitive* when it achieves costs within some constant factor of the cost of the best offline algorithm plus an additive constant. An online algorithm ALG is called  *$r$ -competitive* if there exists a constant  $\alpha$  such that for any request sequence  $\sigma$ ,

$$\text{cost}_{\text{ALG}}(\sigma) \leq r \cdot \text{cost}_{\text{OPT}}(\sigma) + \alpha$$

where  $\text{cost}_{\text{ALG}}(\sigma)$  and  $\text{cost}_{\text{OPT}}(\sigma)$  are the costs of the online algorithm ALG and the best offline algorithm OPT respectively. For the randomized case, we have a similar competitive analysis: A

---

\*E-Mail: [duru@cs.uchicago.edu](mailto:duru@cs.uchicago.edu), Department of Computer Science, The University of Chicago

randomized online algorithm ALG is called *r-competitive* if there exists a constant  $\alpha$  such that for any request sequence  $\sigma$ ,

$$\mathbb{E}[\text{cost}_{\text{ALG}}(\sigma)] \leq r \cdot \text{cost}_{\text{OPT}}(\sigma) + \alpha$$

Again, a randomized algorithm ALG is called *competitive* if there exists an  $r$  such that ALG is  $r$ -competitive. Also, for both settings, *the competitive ratio* of an algorithm can be defined as the infimum over such  $r$ 's and it provides worst-case guarantees on the ratio of the cost (expected cost for randomized setting) of the algorithm to the cost of the best offline sequence of choices with the benefit of hindsight. Finally, *the deterministic (resp. randomized) competitive ratio* for the problem is defined as the infimum over  $r$  for which there exists a deterministic (resp. randomized)  $r$ -competitive algorithm.

For the deterministic case, Manasse, McGeoch and Sleator not only define the question but also conjecture that there exists a  $k$ -competitive deterministic algorithm on any metric space [29]. The  $k$ -server conjecture is almost solved as there is a lower bound of  $k$  established by the founders of the problem [30], whereas Koutsoupias and Papadimitriou [27] proved that the work function algorithm is  $(2k - 1)$ -competitive on any metric space, closing the previously known exponential gap. For the randomized case, not much more than the deterministic case is known for arbitrary metric spaces; the best known algorithm is again the deterministic WFA. Unlike the deterministic case, the question is wide open in the randomized case. For the upper bound analysis, leaving aside more complex metric spaces, still not much is known for the line, even in the simplest case of two servers. On the other hand, the best known lower bound for a randomized algorithm is  $\Omega(\log k / \log \log k)$  due to Bartal et al. [3, 7].

In this paper, to provide a different point of view, we introduce another setting for the problem by allowing the servers to move fractionally instead of moving as a unit; therefore, servicing a request would mean providing one unit server at the point of request. We use the name *fractional algorithm* for algorithms moving servers as fractions. We show that any randomized algorithm can be simulated by a fractional algorithm, and for the line or circle we also show the converse, *i.e.* any fractional algorithm can be simulated by a randomized algorithm. We also prove the equivalence when  $k = 2$  or  $k = n - 1$  on any metric space with  $n$  points. For  $2 < k < n - 1$  on any metric space which is not a line or circle, we present a fractional algorithm which cannot be simulated by any randomized algorithm. Using the fractional setting, we develop a fractional version of the DOUBLE-COVERAGE algorithm, which attains a similar competitive ratio of  $2k$  on the line. Additionally, we analyze the basic question of randomized 2-server problem on the line and provide algorithms that reach the  $e/(e - 1)$  lower bound which is the best known lower bound for the line. The same lower bound was first introduced for general metric spaces by Karlin et al. [24] before it was increased to  $1 + e^{-1/2}$  by Chrobak et al. [16]. On the negative side, we point out some errors affecting the major result of a recent paper by Bartal and Mendel [8] on the randomized problem. Before the errors were acknowledged by the authors, the paper already appeared in *the Journal of Algorithms* as well as *the ACM-SIAM symposium on Discrete algorithms* in 2004.

## 1.1 Outline

In the rest of this section, we provide the standard notions that the reader should be aware of in the next subsection and then define the closely related problems in the literature. In section 2, we present a detailed survey of the problem for the deterministic and the randomized cases, as well as providing a critique on the recent paper of Bartal and Mendel [8], containing some errors. Later

in the following section, we formally define the fractional problem and analyze the equivalence between the randomized and the fractional versions. After having established the equivalence on the line, in section 4, we demonstrate a fractional version of the DOUBLE-COVERAGE algorithm, and finally, we analyze the simpler problem with only two servers on the line in the second half of the last section.

## 1.2 Preliminaries

First of all, the  $k$ -server problem and many other online problems were also stated against an *adversary*, who can find the worst sequence for each online algorithm. This hypothetical adversary has knowledge of our online algorithm ALG and therefore tries to produce a cruel sequence for which ALG will have the highest competitive ratio possible. For the deterministic case, by setting the request sequence beforehand, the adversary loses no power since all the information can be retrieved by just holding the information of the deterministic algorithm; there are no random bits the adversary is unaware of. In the randomized case, the situation is different. There are three possible types of adversaries: *oblivious*, *adaptive* and *fully adaptive*. An *oblivious* adversary still does not know about any of our random bits, and therefore sets the request sequence beforehand. On the other hand, an *adaptive* adversary can decide what the next request will be, based on the choice of the online algorithm for the current request. Yet the adversary can have even more power; a *fully adaptive* adversary has access to all of the random bits of the randomized algorithm. In the last setting, actually there is not much gain in randomization since each path of choices can be viewed as a deterministic algorithm. Throughout the paper, unless otherwise mentioned, we concentrate on the case in which the adversary is oblivious, *i.e.* none of our random bits are revealed to the adversary.

Another important concept is the *lazy* behaviour of an algorithm. A *lazy* algorithm always serves the current request with only one of its servers, and does not move any of its servers if there is already a server at the requested location. It is easy to see that any non-lazy algorithm can be converted to a lazy algorithm without incurring an extra cost. Suppose that a non-lazy algorithm moves more than one server or moves some servers even if there is a server at the requested location. We know that, before the next request, the current request will be served by a server. For the lazy version, we can move that server to serve the request and keep track of the other movements by means of virtual locations for servers but behave lazily, *i.e.* we do not move any other server. For future requests, we will use the virtual locations for deciding, and after making a choice, we will move only one server as described above and update the virtual locations. It is clear that the conversion does not bring an extra cost.

Before introducing any other concepts, we can give the definition of a configuration. A possible position of the  $k$  servers on the metric space  $\mathcal{M}$  is called a *configuration*. Without loss of generality, a configuration is a subset of  $\mathcal{M}$  of size  $k$ , for we can assume that, initially, all servers are at different locations.<sup>1</sup> Then by the laziness argument, at any time, all servers will be at different locations of  $\mathcal{M}$ . It is possible to define *memoryless* algorithms, algorithms which make their choices based only on their current configuration, without the knowledge of their previous requests. We can also talk about the *lazy* adversaries. A *lazy* adversary is an adaptive adversary, which requests the points of its current configuration while the online algorithm is not in the same configuration. In many

---

<sup>1</sup>Starting with a specified configuration, the adversary can lead to any configuration he wanted, by requesting its points a number of times. This procedure just changes the additive constant, but has no effect on the competitive ratio.

important cases lazy adversaries provide the worst sequences, *i.e.* non-lazy adversaries cannot force the online algorithm to have worse competitive ratios. This fact led to *the lazy adversary conjecture* [11], which states that against a memoryless online algorithm the optimal strategy for an adversary would be to behave lazily, *i.e.* to request the points in its configuration whenever the online algorithm is in a different configuration. Peserico has disproven this conjecture [32].

### 1.3 Related Problems

There are several problems closely connected with the  $k$ -server problem. Among them we can count *the paging problem*, (weighted or unweighted) *metrical task systems*, which is an abstraction to many other problems including the  $k$ -server problem and *the  $l$ -evader problem*, essentially the same  $k$ -server problem from a different perspective. Being one of the fundamental problems, *the paging problem* can be stated as follows.

**Definition 1 (The Paging Problem).** Considering a two-level memory system, fast and slow, each memory unit is called a *page*. Assuming that there are  $N$  slow pages and  $k$  fast pages among the slow pages, on a request of a page  $p$ , we need to bring the page  $p$  into the fast memory. If  $p$  is already in the fast memory (hit), there is nothing to be done, but if not (miss), we need to evict a page from the fast memory. *The paging problem* is the problem of minimizing the number of misses. *The weighted paging problem* is the problem of minimizing the cost due to the misses, where the cost of bringing a page into the fast memory is set by the weights of the pages.

The paging problem is a special case of the  $k$ -server problem, on a metric space of  $N$  points with all the distances being the same. For the weighted paging problem, we can create a star graph with  $N + 1$  nodes and define the distances to the center node as half of the page weight for  $N$  outer nodes. The  $k$ -server problem on this graph is equivalent to the weighted paging problem. Secondly, *the  $l$ -evader problem* introduced by Koutsoupias and Papadimitriou [26], is a generalized analog of the  $k$ -server problem as defined below.

**Definition 2 (The  $l$ -Evader Problem).** On the points of a finite metric space, there are  $l$  evaders, and these evaders respond to ejections by moving away from their locations when their locations are requested to be emptied. Again, the problem is to minimize the movement cost over a sequence of ejections.

One important note: We impose the condition that there can be a maximum of 1 evader at each location. Thus the  $l$ -evader problem on an  $n$  point metric space is equivalent to the  $(n - l)$ -server problem, by filling the unoccupied locations with servers and removing the evaders. Finally, the abstraction of many online problems, *metrical task systems*, is introduced by Borodin, Linial and Saks [12].

**Definition 3 (Metrical Task Systems).** A *metrical task system* consists of an  $N$  point metric space  $S$  with a metric distance function  $d : S \times S \rightarrow \mathbb{R}^+$ . Requests in this system are modeled as *tasks*, which are defined as  $N$ -ary vectors  $T = \langle T_1, T_2, \dots, T_N \rangle$  where each  $T_i \in \mathbb{R} \cup \{\infty\}$  is the processing cost for the state  $s_i \in S$ . Given an initial state, the problem is to minimize the cost on a sequence of tasks. Starting with the initial state, an algorithm may move to any state (or not move) for a transition cost determined by  $d$  and serve the next task with the processing cost for the occupied state defined by the task.

Every  $k$ -server problem is a special case of an MTS where each configuration of  $k$  servers is a state of the corresponding MTS with tasks having a zero processing cost if the request is an element of the state, and infinity otherwise. Transition cost from one state to another is the minimum cost for shifting between the corresponding configurations. Under these settings we can simulate every  $k$ -server problem by an MTS.

## 2 Overview of the Known Results

Before discussing the original and randomized versions of the problem in detail, we discuss some alternative models considered in the literature. Along with the original definition of the  $k$ -server problem in [29], the founders define the *asymmetric* version as well, where the graph defining the distances is a directed graph in which costs for moving in each direction may differ. Another intuitive generalization is *the weighted  $k$ -server problem*, introduced by Fiat and Ricklin [21], where each server has a weight assigned to it and the cost associated with that server is the server weight multiplied by the total distance traversed by that server. Apart from those, Bartal and Rosén [9] presented a *distributed* setting where there is no central decision maker; instead servers need to pay for communication to send any information. Yet another interesting variant is *the dynamic servers problem* introduced by Charikar et al. [13] which allows us to increase the number of servers at the expense of a rental cost. Also, according to Koutsoupias and Taylor [28], Saks and Burley considered another setting inspired by a crew trying to shoot scenes in Manhattan, *the CNN problem*. The CNN problem is defined on the Euclidean plane, wherein the requirement of serving a request is relaxed in the way that a server can move to a location that lies in the same horizontal or vertical line with the request. Another interesting variant was introduced by Borodin and El-Yaniv [11], *the  $(h, k)$ -server problem*. In this problem, they compare a  $k$ -server online algorithm to the best  $h$ -server offline algorithm with  $h \leq k$ , instead of  $h = k$ . Apart from these variants, we focus on the original and the randomized problems and especially try to understand the randomized version for it is a wide open area of research.

### 2.1 The Deterministic Version

The  $k$ -server conjecture was nearly brought to a conclusion when  $(2k - 1)$ -competitiveness of WFA was proven. Prior to the above result, the best known upper bound was exponential in  $k$ , inspired by a memoryless randomized algorithm, *the HARMONIC algorithm*, proven to be  $O(k2^k)$ -competitive by Grove in [22]. There were some improvements after the derandomization of the algorithm by Bartal and Grove in [6], which is currently the best known competitive ratio for the harmonic algorithm, but because the current ratio is  $O(2^k \log k)$ , it is still exponential in  $k$ .

Without the use of WFA, the conjecture was already proven for some special metric spaces. In one such case, Sleator and Tarjan [35] proved the result while working on *the paging problem*, a special case of the  $k$ -server problem, where all distances are the same in the metric space. For the paging problem, in the same paper [35], they also proved the lower bound of  $k$  for the deterministic case. On trees, Chrobak and Larmore in [15] came up with the extension of the DOUBLE-COVERAGE algorithm defined on the line by Chrobak et al. in [14] both proving the conjecture. In his PhD thesis [25], Koutsoupias also enlarged the set of metric spaces on which the conjecture holds by proving that WFA is  $k$ -competitive on metric spaces with at most  $k + 2$  points. On the other hand, the analogous work function algorithm for MTS was proven to settle the MTS problem for the

deterministic case. Borodin et al. proved both the lower bound and the upper bound of  $2N - 1$  in the same paper they describe MTS [12]. Coming back to the  $k$ -server problem, the question has remained at the same stage after the improvement of the competitive ratio to  $2k - 1$  by WFA with a gap of  $k - 1$  for most of the metric spaces.

### 2.1.1 Work Function Algorithm

It is shown that the greedy approach and the follow the leader (leader being the adversary) approach are not competitive at all. The work function algorithm, though, makes its choice not just by minimizing the distance to the requested point or by minimizing the distance of its configuration to that of the adversary, but both. For the purposes of WFA, Koutsoupias and Papadimitriou treat configurations as multisets instead of sets. For convenience, for any multiset  $X$  let  $X + a$  denote the multiset  $X \cup \{a\}$  and  $X - a$  denote  $X - \{a\}$ . Let us also denote the initial  $i$  requests of a sequence  $\sigma$  as  $\sigma^{(i)}$  and the  $i^{\text{th}}$  request as  $\sigma_i$ . For each configuration  $C$  and request sequence  $\sigma$ , the  $k$ -server *work function*  $w_\sigma(C)$  is defined as the optimal offline cost of servicing each request of  $\sigma$  sequentially, starting from the previously set initial configuration  $C_0$  and ending at configuration  $C$ . To give some examples,  $w_\emptyset(C)$  is the minimum cost for moving servers from the initial configuration  $C_0$  to the final configuration  $C$ , whereas  $w_{\sigma^{(i+1)}}(C) = w_{\sigma^{(i)}}(C)$  for  $\sigma_{i+1} \in C$ . If  $\sigma_{i+1} \notin C$  then we can think of  $w_{\sigma^{(i+1)}}(C)$  as the optimal offline cost, in which the optimal algorithm has to serve  $\sigma_{i+1}$ , before ending in  $C$ . Then we have

$$w_{\sigma^{(i+1)}}(C) = \min_{x \in C} \{w_{\sigma^{(i+1)}}(C - x + \sigma_{i+1}) + d(\sigma_{i+1}, x)\} \quad (1)$$

Here for any  $x$ ,  $w_{\sigma^{(i+1)}}(C - x + \sigma_{i+1}) + d(\sigma_{i+1}, x)$  cannot be smaller than  $w_{\sigma^{(i+1)}}(C)$ , since we can think of the above formula as the optimal cost of servicing the sequence  $\sigma^{(i+1)}$  and ending at  $C - x + \sigma_{i+1}$  plus the cost of switching to configuration  $C$ . Thus, we have the equality in (1) also for the case  $\sigma_{i+1} \in C$ , allowing us to say that (1) is the recursive formula of the work function  $w$ . On the other hand, we can rewrite the recursion as  $w_{\sigma^{(i+1)}}(C) = \min_{x \in C} \{w_{\sigma^{(i)}}(C - x + \sigma_{i+1}) + d(\sigma_{i+1}, x)\}$  since final configurations include  $\sigma_{i+1}$ . We can also have a shorthand notation for the work function, let  $w(C) = w_{\sigma^{(i)}}(C)$ . Now we can define the famous WFA.

**Algorithm WFA:** Given the next request  $\sigma_{i+1}$ , assume WFA is in configuration  $C$  after servicing  $\sigma^{(i)}$ . WFA serves the request  $\sigma_{i+1}$  with the server that minimizes the work function, *i.e.* with the server  $s \in C$  satisfying

$$s = \arg \min_{x \in C} \{w(C - x + \sigma_{i+1}) + d(\sigma_{i+1}, x)\}$$

with a random choice in the case of more than one minimizer.

We are not going to prove the  $(2k - 1)$ -competitiveness of WFA here. The proof is based on the quasi-convexity of the work functions and uses the duality lemma. Currently, the metric spaces for which WFA achieves a competitive ratio of  $k$ , are the line, metric spaces with  $k + 2$  points and the metric spaces representing the weighted paging problem. After the competitive ratio of  $2k - 1$  was proven for WFA, Koutsoupias and Papadimitriou made the WFA conjecture, a strengthening of the  $k$ -server conjecture. Simply, WFA is conjectured to be  $k$ -competitive for arbitrary metric spaces [27].

### 2.1.2 Double Coverage Algorithm

On the line, the conjecture was proven by the DOUBLE-COVERAGE algorithm [14]. This algorithm was then extended to DC-TREE algorithm [15] to prove the conjecture on the trees, but here, we are interested in the algorithm only on the line. However, the extension is somewhat intuitive. Here is the simple DC algorithm.

**Algorithm DC:** If the next request is on the left of the leftmost server or similarly on the right of the rightmost, serve it with the closest server. Otherwise, it is in between the closest two servers. In that case, move only those two closest servers, one from each side, towards the requested point at constant speed, stopping both whenever one of them serves the request.

The algorithm is simple and proves the conjecture on the line by the use of a potential function. The potential function technique is heavily used in proving competitiveness, though the technique was different in the proof of  $(2k - 1)$ -competitiveness of the WFA. For the technique used here, we have a potential function  $\Phi$ , bounded from below at all times and we guarantee a competitive ratio of  $k$ , showing that on the moves of OPT for which it pays  $c$ , the potential  $\Phi$  increases by at most  $kc$  and on the moves of DC costing  $c$ ,  $\Phi$  decreases by at least  $c$ . This is true because inductively we can prove that

$$\Phi_i - \Phi_0 \leq k \cdot \text{cost}_{\text{OPT}}(\sigma^{(i)}) - \text{cost}_{\text{DC}}(\sigma^{(i)})$$

where  $\Phi_i$  is the potential after both of the algorithms serve the  $i^{\text{th}}$  request and  $\sigma^{(i)}$  is the first  $i$  requests of the sequence  $\sigma$ . Now we can state the theorem for the DC algorithm.

**Theorem 1.** *DC is  $k$ -competitive.*

*Proof.* At any time, let  $M_{\min}$  denote the minimum cost of moving DC's servers to match the configuration of OPT, and let  $\Sigma_{\text{DC}} = \sum_{i < j} d(s_i, s_j)$  where  $s_i$ 's are the servers numbered  $1, 2, \dots, k$ . The potential function below matches the given criteria

$$\Phi = k \cdot M_{\min} + \Sigma_{\text{DC}}$$

To show that it matches the given criteria, first, we observe that  $\Phi$  is always non-negative, thus bounded from below. Second, on any move of OPT of cost  $c$ ,  $\Sigma_{\text{DC}}$  never changes and  $M_{\min}$  can increase by at most  $c$ , satisfying the criteria. Finally we look at DC's moves of cost  $c$ . There are two cases: DC moves only one server or DC moves two servers. In the first case it is easy to see that  $\Sigma_{\text{DC}}$  increases by exactly  $(k - 1)c$ , whereas  $M_{\min}$  decreases by  $c$  since we know that OPT has a server at that location and we have to make the exact move no matter what, in order to match OPT's configuration. For the second case, similarly one of the moves has to be done in order to match OPT's configuration; therefore, there is a decrease in  $M_{\min}$  by an amount of  $c/2$  (because  $c$  is the total cost). The other move may increase  $M_{\min}$  by at most the same amount, but overall,  $M_{\min}$  does not increase. Now, we know that there is no server in between the servers moving, so if we analyze  $\Sigma_{\text{DC}}$ , we can divide updated terms into three groups: ones that involve one of the servers on the right, ones that involve one of the servers on the left, and the single term that involves both of the closest servers. Since these servers move towards each other, the sum of the terms in the first group does not change. The same is true for the second group. For the third group, clearly the only term decreases by  $c$ . Therefore, we prove the theorem.  $\square$

## 2.2 The Randomized Version

The analogous  $k$ -server conjecture for the randomized problem claims that there exists a  $\Theta(\log k)$ -competitive randomized online algorithm. Yet, for arbitrary metric spaces, the best known online algorithm is the deterministic WFA. The best online algorithm randomized in nature is the HARMONIC algorithm, introduced by Raghavan and Snir [33], with a best known competitive ratio of  $O(2^k \log k)$  due to Bartal and Grove [6]. The algorithm is simple to describe:

**Algorithm HARMONIC:** For each uncovered request, serve the request with a randomly chosen server, each of which is assigned a probability inversely proportional to its distance from the requested location. Assuming that no server resides at the requested point, for each  $1 \leq i \leq k$ , server  $s_i$  is chosen with probability  $\frac{1/d_i}{\sum_{j=1}^k 1/d_j}$ , where  $d_i$  is the distance of the server  $s_i$  to the requested point.

Raghavan and Snir [33] also proved that HARMONIC has a competitive ratio of  $\frac{k(k+1)}{2}$  against a lazy adversary, and this bound is tight since there are examples on which the oblivious adversary forces HARMONIC to do  $\frac{k(k+1)}{2}$  times the offline work. Thus, HARMONIC cannot achieve a  $O(\log k)$  competitive ratio, but still it is a simple, time-efficient algorithm.

### 2.2.1 Lower Bound Analysis

The randomized  $k$ -server conjecture claims a  $\Theta(\log k)$  competitive ratio but the currently best known lower bound is  $\Omega(\log k / \log \log k)$  due to Bartal et al. [7] using the techniques in Bartal, Bollobás and Mendel [3], where they proved a lower bound of  $\Omega(\log k / \log^2 \log k)$ . This result was an improvement to the previously known competitive ratio of  $\Omega(\sqrt{\log k / \log \log k})$  by Blum et al. [10]. In the same paper improving the lower bound for the randomized  $k$ -server problem, Bartal et al. [7] also improved the lower bound for the MTS problem to  $\Omega(\log N / \log \log N)$ , which is also conjectured to be  $\Theta(\log N)$ . The proof of the newest results is based on the main theorem stated in [3] that every metric space contains a large space which is approximately an HST, a *hierarchically well-separated tree*. HSTs, introduced by Bartal [2], are simpler metric spaces for which the analysis are easier for both of the problems. The following fact together with the lower bound that they establish on HSTs help them to complete the proof: The existence of an  $r$ -competitive algorithm on a metric space  $\mathcal{M}$  implies the existence of an  $\beta r$ -competitive algorithm on a metric space  $\mathcal{N}$ , where the distances in the metric space  $\mathcal{N}$  are within a factor  $\beta$  of those in  $\mathcal{M}$ . These results are very close to the conjectured bounds but still there is an exponential gap between the best known upper and lower bounds for arbitrary metric spaces.

### 2.2.2 Results for Special Metric Spaces

The randomized  $k$ -server conjecture is solved for some special metric spaces. The first result is on the uniform metric spaces, which represents the special case of the randomized paging problem. Fiat et al. [19] show that the paging problem has an exact randomized competitive ratio of  $H_k = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k}$ , by proving both bounds. The upper bound was proven before with the use of a complicated algorithm PARTITION by McGeoch and Sleator [31], and this fact is emphasized once more with a simpler algorithm EQUITABLE by Achlioptas, Chrobak and Noga [1]. On the other hand, while proving a lower bound, Blum et al. [10] also prove the conjecture on any  $p(n)$ -unbalanced metric spaces for some polynomial  $p$ , where a  $\delta$ -unbalanced metric space is defined as a

restriction on the distances such that for any three distinct points, the ratio of the largest distance to the shortest one cannot exceed  $\delta$ . Recently, Irani [23] provides an  $O(\log k)$ -competitive algorithm for the weighted paging problem with two page weights; therefore, he extends the result for the randomized  $k$ -server conjecture.

The first important result for arbitrary metric spaces was on the MTS problem. Bartal et al. [4] presented an  $O(\log^6 N)$ -competitive algorithm which implies that on metric spaces of  $n = k + n_0$  points, there exists an  $O(n_0^6 \log^6 k)$ -competitive algorithm for the  $k$ -server problem. The main technical result of the paper is actually an  $O(\log^2 N)$ -competitive algorithm for  $\Omega(\log^2 N)$ -HST spaces, and the algorithm that achieves a  $\text{polylog}(n)$  competitive ratio that can be built using the techniques introduced by Bartal [2]. Fiat and Mendel [20], improve these results by improving the upper bound for MTS problem on the HST's to  $O(\log N \log \log N)$  thus yielding an  $O((\log N \log \log N)^2)$  competitive ratio for the MTS problem. This result is further improved to  $O((\log N)^2 \log \log N)$  with the help of the achievements in Fakcharoenphol et al. [18]. For the  $k$ -server problem, this result brings a randomized algorithm with a competitive ratio of  $O((n_0 \log k)^2 \log \log k)$  for metric spaces with  $n = k + n_0$  points. Apart from the above result conditioned on the number of points in the metric space, Seiden [34] presents an  $O(\text{polylog}(k))$  algorithm on metric spaces which can be separated into a polylogarithmic number of widely separated subspaces.

For the problem on the line, Csaba and Lodha [17] present an algorithm with a competitive ratio of  $O(n^{\frac{2}{3}} \log n)$ , which is sublinear in  $n$ , the number of points. The algorithm they describe works for the equally spaced points on the line. For a special case of 2 servers on the line, Bartal, Chrobak and Larmore [5] present an algorithm called DC2, motivated by the DC algorithm of Chrobak et al. [14]. The DC2 algorithm achieves a competitive ratio of  $\frac{155}{78} \approx 1.987$ , for the first time below 2. The algorithm is based on a variant of the  $k$ -server problem they define, the  $(k, l)$ -server problem, in which the requests must be served using  $l$  of the  $k$  servers. It is also important to mention that, the  $(k, l)$ -server problem is a special version of the fractional  $k$ -servers problem we will define in the next section, where the number of the servers in the fractional setting is the ratio  $k/l$ .

Finally, Bartal and Mendel wrote a paper [8], introducing the DEMAND-MARKING algorithm, with the intention to extend the result in [17], by Csaba and Lodha. But the description of the algorithm being imprecise together with some lemmas claiming false statements, the proof of the main result contains errors, which the authors acknowledge as well.

### 2.2.3 Demand Marking Algorithm

Bartal and Mendel try to bring new results to the randomized problem by using metric approximation techniques. They define a  $(\theta, b, t)$ -decomposable space as a metric space  $M = (V, d)$  that can be partitioned into  $t$  blocks  $B_1, \dots, B_t$  with the following properties:

- the distance between any two points in the same block is 1.
- Each block contains at most  $b$  points.
- Denote by  $\delta$  the minimum distance between two points in different blocks, and by  $\Delta$  the maximum distance between two points in different blocks. Then  $\delta \geq 1$ , and  $\theta = \frac{\Delta}{\delta}$ .

They establish connections between decomposable spaces and arbitrary metric spaces as follows: They prove that any metric  $\mathcal{M}$  can be  $O(\mu \log_\mu n)$  probabilistically approximated by a set of exact  $\mu$ -HSTs, where  $\mu = \Delta^{1/h}$  for some fixed  $h$ . Furthermore they establish that each exact  $\mu$ -HST in

their set is actually a  $(\frac{\Delta}{\mu^2}, b_M(\mu), t)$ -decomposable, for some  $t \leq n$ . Having the problem reduced to decomposable spaces, they introduce the DEMAND-MARKING algorithm for  $(\theta, b, t)$ -decomposable spaces claiming a competitive ratio of  $O(\max\{b, \theta\} \log t)$ . This result combined with the above analysis, clearly it follows that there exists randomized algorithms achieving  $O(\max\{\frac{\Delta}{\mu^2}, b_M(\mu)\} \log n)$ -competitiveness for  $\mu$ -HSTs. Using this result they obtain better bounds for growth rate bounded graphs.

We show that their definition of the DEMAND-MARKING algorithm is not precise and needs to be revised. Also there are counter examples for Lemma 20 in their paper which also invalidates Lemma 21. They state that the proof of their claim on the  $O(\max\{b, \theta\} \log t)$  competitiveness follows from Lemmas 19 and 21 but since we have counterexamples for Lemma 21, the claim is no longer proven. A trivial update to the definition and corrections to the Lemmas 20 and 21 increases the competitive ratio by a factor of  $t$ , thus the main result of getting down to logarithmic competitiveness is lost. Nevertheless, we still believe that the main result is true, but has to be proven using more complicated analysis.

### 3 The Randomized Problem with Fractional Analysis

The original  $k$ -server problem requires a deterministic choice of a single server at the time of each request, *i.e.* a deterministic algorithm is required to decide which server to move in order to serve the request. Randomized decision algorithms may potentially have much better competitive ratios, yet still no randomized algorithm has been proven to have a significantly better competitive ratio than the work function algorithm in the general case. In order to analyze the randomized problem better, we introduce a different setting. One can think of servers as fractional entities, as opposed to units. Although it is not intuitive to view servers as fractions, still it is hypothetically considerable to move fractions of servers. Then in the new setting, intuitively, we can picture the situation as breaking each server down to some portions each of which has a label, and servicing a request would require all of the portions be present at the requested location. But this is no different than visualizing a probability distribution of configurations, each of which is a set of locations of portions with a particular label. It is easy to see that any randomized algorithm is equivalent to maintaining a probability distribution over the configurations; therefore, this setting with the above requirement does not bring new features to the randomized problem except for a visualization. Instead, we can relax the requirement of serving a request to providing only a total of one unit server at the request without having labels for portions. Formally, after servicing the  $t^{\text{th}}$  request (at time  $t$ ), we can define a *weight* function  $W^t : \mathcal{M} \rightarrow \mathbb{R}$ , satisfying the following properties

1.  $W^t(x) \geq 0$  for all  $x \in \mathcal{M}$ .
2.  $W^t(\sigma_t) \geq 1$ , where  $\sigma_t$  is the request at time  $t$ .
3. The *support*  $S^t$  of the weight function  $W^t$  is finite, where  $S^t = \{x | W^t(x) > 0, x \in \mathcal{M}\}$  is the set of points with positive weight. ( $|S^t| < \infty$ )
4.  $\sum_{x \in S^t} W^t(x) = k$ .

To accommodate the initial configuration we also assume that  $W^0$  is integral. Now we can define a fractional version of the problem, for the above relaxation changes the question.

**Definition 4 (Fractional  $k$ -Server Problem).** *The fractional  $k$ -server problem consists of a metric space  $\mathcal{M}$  and a total of  $k$  fractional servers located at the points of the metric space as indicated by a weight function  $W$ . Given a sequence of requests, each request must be served by providing one unit server at the point of request, *i.e.* the weight function  $W$  must be updated accordingly to satisfy the above properties by moving fractional servers to the request. The cost of an algorithm for servicing a sequence of requests is the cumulative weighted sum of the cost incurred by each server, where moving a  $w$  fraction of a server for a distance of  $\delta$  costs  $w\delta$ .*

To help our analysis, one can also define the fractional version of the  $l$ -evader problem as well. Simply, evaders can move fractionally, and whenever a location needs to be emptied, all we need to do is to move the evader away from that location to other locations provided that at the end no location will hold more than one unit evader. Analyzing the randomized problem, the introduction of the fractional setting gives rise to a natural question: Is the fractional version equivalent to the randomized version? On the one hand, it is easy to simulate any randomized algorithm by a fractional algorithm; however, the converse is not quite true. On the line and circle, we show that these problems are equivalent. We also prove the same result when  $k = 2$  or  $k = n - 1$  for any  $n$  point metric space. However, it is surprising to see that for other metric spaces there are fractional algorithms which cannot be simulated by randomized algorithms.

### 3.1 Simulation of Randomized Algorithms

Starting with the randomized setting, we mentioned before that any randomized algorithm is equivalent to maintaining a probability distribution over the configurations. For a randomized algorithm  $\text{ALG}$ , let  $P_{\text{ALG}}^t(C)$  denote the probability of being in a configuration  $C$  at time  $t$ . Concurrently, we can define a fractional algorithm  $\text{ALG}'$  for  $\text{ALG}$ . After servicing the  $t^{\text{th}}$  request, define the total weight at each location  $x$  as

$$W_{\text{ALG}'}^t(x) = \sum_{C, x \in C} P_{\text{ALG}}^t(C) \quad (2)$$

To complete the description of  $\text{ALG}'$ , for shifting from one weight distribution to another, let  $\text{ALG}'$  move servers according to the minimum cost shift between those two distributions. By the following lemma, we show that the intuitive translation actually satisfies.

**Lemma 1.** *For any sequence of requests  $\sigma$ ,  $\text{cost}_{\text{ALG}'}(\sigma) \leq E[\text{cost}_{\text{ALG}}(\sigma)]$ .*

*Proof.* Suppose that, at time  $t$  being in a configuration  $C$  with probability  $p$ ,  $\text{ALG}$  shifts to configurations  $C_1, C_2, \dots, C_m$  with probabilities  $s_1, s_2, \dots, s_m$  for servicing the next request. Starting with a weight distribution  $W_{\text{ALG}'}^t$  defined as above, we can simulate this shift at the same cost by moving the representative servers of  $C$  as follows: For every  $i = 1, 2, \dots, m$  and  $x \in C$  move  $p \times s_i$  fraction from  $x$  to the point  $x'_i$ , where  $x'_i$  is the point to which the server at  $x$  moves while satisfying the shift from  $C$  to  $C_i$ . One can easily see that, the contribution of these moves to the expectation of the cost of the randomized algorithm  $\text{ALG}$  is exactly the same cost of moving the specified fractions between those locations for the fractional algorithm  $\text{ALG}'$ . It is also clear that after servicing the next request according to the fractional moves as defined above,  $\text{ALG}'$  preserves the predefined weight distribution at time  $t + 1$ . Thus, there is a possible shift of the weights, which in turn allows  $\text{ALG}'$  to have the same cost as  $\text{ALG}$  pays in expectation. Since  $\text{ALG}'$  moves servers according to the minimum cost shift between the distributions, it possibly attains a lower cost, but definitely not a higher one.  $\square$

With the definition of  $ALG'$ , we introduce the notion of simulation between these two types of algorithms. It is easy to observe that, in order to have the same exact effects, we need to have the equality in (2) satisfied for any  $t = 1, 2, \dots, |\sigma|$ , for each request of the sequence  $\sigma$ . The above lemma proves that this simulation is possible without an extra cost, that is to say, fractional algorithms are generalized versions of randomized algorithms.

### 3.2 Equivalence on the Line and Circle

Motivated by the similarities between the fractional and the randomized versions, one can hope that any fractional algorithm would be simulated by a randomized algorithm. Indeed, this is true on some metric spaces, the line and circle, and also on some cases depending on the number of servers, when  $k = 2$  or  $k = n - 1$ . Then, we can state our main result with the help of the following lemmas proving that the simulation is possible for the above cases.

**Theorem 2.** *The fractional  $k$ -server problem is equivalent to the randomized  $k$ -server problem on the line or circle, or if  $k = 2$  or  $k = n - 1$  for arbitrary metric spaces.*

The proof of the theorem is evident with the following lemmas 2, 3 and 4.

**Lemma 2.** *Any fractional algorithm on the line or circle can be simulated by a randomized algorithm at the same cost.*

*Proof.* First, given an instance of a fractional algorithm on the line, we will show how to represent the same instance in terms of a probability distribution over configurations, and how to update that probability distribution when the weight distribution changes. Then, we will show how to modify the same technique to be able to extend the result to the circle. Starting with the line, we can define an elementary move as the shift of some fractional server from one location to another where there are no servers in between. It is easy to see that the elementary moves span the set of all possible movements. Therefore, if we can perform the simulation on elementary moves at the same cost, we can simulate any move without an extra cost. Let  $W(x)$  denote the weight at a location  $x$  at an instance and suppose we are moving a  $w$  fraction of the server from  $x_0$  to  $x'_0$ . Supposing the move is elementary, we approve that there are no servers between  $x_0$  and  $x'_0$ . Let the resulting weight distribution be  $W'(x)$ , then  $W'(x) = W(x)$  for all  $x \neq x_0$  or  $x'_0$  and for those points we have  $W'(x_0) = W(x_0) - w$  and  $W'(x'_0) = W(x'_0) + w$ . This elementary move has a cost of  $w d(x_0, x'_0)$  in the fractional setting. Let  $S$  be the support function of a weight distribution defined similarly,  $S(W) = \{x | W(x) > 0\}$ . Clearly, even if the metric space  $\mathcal{M} \subset \mathbb{R}$  is infinite,  $S(W)$  will be finite since at any instance, there will be a finite number of requests thus far. We can define a function  $f : \mathbb{R}^{\mathcal{M}} \times [0, k) \rightarrow \mathcal{M}$

$$f(W, \delta) = \min\{y \in S(W) \mid \sum_{\substack{x \in S(W) \\ x \leq y}} W(x) > \delta\}.$$

We can regard  $f$  as a locator to spot the server at offset  $\delta$ , the server just after a total of  $\delta$  weight from left. Now we can define a randomized algorithm to simulate the same elementary move from  $x_0$  to  $x'_0$ . Let us develop a procedure to build configurations by defining another function  $g : \mathbb{R}^{\mathcal{M}} \times [0, 1) \rightarrow \{C | C \subset \mathcal{M}, |C| = k\}$ , from a similar domain except for the offset, which is now  $[0, 1)$ , to the set of configurations. Given an offset  $\delta \in [0, 1)$ , let  $g(W, \delta)$  be the configuration

consisting of  $f(W, \delta), f(W, 1+\delta), \dots, f(W, k-1+\delta)$ . We can also call  $g(W, \delta)$  as the configuration at offset  $\delta$ , according to the weight distribution  $W$ . We can assign probabilities to the configurations in  $\mathcal{C} = \{g(W, \delta) | \delta \in [0, 1)\}$  with each configuration  $C \in \mathcal{C}$  having a probability equal to the length of the interval  $\{\delta \in [0, 1) | g(W, \delta) = C\}$ . After the transfer of the weight from  $x_0$  to  $x'_0$ , configurations at some particular offsets will shift their servers from  $x_0$  to  $x'_0$ . But clearly, the total weight of configurations which shifted their servers will be the weight transferred from  $x_0$  to  $x'_0$ , which is  $w$ . Hence, we have a shift with probability  $w$  and the shift costs the same for every configuration, the distance between  $x_0$  and  $x'_0$ , which is  $d(x_0, x'_0)$ . Thus, the cost of this elementary move in expectation is  $wd(x_0, x'_0)$ , the same cost incurred by the fractional algorithm. This argument completes the proof on the simulation of the moves on the line. Finally, for each request, we need to have every configuration include the requested point. For this matter, we observe that any fractional algorithm maintains at least one unit server at that location, which suggests that for any  $\delta$ , the requested point will be included in the configuration at the offset  $\delta$ .

For the circle, we cut the circle at a location to form a line segment of length  $\ell$ , and place the servers on this line segment. We extend this segment to the real line by copying the same line segment (without servers) to both ends infinitely many times. Now we have an analogous version of the problem on the line. For each move on the circle, we do the simulation by moving an equivalent server on the line. With this simulation, we can assume that the problem is defined on the line, except for a difference that the requests are over a set of equivalent points instead of one point. But it does not affect our analysis; therefore, we can apply the same technique directly for the proof of the simulation.  $\square$

The above technique actually provides labels for portions of servers, with these labels being the offsets in  $[0, 1)$ ; hence, the previous discussion about the formulation of the fractional version informally proves the lemma. On the other hand, it is not evident that we can use the same technique for other metric spaces. However, for the other cases for which the simulation is possible, we will be able to find labels for each portion using a different schema.

**Lemma 3.** *If the number of servers is 2, we can simulate any fractional algorithm on any metric space by a randomized algorithm at the same cost.*

*Proof.* For the case of fractional 2-server problem, one important fact to note will be that, after each request, the point of request will have one weight, leaving a maximum of one unit server to be fractioned. Denoting the request at time  $t$  by  $\sigma_t$ , for any fractional algorithm ALG, we will have  $W_{\text{ALG}}^t(\sigma_t) = 1$ , assuming that ALG behaves lazily. Then we define the corresponding configurations and probability distribution among them as follows: At time  $t$ , every configuration will have a server at  $\sigma_t$ , and a server at location  $x$  with probability  $W_{\text{ALG}}^t(x)$ . For the next request, assuming that ALG moves  $M_{\text{ALG}}^t(x)$  fraction from  $x$  to the requested point  $\sigma_{t+1}$ , we have  $\sum_x M_{\text{ALG}}^t(x) = 1$  together with  $M_{\text{ALG}}^t(\sigma_{t+1}) = W_{\text{ALG}}^t(\sigma_{t+1})$  and the interpretation is that weight at the requested locations stays there. We can now define our randomized algorithm ALG' as follows: Being in a configuration  $C^t(x) = \{x, \sigma_t\}$ , we move to configuration  $C^{t+1}(\sigma_t)$  with probability  $M_{\text{ALG}}^t(x)/W_{\text{ALG}}^t(x)$ , and otherwise we move to configuration  $C^{t+1}(x)$ . One can easily check that the shift of the distribution of configurations defined as above agrees with the predefined distribution in terms of the weight distribution  $W_{\text{ALG}}^{t+1}$ . Furthermore the expected cost of ALG' for a request is the same cost incurred by ALG for the same request. Hence, we have the desired conclusion.  $\square$

**Lemma 4.** *The fractional 1-evader problem is equivalent to the randomized 1-evader problem on any metric space.*

*Proof.* The fractional simulation of any randomized 1-evader algorithm follows from lemma 1, thus all we need to prove is that, any fractional algorithm using one evader can be simulated by a randomized algorithm at the same cost. In the case of one evader, the only needed move is to empty the location requested without worrying about the capacity restriction. For a randomized simulation, basically, we define the probability distribution over the evader locations to be exactly the weight distribution of the only evader in the fractional problem. Simply, we move the evader away from the requested point to the other points of the metric space with probabilities proportional to the fractions shifted by the fractional algorithm. It is straightforward to see that the costs are the same, thus equivalence is evident.  $\square$

Showing that the set of online algorithms is the same for both of the settings, we are able to deduce that the fractional problem is equivalent to the randomized problem for the above cases; consequently, the proof of our theorem is complete.

For the above cases we have the equivalence; however, for the other cases we show that exact simulation is not possible at the same cost. We can distinguish these cases by some restrictions on the metric space and on the number of servers. First of all, the graph representing the metric space should contain a node of degree at least 3. With this restriction we eliminate the metric space line and the circle. Secondly, together with  $k > 2$ , we need  $n > k + 1$  to eliminate the equivalences. The above restrictions are necessary and sufficient in order to be able to describe all other cases. Then, one can analyze a local behaviour of a fractional algorithm on such metric spaces. We can consider a graph of 5 nodes with 3 servers, with the graph being different than a line graph. We prove that exact simulation is not possible for these localized problems by the help of the examples shown in figures 1 and 2, allowing us to state the following theorem:

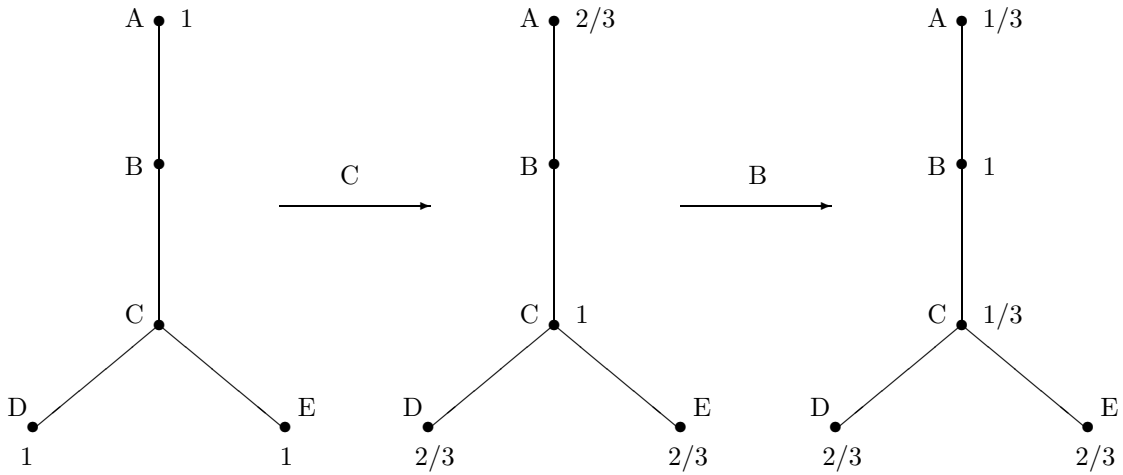


Figure 1: A fractional algorithm on a simple tree

**Theorem 3.** *Apart from the line and the circle, when  $k > 2$  and  $n > k + 1$ , there exist fractional algorithms that cannot be simulated by any randomized algorithm without an extra cost.*

*Proof.* We will analyze the local behaviour as mentioned above, using the two types of graphs shown in the figures. The example shown above in figure 1 provides a fractional algorithm for the request sequence  $C, B$  starting with the initial configuration  $\{A, D, E\}$ . It is enough to prove that no randomized algorithm can simulate this fractional algorithm on the request sequence  $C, B$  to accomodate both of the next possible requests at locations  $D$  and  $E$  without an extra cost. The proof is based on the following difference between two types of algorithms: Any randomized algorithm, after each request, has to specify a probability distribution of the configurations explicitly, whereas fractional algorithms just establish a total weight at a location, allowing the breakdown of the weights to be postponed and resolved at a later stage of the request sequence. For this specific example, after servicing the first request  $C$ , there are 3 different configurations, each with  $1/3$  probability:  $\{C, D, E\}, \{A, C, D\}, \{A, C, E\}$ . Having the next request at  $B$ , it is clear that  $\{C, D, E\}$  has to shift to  $\{B, D, E\}$ , while both of the other configurations have two options to serve the request, moving the server at  $A$  or the server at  $C$ . For configurations  $\{A, C, D\}$  and  $\{A, C, E\}$ , let the probabilities to move from  $A$  be  $p$  and  $q$  respectively. In order to have the right simulation, we need to have  $p + q = 1$ . For the third request, suppose that the fractional algorithm moves the server at  $C$  to the point of request. Clearly, in return, any configuration in the randomized setting should be able to make the same move. If the last request is  $E$ , then for the configuration  $\{A, B, D\}$ , which has  $(1 - p)/3$  probability in the distribution, we must make an extra move. But this will bring an extra cost, so the only solution to simulate without an extra cost is to preset zero probability for that configuration, which suggests  $p = 1$ . Using similar arguments, we also get  $q = 1$ , and therefore the contradiction. Having showed that the simulation is not possible for the above metric space, the proof requires a little longer sequence  $B, C, B$ , to be able to say that no randomized algorithm can simulate the fractional algorithm shown below in figure 2 on a star graph. Again there are five possible configurations  $\{B, D, E\}, \{A, B, D\}, \{A, B, E\}, \{B, C, D\}$  and  $\{B, C, E\}$  with probabilities  $1/3, p, 1/3 - p, q$  and  $1/3 - q$  respectively, together with  $p + q = 1$ . Similar arguments follow for the next two possible requests  $D$  and  $E$ .  $\square$

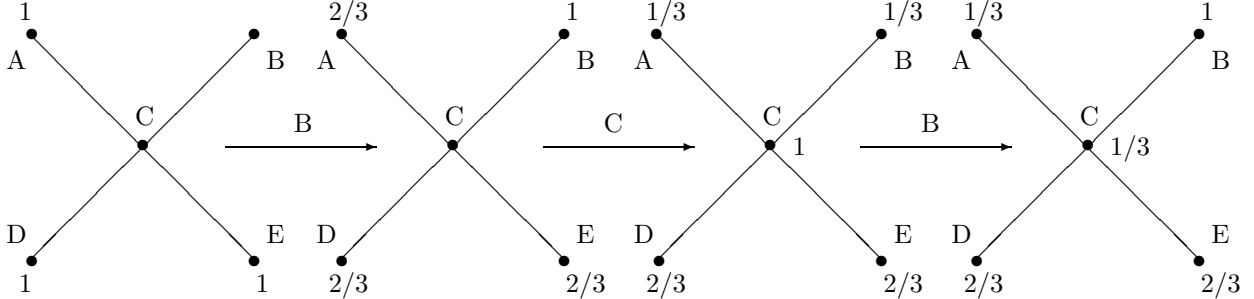


Figure 2: A fractional algorithm on a star graph

### 3.3 Remarks

We have analyzed the equivalence of both versions and characterized the cases for which they are equivalent and for which they are not. For the cases for which they are equivalent, the fractional movement ability of the servers surely brings a new deterministic technique to attack the randomized problem. This technique is deterministic since any randomization of the fractional movements can

be averaged to yield only one fractional behaviour which will cost exactly the same. On the other hand, for the cases for which these versions are not equivalent, we still do not know if fractional algorithms perform better by attaining lower costs. Actually we do not measure the performance of algorithms by measuring the actual costs, instead, we compare the algorithms by their competitive ratios. But, the competitive analysis is the way to measure the performance of an algorithm against the best offline algorithm of the same kind, which will be the best offline fractional algorithm in this case.

For the randomized case, the best deterministic offline algorithms are actually the best randomized offline algorithms. To prove this fact, one can realize that each path of random choices can be seen as a deterministic path, and among many deterministic paths there is one that minimizes the cost. The path with the minimum cost, in other words the best deterministic path definitely does not do worse than the overall randomized algorithm in expectation. But for the fractional case, we do not know whether there are better offline fractional algorithms than the ones that always move the servers as a unit, or not. Here, we conjecture that the cost of the best offline fractional algorithm would not increase when we restrict ourselves to moving the servers as a unit. Our conjecture is already established in the cases for which we have proved the equivalence of the fractional and the randomized versions. Whether our conjecture is true in general or not, we shall introduce the usual competitive analysis for the fractional problem, in which we similarly define  $r$ -competitiveness, competitive ratio of an algorithm and the competitive ratio of the fractional problem in comparison with the best offline fractional algorithm.

If we can prove our conjecture, then we can establish a fact that suggests the randomized competitive ratio is bounded from below by the fractional competitive ratio. But for the moment, since the best offline fractional algorithm may achieve better results by moving servers fractionally, the fractional competitive ratio may even be greater than the randomized competitive ratio.

## 4 Analysis on the Line

The results obtained for the randomized case is not as satisfying as the deterministic case except for the paging problem and some special metric spaces. Even for the line, there is an exponential gap between the lower and the upper bounds. With the equivalence of both versions, the randomized and the fractional, we can analyze the problem on the line using the advantages of the fractional setting. The fractional setting provides an overall picture of the servers, instead of only having the detailed view of the configurations. What we are interested in is the expected cost and using the fractional setting we can make “expected” moves in the overall picture by the fractional movement ability of the servers. In the rest of the paper, we develop and analyze a randomized version of the DC algorithm introduced in [14], and then focus on the even more seemingly simpler problem of two servers.

### 4.1 Fractional Double Coverage Algorithm

In the survey of the  $k$ -server problem, we presented a proof for the DC algorithm in the deterministic setting showing that DC is  $k$ -competitive, which is the best one can hope for. We think that the main idea of the DC algorithm is to behave greedily and not to move away from OPT’s configuration. In achieving this, DC always moves two units of servers instead of one. With the fractional setting though, we will be able to move only one unit server at a time and yet manage not to move away

from OPT's configuration by developing a fractional double coverage algorithm which in practice does not achieve better bounds, but still worth analyzing.

**Algorithm FDC:** If there are no servers on the left or on the right of the next request, serve it with the closest server. Otherwise, it is in between the closest two servers, without considering the possible server at the request. In that case, move some fraction of servers from both of the locations inversely proportional to their distances, in order to provide a maximum total weight of one at the request. If one of the locations gets depleted, and the request has not been served yet, apply the same procedure again.

**Lemma 5.** *FDC is  $2k$ -competitive.*

*Proof.* The proof technique is the same. Let  $M_{min}$  denote the minimum cost of moving FDC's servers to match the configuration of OPT. Let  $\Sigma_{FDC}$  denote the weighted sum of all interpoint distances between the fractional servers of FDC:  $\Sigma_{FDC} = \sum_{x,x' \in S(W)} W(x)W(x')d(x,x')$ , where  $x, x'$  refer to points of the line,  $W(x), W(x')$  weights at those locations and  $S$  the support function for a weight distribution. The following potential function helps to prove  $2k$ -competitiveness:

$$\Phi = 2k.M_{min} + \Sigma_{FDC}$$

Since  $\Phi$  is nonnegative, bounded below, if we can prove that adversary's any move of distance  $c$  cannot increase  $\Phi$  more than  $2kc$  while FDC's move of cost  $c$  decreases the potential by at least  $c$ , then we are done. The first part is easy since any move of the adversary cannot change the second term of the potential and it can increase  $M_{min}$  by at most  $c$  that would increase the potential not more than  $2kc$ . For the second part, we analyze the whole move for a request by breaking it up into phases, which consist of moves of the procedure described in the algorithm until one of the servers get depleted or the request satisfied.

For an individual phase either of the following cases can happen: there is only one server moving or there are two of them. In the first case, we can definitely say that the move of cost  $c$  decreases  $M_{min}$  by  $c$ . On the other hand, this move cannot increase  $\Sigma_{FDC}$  by more than  $2(k-1)c$ . To prove this claim, we consider the overall effect after the request is served; the increasing effect of this move is on the other  $k-1$  weight which is not serving the request, cumulating to a total increase of  $2(k-1)w_0d_0$ , which is  $2(k-1)c$ . Then, regardless of any other causes to decrease the sum, we can say that  $\Phi$  decreases at least by  $2c$  for the first case. For the second case, a similar argument used in the proof of the  $k$ -competitiveness of the DC algorithm is useful: Since we move servers from both sides with the same cost  $c/2$ , while one of the moves decreases  $M_{min}$  by  $c/2$ , the other cannot increase by more than  $c/2$ . Therefore  $M_{min}$  does not increase, whereas we will show that  $\Sigma_{FDC}$  decreases by at least  $c$ . Before moving any server for the current request  $\sigma_t$ , let  $w_0 = W(\sigma_t)$  and to serve the request, assume the amount of fraction we are going to move from left is  $w_L$  and from right is  $w_R$ . Clearly  $w_L + w_0 + w_R = 1$  since we have to serve the request. Now if we analyze the current situation, we move two fractions, one from left, say an amount of  $w_\ell$  for a distance of  $d_\ell$ , and one from right, again say an amount of  $w_r$  for a distance of  $d_r$ . By definition we have the equalities  $w_\ell d_\ell = w_r d_r = c/2$ . Using the same arguments of the proof for DC, for the servers that does not serve the request, *i.e.* those which stay at their locations, the cumulative effect of  $w_\ell$  and  $w_r$  on the terms of  $\Sigma_{FDC}$  involving those servers is zero. On the other hand, we can analyze the effect within the fractions that serve the request. For  $w_r$  moving to the left by  $d_r$ , clearly the summation decreases by  $2(w_0 + w_L)w_r d_r$  and we may assume that we do not increase the summation by moving

away from the servers on the right hand side, since they will meet at the request later. Similarly, we have a decrease in the amount of  $2(w_0 + w_R)w_L d_\ell$ , for the server moving from left. Thus the total decrease is at least  $2(w_0 + w_L)c/2 + 2(w_0 + w_R)c/2 \geq (w_L + w_0 + w_R)c = c$ , concluding the analysis for the second case. Hence, using the potential function technique, we prove that FDC is  $2k$ -competitive.  $\square$

Although we were not able to prove a better competitive ratio, we conjecture that FDC is  $k$ -competitive and we already know that this bound is tight, since even for  $k + 1$  points there exists a series of sequences for which FDC has a competitive ratio of  $k$  in the limit. Hence, for the moment, it means that we cannot go beyond the best results obtained for the deterministic case. However, when we focus on the even more simpler case of two servers, for a set of metric spaces we develop an algorithm which provides both upper and lower bounds for the 2-server problem.

## 4.2 Special Case: 2-Server Problem

One can think that the problem of two servers is easy, furthermore if the servers are on the line then even trivial. However, the randomized problem of two servers even on a metric space of three points has not been solved yet. The first important result for the 2-server problem came for the lower bound analysis in the paper of Karlin et al. [24], proving that on an isosceles triangle there is a lower bound of  $e/(e - 1)$  in the limit. This lower bound was improved to  $1 + e^{-1/2}$  by Chrobak et al. [16], by providing a sequence of requests on a metric space of four points. On the other hand, Bartal, Chrobak and Larmore [5] improved the upper bound to  $\frac{155}{78} \approx 1.987$  as we mentioned earlier. Since then the gap for the problem has remained the same. In this section, we provide an algorithm for two servers on three point metric spaces similar to the algorithm in [24] which establish similar lower and upper bounds. Before going into the definition, we need to define introduce concepts.

**Definition 5 (Dominating Configuration).** For a sequence of requests  $\sigma$ , a configuration  $C$  is called *dominating* if all other possible choices of serving the sequence ending in a different configuration  $C'$  costs not less than the cost incurred by the version that ends in  $C$  plus the minimum cost matching  $C$  and  $C'$ .

**Lemma 6.** *If there is a dominating configuration  $C$  for a sequence  $\sigma$  of requests, for all sequences  $\sigma'$  with an initial segment of  $\sigma$ ,  $OPT$  will not do worse by being in  $C$  after serving  $\sigma$ .*

*Proof.* Assume that  $OPT$  starting from  $C_0$ , the initial configuration, moves to  $C_i$  after serving  $\sigma'_i$ ,  $i^{th}$  request and assume  $C_r \neq C$  where  $r = |\sigma|$ . Now we can consider a different path of configurations starting from  $C_0$ , that leads to  $C$  after serving the sequence  $\sigma$ , then jumping to  $C_r$  and simulating  $OPT$  for the rest of the sequence  $\sigma'$ . The cost after jumping to  $C_r$  will be exactly equal to the cost of  $OPT$ 's movements for that same phase. For the initial phase, by the definition of the dominating configuration, we have at most the same cost of  $OPT$ , yielding us the desired result.  $\square$

**Lemma 7.** *If there exists a dominating configuration  $C$  for a sequence of requests  $\sigma$ , in terms of competitiveness any algorithm  $Alg$  would not do worse by shifting to that configuration, after serving the requests.*

*Proof.* For any sequence  $\sigma'$  with an initial segment of  $\sigma$  define a new sequence  $\sigma'_C$  as follows: Keep the initial segment, concatenate the sequence of requests  $C_1, C_2, \dots, C_k$  many times to ensure that

$Alg$  will move to configuration  $C$ , then add the rest of the sequence  $\sigma'$ . Since  $C$  is a dominating configuration  $OPT$  will not be incurring any cost due to the requests at locations of configuration  $C$ . On the other hand, after the sequence  $\sigma$  let  $Alg$  be in configuration  $C'$ , on the second phase of the new sequence  $Alg$  will be having a cost at least the cost matching between  $C'$  and  $C$ . Since the cost of  $OPT$  does not change for both of the sequences  $\sigma'$  and  $\sigma'_C$ ,  $Alg$  will be as competitive as before if it moves from  $C'$  to  $C$ , just after serving last request of  $\sigma$ .  $\square$

For a metric space consisting of three points  $A, B$  and  $C$ , with distances  $d(A, B) = 1$  and  $d(B, C) = d \in \mathbb{Z}$ , we will define a fractional algorithm  $2SERVER$  which is approximately  $e/(e-1)$ -competitive. To be exact we call the ratio  $\alpha_d$ , which is  $\frac{e_d - \frac{1}{d+1}}{e_d - 1}$  where  $e_d = (\frac{d+1}{d})^d$ .

**Algorithm  $2SERVER$ :** Given any sequence of requests we will mark phases within the sequence: A phase either consists of  $d$  consecutive  $BA$ 's or some initial part of the full phase with a request at  $C$  at the end. Assuming that the initial configuration is  $(A, C)$ , any requests at  $A$  or  $C$  would not have an effect. So we can assume the sequence starts with a phase. At the end of each phase there will be a tail of requests before the next phase leading us to a dominating configuration of  $(A, C)$ . Our algorithm is simple, it establishes  $w_i = \alpha_d - (\alpha_d - 1)(\frac{d+1}{d})^i$  weight at location  $C$  as a result of serving the  $i^{th}$   $B$  of the phase. At the end of each phase we move one unit server to two locations requested finally for the phase.

**Lemma 8.** *At the end of each phase there is a dominating configuration, which consists of last two requests of the phase.*

*Proof.* Each phase begins with the dominating configuration  $(A, C)$  and for  $OPT$ , the first choice to serve  $B$ , is moving from either  $C$  or  $A$ . If the first choice is not moving from  $C$ , it will not be profitable to move from  $C$  ever again, till the end of the phase. Thus there are exactly two choices to serve the whole phase: First is to move from  $C$ , and back to  $C$  at the end of the phase if requested, and second is to shuttle the server at  $A$  between  $A$  and  $B$ . For each type of phase, one can easily compare the costs of both choices and figure out that last two requests define the dominating configuration.  $\square$

**Theorem 4.**  *$2SERVER$  is  $\alpha_d$ -competitive.*

*Proof.* It is enough to show that the algorithm is  $\alpha_d$  competitive in each of the phases. We can analyse the claim on phases where the dominating configuration is  $(A, B)$ , i.e. the phase consists of  $d$  consecutive  $BA$ 's, and where it is not the case. The offline cost of serving the first type of phases is  $d$ . Our cost after serving the phase will be the total fraction of servers shuttling between  $A$  and  $B$  together with a cost of  $d$  for moving all of the fractions from  $C$  to  $B$ . So the total cost is  $d + \sum_{a=1}^d 2w_a$ . There will be an extra cost of  $d$  to go back to the initial configuration thus the ratio will be  $(2d + \sum_{a=1}^d 2w_a)/2d$  which is exactly  $\alpha_d$ . For the second type of the phases, we only need to analyse the cases where the phase ends at a dominating configuration of  $(B, C)$ , since otherwise, our cost is not more and  $OPT$  has the same cost, yielding a lower ratio. Assume there are  $i$  number of  $B$ 's before  $C$ , therefore the cost of  $OPT$  is  $2i - 1$  and we will have plus 1 for moving back to the initial configuration thus  $2i$ . Except for the last request, our cost is the total shuttling fraction of the servers between  $A$  and  $B$  plus  $d$  times the fraction of server moved from  $C$  to  $B$ . Last request costs us moving all of the server at  $A$  to  $C$ . Therefore totally our cost is

$\sum_{a=1}^{i-1} 2w_a + w_i + d(1 - w_i) + (d + 1)(1 - w_i)$ . When we calculate the ratio we get the same result  $\alpha_d$  which completes the proof.  $\square$

It is straightforward to see that  $\alpha_d$  is less than  $e/(e - 1)$  for all  $d$ , and in the limit as  $d$  goes to infinity we have the equality.

## References

- [1] Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theor. Comput. Sci.*, 234(1-2):203–218, 2000.
- [2] Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *FOCS '96: Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, page 184. IEEE Computer Society, 1996.
- [3] Y. Bartal, B. Bollobás, and M. Mendel. A ramsey-type theorem for metric spaces and its applications for metrical task systems and related problems. In *FOCS '01: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, page 396. IEEE Computer Society, 2001.
- [4] Yair Bartal, Avrim Blum, Carl Burch, and Andrew Tomkins. A polylog(n)-competitive algorithm for metrical task systems. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 711–719. ACM Press, 1997.
- [5] Yair Bartal, Marek Chrobak, and Lawrence L. Larmore. A randomized algorithm for two servers on the line. *Inf. Comput.*, 158(1):53–69, 2000.
- [6] Yair Bartal and Eddie Grove. The harmonic k-server algorithm is competitive. *J. ACM*, 47(1):1–15, 2000.
- [7] Yair Bartal, Nathan Linial, Manor Mendel, and Assaf Naor. On metric ramsey-type phenomena. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 463–472. ACM Press, 2003.
- [8] Yair Bartal and Manor Mendel. Randomized k-server algorithms for growth-rate bounded graphs. *J. Algorithms*, 55(2):192–202, 2005.
- [9] Yair Bartal and Adi Rosén. The distributed k-server problem - competitive distributed translator for k-server algorithms. *J. Algorithms*, 23(2):241–264, 1997.
- [10] Avrim Blum, Howard Karloff, Yuval Rabani, and Michael Saks. A decomposition theorem for task systems and bounds for randomized server problems. *SIAM J. Comput.*, 30(5):1624–1661, 2000.
- [11] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
- [12] Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992.

- [13] Moses Charikar, Dan Halperin, and Rajeev Motwani. The dynamic servers problem. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 410–419. Society for Industrial and Applied Mathematics, 1998.
- [14] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. In *SODA '90: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 291–300. Society for Industrial and Applied Mathematics, 1990.
- [15] Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM J. Comput.*, 20(1):144–148, 1991.
- [16] Marek Chrobak, Lawrence L. Larmore, Carsten Lund, and Nick Reingold. A better lower bound on the competitive ratio of the randomized 2-server problem. *Inf. Process. Lett.*, 63(2):79–83, 1997.
- [17] B. Csaba and S. Lodha. A randomized on-line algorithm for the k-server problem on a line. Technical Report 2001-34, DIMACS, 2001.
- [18] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 448–455. ACM Press, 2003.
- [19] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel D. Sleator, and Neal E. Young. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991.
- [20] Amos Fiat and Manor Mendel. Better algorithms for unfair metrical task systems and applications. *SIAM J. Comput.*, 32(6):1403–1422, 2003.
- [21] Amos Fiat and Moty Ricklin. Competitive algorithms for the weighted server problem. *Theor. Comput. Sci.*, 130(1):85–99, 1994.
- [22] E. F. Grove. The harmonic online k-server algorithm is competitive. In *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 260–266. ACM Press, 1991.
- [23] Sandy Irani. Randomized weighted caching with two page weights. *Algorithmica*, 32(4):624–640, 2002.
- [24] Anna R. Karlin, Mark S. Manasse, Lyle A. McGeoch, and Susan Owicki. Competitive randomized algorithms for non-uniform problems. In *SODA '90: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 301–309. Society for Industrial and Applied Mathematics, 1990.
- [25] Elias Koutsoupias. *On-line algorithms and the K-server conjecture*. PhD thesis, 1995.
- [26] Elias Koutsoupias and Christos Papadimitriou. The 2-evader problem. *Inf. Process. Lett.*, 57(5):249–252, 1996.
- [27] Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42(5):971–983, 1995.

- [28] Elias Koutsoupias and David Scot Taylor. The cnn problem and other k-server variants. *Theor. Comput. Sci.*, 324(2-3):347–359, 2004.
- [29] Mark S. Manasse, Lyle A. McGeoch, and Daniel D. Sleator. Competitive algorithms for on-line problems. In *STOC '88: Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 322–333, 1988.
- [30] Mark S. Manasse, Lyle A. McGeoch, and Daniel D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990.
- [31] Lyle A. McGeoch and Daniel Dominic Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991.
- [32] Enoch Peserico. The lazy adversary conjecture fails. In *SPAA '02: Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, pages 143–144. ACM Press, 2002.
- [33] Prabhakar Raghavan and Marc Snir. Memory versus randomization in on-line algorithms (extended abstract). In *ICALP '89: Proceedings of the 16th International Colloquium on Automata, Languages and Programming*, pages 687–703. Springer-Verlag, 1989.
- [34] Steven S. Seiden. A general decomposition theorem for the k-server problem. *Inf. Comput.*, 174(2):193–202, 2002.
- [35] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.