

# Improved Online Algorithms for the Sorting Buffer Problem\*

Iftah Gamzu<sup>†</sup>

Danny Segev<sup>‡</sup>

## Abstract

An instance of the *sorting buffer* problem consists of a metric space and a server, equipped with a finite-capacity buffer capable of holding a limited number of requests. An additional ingredient of the input is an online sequence of requests, each of which is characterized by a destination in the given metric; whenever a request arrives, it must be stored in the sorting buffer. At any point in time, a currently pending request can be served by drawing it out of the buffer and moving the server to its corresponding destination. The objective is to serve all input requests in a way that minimizes the total distance traveled by the server.

In this paper, we focus our attention on instances of the problem in which the underlying metric is either an *evenly-spaced line metric* or a *continuous line metric*. Although such restricted settings may appear to be very simple at first glance, we demonstrate that they still capture one of the most fundamental problems in the design of storage systems, known as the *disk arm scheduling* problem. Our main findings can be briefly summarized as follows:

1. We present a *deterministic*  $O(\log n)$ -competitive algorithm for  $n$ -point evenly-spaced line metrics. This result improves on a randomized  $O(\log^2 n)$ -competitive algorithm due to Khandekar and Pandit (STACS '06). It also refutes their conjecture, stating that a deterministic strategy is unlikely to obtain a non-trivial competitive ratio.
2. We devise a *deterministic*  $O(\log N \log \log N)$ -competitive algorithm for continuous line metrics, where  $N$  denotes the length of the input sequence. In this context, we introduce a novel discretization technique, which is of independent interest, as it may be applicable in other settings as well.
3. We establish the first non-trivial lower bound for the evenly-spaced case, by proving that the competitive ratio of any deterministic algorithm is at least  $\frac{2+\sqrt{3}}{\sqrt{3}} \approx 2.154$ . This result settles, to some extent, an open question due to Khandekar and Pandit (STACS '06), who posed the task of attaining lower bounds on the achievable competitive ratio as a foundational objective for future research.

---

\*An extended abstract of this paper appeared in *Proceedings of the 24th International Symposium on Theoretical Aspects of Computer Science*, pages 658–669, 2007.

<sup>†</sup>School of Computer Science, Tel-Aviv University, Israel. Email: [iftgam@post.tau.ac.il](mailto:iftgam@post.tau.ac.il). Supported in part by the German-Israeli Foundation and by the Israel Science Foundation.

<sup>‡</sup>School of Mathematical Sciences, Tel-Aviv University, Israel. Email: [segevd@post.tau.ac.il](mailto:segevd@post.tau.ac.il).

## 1 Introduction

An instance of the *sorting buffer* problem consists of a metric space  $(V, d)$ , a server initially positioned at  $\rho_0 \in V$ , and a finite-capacity sorting buffer, capable of holding up to  $k$  requests. An additional ingredient of the input is an online sequence  $\sigma = \langle \sigma_1, \dots, \sigma_N \rangle$  of  $N$  requests, each of which corresponds to a point in  $V$ ; whenever a request arrives, it must be stored in the sorting buffer. At any point in time, a currently pending request  $\sigma_i$  can be served by drawing it out of the buffer and moving the server to  $\sigma_i$ . The objective is to serve all input requests in a way that minimizes the total distance traveled by the server.

The sorting buffer problem models a diverse collection of applications in networking, file server management, computer graphics, and even in the automotive industry. We refer the reader to directly related papers [5, 8, 9] and the references therein for a comprehensive review of these applications. However, to the best of our knowledge, essentially no non-trivial results are known for this problem in its utmost generality, i.e., when the given metric space has no particular structure. In fact, this statement holds even for the seemingly simple offline version, in which the input sequence  $\sigma$  is known in advance.

In light of this state of affairs, we focus our attention on instances of the problem in which the underlying metric is either an *evenly-spaced line metric* or a *continuous line metric*. More formally, in the former case  $V = \{1, \dots, n\}$ , whereas in the latter  $V = \mathbb{R}$ , noting that the distance function in both cases is  $d(p, q) = |p - q|$ . Although such restricted settings may appear to be very simple at first glance, we proceed by demonstrating that line metrics capture one of the most fundamental problems in the design of storage systems.

In most disk devices, the *seek time*, which is the time it takes the disk arm to move to the proper cylinder, dominates the time it takes to complete a read/write request. Consequently, reducing the mean seek time can dramatically improve the performance of the underlying storage system. Needless to say, when requests are served in the exact same order by which they arrive (i.e., FIFO order), the seek time is a predetermined constant. However, modern disks are capable of handling requests in an out-of-order fashion by maintaining a limited capacity buffer, in which requests can be temporarily stored. Hence, a scheduling policy that utilizes such a buffer to reorder requests may achieve a significant improvement over FIFO scheduling. Efficiently designing and implementing buffer-based scheduling policies has become one of the foremost objectives in the design of storage systems; it is referred to as the *disk arm scheduling* problem (see, for example, [10, 11]). This problem can be modeled as a sorting buffer instance on a line metric. Specifically, the disk's cylinders correspond to a set of points on the real line, the disk arm corresponds to the server, and the buffer used to reorder read/write requests corresponds to the sorting buffer.

The evenly-spaced line case has recently been studied by Khandekar and Pandit [7], who proposed a *randomized* online algorithm that obtains an expected competitive ratio of  $O(\log^2 n)$  against an oblivious adversary. Their approach is based on probabilistically embedding the given metric into a distribution over *binary hierarchically well-separated trees* [2, 3, 6]. It is worth noting that even though an embedding of this nature may seem somewhat artificial, the structural properties it guarantees considerably simplify the tasks of suggesting a buffer management policy and analyzing its performance.

### 1.1 Our results

**Evenly-spaced line metrics.** The main result of this paper is a *deterministic* online algorithm for the sorting buffer problem on an evenly-spaced line metric, which yields a competitive ratio of  $O(\log n)$ . This result improves on the randomized  $O(\log^2 n)$ -competitive algorithm due to Khandekar and Pandit [7]. It also refutes their conjecture, stating that a deterministic strategy is unlikely to obtain a non-trivial competitive ratio. The specifics of this algorithm are presented in Section 2.

**Continuous line metrics.** We study the sorting buffer problem on a continuous line metric, and employ the algorithm mentioned in the previous item as a subroutine to devise a deterministic online algorithm. Consequently, we achieve a competitive ratio of  $O(\log N \log \log N)$ , where  $N$  denotes the length of the input sequence. This result appears in Section 3.

**A deterministic lower bound.** We establish the first non-trivial lower bound for the sorting buffer problem on an evenly-spaced line metric. Specifically, we prove that the competitive ratio of any deterministic online algorithm is at least  $\frac{2+\sqrt{3}}{\sqrt{3}} \approx 2.154$ . This result settles, to some extent, an open question due to Khandekar and Pandit [7], who posed the task of attaining lower bounds on the achievable competitive ratio as a foundational objective for future research. Further details are provided in Section 4.

## 1.2 Related work

Räcke, Sohler and Westermann [9] seem to have been the first to study the sorting buffer problem in online settings, concentrating on the uniform case, in which all pairwise distances are equal. Their main result was a deterministic online algorithm that has a competitive ratio of  $O(\log^2 k)$ , where  $k$  denotes the buffer capacity. They also established a lower bound on the competitive ratio of several well-known heuristics. For example, they proved a lower bound of  $\Omega(k)$  on the performance of the Most-Common-First strategy and a lower bound of  $\Omega(\sqrt{k})$  on that of FIFO and Least-Recently-Used. Later on, Englert and Westermann [5] improved the main result of Räcke et al. [9], by suggesting a deterministic  $O(\log k)$ -competitive algorithm. In fact, their algorithm extends to a non-uniform case which is referred to as a star-like metric. They also investigated the possible gain in using a sorting buffer and showed that, for any metric space, a buffer of size  $k$  cannot reduce the total distance traveled by a factor of more than  $2k - 1$ . Very recently, Englert, Röglin and Westermann [4] suggested an alternative way of analyzing the algorithm of Englert and Westermann [5], and experimentally evaluated the performance of several strategies on random input sequences.

A concurrent line of work, initiated by Kohrt and Pruhs [8], studied the offline setting. They considered a maximization version of the sorting buffer problem, in which the objective is to maximize the cost reduction compared to a bufferless schedule, and proposed a polynomial-time 20-approximation on a uniform metric. Subsequently, Bar-Yehuda and Laserson [1] examined a generalized variant, for which they presented a polynomial-time algorithm that achieves an approximation ratio of 9.

## 2 Evenly-Spaced Line Metrics

In this section, we study the sorting buffer problem on an evenly-spaced line metric, and devise a deterministic online algorithm that achieves a competitive ratio of  $O(\log n)$ . Prior to describing the finer details of our approach, we introduce the notion of a doubling partition, which will considerably simplify the suggested algorithm and its analysis.

### 2.1 Doubling partitions

**Definition 2.1.** A *doubling partition* with respect to a point  $p \in V$ , denoted by  $\text{DP}(p)$ , is a partition of  $V \setminus \{p\}$  into  $2(\lfloor \log n \rfloor + 1)$  pairwise-disjoint sets of points  $L_0(p), \dots, L_{\lfloor \log n \rfloor}(p)$ ,  $R_0(p), \dots, R_{\lfloor \log n \rfloor}(p)$ , where

$$L_i(p) = \{q < p : 2^i \leq d(q, p) < 2^{i+1}\} \quad \text{and} \quad R_i(p) = \{q > p : 2^i \leq d(q, p) < 2^{i+1}\} .$$

Figure 1 provides a concrete example to a doubling partition in an evenly-spaced line metric.

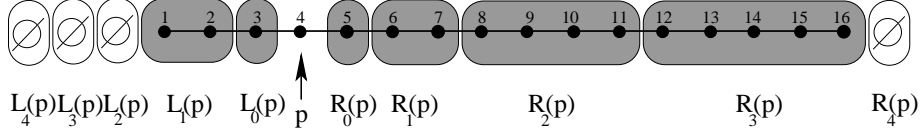


Figure 1: A doubling partition in an evenly-spaced 16-point line metric with respect to the point  $p = 4$ . Note that empty doubling partition sets are marked with  $\emptyset$ .

## 2.2 The algorithm

Noting that Englert and Westermann [5, Thm. 1] established an upper bound of  $O(k)$  on the competitive ratio of the FIFO strategy for any metric space, we may assume in the remainder of this section that  $k \geq 2(\lfloor \log n \rfloor + 1)$ . Furthermore, for ease of exposition, it would be convenient to denote  $m = 2(\lfloor \log n \rfloor + 1)$  and assume that  $k$  is an integral multiple of  $m$ .

Algorithm Moving Partition, formally described in Figure 2, works in phases, each of which is logically built from an *accumulation step*, in which newly read requests are stored, followed by a *clearance step*, in which the server travels to clear subsets of pending requests.

Figure 2: Algorithm Moving Partition

In each phase do ...

**Initialization:** Let  $p$  be the current position of the server. Associate a unique  $\frac{k}{m}$ -sized sub-buffer (of the  $k$ -sized sorting buffer) with each of the  $m$  points sets in  $DP(p)$ .

**The accumulation step:** Store each arriving request in the sub-buffer corresponding to the doubling partition set this request relates to<sup>1</sup>. If the current request relates to  $p$ , it is served immediately. This step ends when one of the sub-buffers becomes full or when the sequence of requests ends.

**The clearance step:**

- If one of the sub-buffers is full, let  $0 \leq t \leq \lfloor \log n \rfloor$  be the maximal index for which at least one of  $R_t(p)$  and  $L_t(p)$  has at least  $\frac{k}{2m}$  pending requests in its corresponding sub-buffer. We assume without loss of generality that the latter property is satisfied by  $R_t(p)$  (henceforth, the *maximal half-full set*), and designate its leftmost point by  $q$ . Move the server  $p \rightsquigarrow$  leftmost point of  $L_t(p) \rightsquigarrow$  rightmost point of  $R_t(p) \rightsquigarrow q$ , while clearing all pending requests that relate to  $\bigcup_{i=0}^t (L_i(p) \cup R_i(p))$  along the way. Then, the phase ends.
- If the sequence of requests ends, move the server  $p \rightsquigarrow$  leftmost point in buffer  $\rightsquigarrow$  rightmost point in buffer, while clearing all pending requests along the way. Then, the algorithm ends.

Figure 3 illustrates how the server travels during the clearance step when one of the sub-buffers becomes full. We remark that if  $L_t(p)$  is the maximal half-full set (instead of  $R_t(p)$ ), the server travels in a completely symmetrical way. That is, the server first travels to the rightmost point of  $R_t(p)$ , then to the leftmost point of  $L_t(p)$ , and finally to  $q$ , which is the rightmost point of  $L_t(p)$  in this case.

## 2.3 Analysis

To prove the correctness of Algorithm Moving Partition, it is sufficient to show that, at any point in time, the sorting buffer holds at most  $k$  pending requests. However, the following theorem demonstrates that the suggested algorithm satisfies an even stronger property.

<sup>1</sup>We say that a request *relates to a set of points*  $S \subseteq V$  when the destination of this request lies in  $S$ .

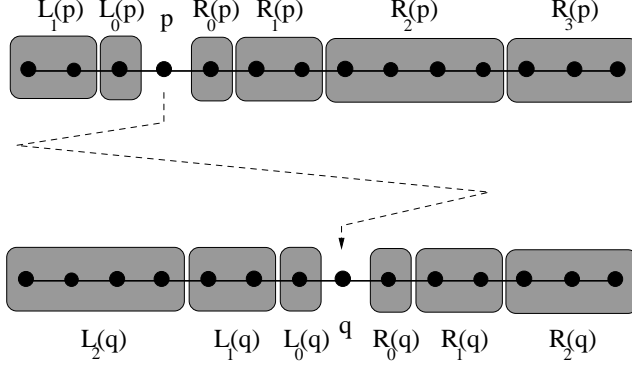


Figure 3: The server's movement during the clearance step when its initial position is  $p$  and the maximal half-full set is  $R_2(p)$ .

**Theorem 2.2.** *At any point in time, none of the sub-buffers (associated with the current doubling partition) overflows.*

**Proof.** We prove, by induction on the number of phases, that every sub-buffer contains strictly less than  $\frac{k}{m}$  pending requests at the beginning of any phase. The theorem follows by observing that sub-buffers cannot overflow during a phase (particularly, during the accumulation step).

The induction claim is trivially satisfied at the beginning of the first phase, since the sorting buffer is currently empty. Suppose the claim is satisfied at the beginning of phase  $\ell$  and suppose that in the clearance step of this phase, all pending requests that relate to  $\bigcup_{i=0}^t (L_i(p) \cup R_i(p))$  are cleared and that the final position of the server is  $q$ , which is without loss of generality the leftmost point of  $R_t(p)$ . We next show that every sub-buffer associated with a set in  $DP(q)$ , which is exactly the doubling partition at the beginning of phase  $\ell + 1$ , contains strictly less than  $\frac{k}{m}$  pending requests:

**$L_i(q)$  for  $0 \leq i \leq t$ .** It is easy to verify that  $\bigcup_{j=0}^t L_j(q) \subseteq \bigcup_{j=0}^t (L_j(p) \cup R_j(p)) \cup \{p\}$ . Thus, since all pending requests that relate to  $\bigcup_{j=0}^t (L_j(p) \cup R_j(p)) \cup \{p\}$  were served during the clearance step, it follows that the sub-buffer associated with  $L_i(q)$  is empty.

**$L_i(q)$  for  $t + 1 \leq i \leq \lfloor \log n \rfloor$ .** Consider a point  $v \in L_i(q)$ . Notice that  $d(v, q) \geq 2^i$ ,  $d(q, p) = 2^t$ , and  $d(v, p) + d(p, q) = d(v, q)$  since  $v < p < q$ . Hence, we have

$$d(v, p) = d(v, q) - d(p, q) \geq 2^i - 2^t \geq 2^i - 2^{i-1} = 2^{i-1}.$$

On the other hand,  $d(v, p) < d(v, q) < 2^{i+1}$ . Consequently,  $v \in L_{i-1}(p) \cup L_i(p)$ . This implies that any pending request that relates to  $L_i(q)$  was previously stored in one of the sub-buffers associated with  $L_{i-1}(p)$  and  $L_i(p)$ . Recall that every sub-buffer associated with a set of  $DP(p)$  holds at most  $\frac{k}{2m} - 1$  pending requests at the end of a clearance step. Thus, the sub-buffer associated with  $L_i(q)$  holds at most  $\frac{k}{m} - 2$  pending requests.

**$R_i(q)$  for  $0 \leq i \leq \lfloor \log n \rfloor$ .** Now consider a point  $v \in R_i(q)$ . Clearly,  $v \in R_j(p)$  for some  $j \geq i$ . Accordingly, all the metric points of  $R_i(q)$  appear in at most two consecutive sets of  $DP(p)$ . Arguments similar to those of the previous item imply that the sub-buffer associated with  $R_i(q)$  holds at most  $\frac{k}{m} - 2$  pending requests. ■

In what follows, we prove that the algorithm under consideration achieves a competitive ratio of  $O(\log n)$ . For ease of presentation, it would be convenient to view the evenly-spaced line metric as an undirected graph  $G = (V, E)$  with  $V = \{1, \dots, n\}$  and  $E = \{(1, 2), \dots, (n-1, n)\}$ . In addition, we introduce the following notation:

- Let  $\text{OPT}$  denote the total distance traveled by the server in an optimal solution, and let  $\text{ON}$  denote the total distance traveled by the server in Algorithm Moving Partition.
- Let  $P$  be the sequence of points  $p_0, p_1, \dots, p_\ell$ , where  $p_i$  is the position of the server at the end of the  $i$ -th phase of Algorithm Moving Partition. Note that  $p_0$  is the initial server position  $\rho_0$  and that  $p_\ell$  is the position of the server just before the final clearance step begins.
- Let  $C_P(e)$  be the number of times the edge  $e \in E$  is crossed with respect to the walk determined by the points of  $P$  (i.e., the walk  $p_0 \rightsquigarrow p_1 \rightsquigarrow \dots \rightsquigarrow p_\ell$ ), and let  $C_{\text{OPT}}(e)$  be the number of times this edge is crossed in the optimal solution. Notice that  $\sum_{e \in E} C_P(e) = \sum_{i=1}^{\ell} d(p_{i-1}, p_i)$  whereas  $\sum_{e \in E} C_{\text{OPT}}(e) = \text{OPT}$ .

**Lemma 2.3.**  $\text{ON} \leq 7 \sum_{e \in E} C_P(e) + 2 \cdot \text{OPT}$ .

**Proof.** Consider a phase  $1 \leq i \leq \ell$ , in which a full sub-buffer was detected, and let  $t$  be the index of the maximal half-full set of that phase. Then, the total distance traveled by the server in this phase, denoted by  $\text{ON}_i$ , is at most  $3(2^{t+1} - 1) + (2^t - 1) \leq 7 \cdot 2^t$ , while  $d(p_{i-1}, p_i) = 2^t$ . Thus,

$$\sum_{i=1}^{\ell} \text{ON}_i \leq 7 \sum_{i=1}^{\ell} d(p_{i-1}, p_i) = 7 \sum_{e \in E} C_P(e).$$

During the clearance step of the final phase (i.e., phase  $\ell + 1$ ), the server clears all pending requests in the buffer. Obviously, the total distance  $\text{ON}_{\ell+1}$  traveled by the server in this phase satisfies  $\text{ON}_{\ell+1} \leq 2 \cdot \text{OPT}$ , as the server crosses each edge between the leftmost and the rightmost requests that have ever arrived (including the initial server position) at most twice, and any feasible solution must cross each such edge at least once. Hence,

$$\text{ON} = \sum_{i=1}^{\ell+1} \text{ON}_i \leq 7 \sum_{e \in E} C_P(e) + 2 \cdot \text{OPT}.$$

■

**Lemma 2.4.**  $C_P(e) \leq (12m + 4)C_{\text{OPT}}(e)$  for every  $e \in E$ .

**Proof.** Let  $e = (i, i + 1)$ . We first prove the following two claims.

**Claim I:  $C_{\text{OPT}}(e) \geq 1$  whenever  $C_P(e) \geq 1$ .** Assume without loss of generality that the first time  $e$  is crossed in the walk determined by  $P$  is from left to right. Accordingly, the initial position of the server must reside left of  $e$  (i.e.,  $p_0 \in \{1, \dots, i\}$ ), and there must be at least one request that resides right of  $e$  (i.e., in  $\{i + 1, \dots, n\}$ ). Thus, in any feasible solution, the server must cross the edge  $e$ .

**Claim II:  $C_{\text{OPT}}(e) \geq \lfloor \frac{C_P(e)}{6m+2} \rfloor$ .** In every clearance step of Algorithm Moving Partition, each of the  $m$  sub-buffers that has at least  $\frac{k}{2m}$  pending requests is cleared. Thus, at the beginning of each phase, the overall number of pending requests is at most  $\frac{k}{2}$ . Also notice that each time  $P$  crosses  $e$  from left to right, the server clears at least  $\frac{k}{2m}$  requests that reside right of  $e$ . Consequently, in  $3m + 1$  times  $P$  crosses  $e$  from left to right, at least  $(3m + 1)\frac{k}{2m} - \frac{k}{2} > k$  requests must have arrived to the right of  $e$ . Since similar arguments are applicable to the opposite case (i.e., when  $e$  is crossed from right to left), and since  $e$  is alternately crossed by  $P$  from different directions, it follows that during  $6m + 2$  times  $P$  crosses  $e$ , any algorithm must cross it at least once.

The lemma clearly holds when  $C_P(e) = 0$ . When  $C_P(e) \geq 1$ , the claims stated above imply that  $\frac{C_P(e)}{6m+2} \leq \lfloor \frac{C_P(e)}{6m+2} \rfloor + 1 \leq 2C_{\text{OPT}}(e)$ , or equivalently  $C_P(e) \leq (12m + 4)C_{\text{OPT}}(e)$ . ■

**Theorem 2.5.** *Algorithm Moving Partition is  $O(\log n)$ -competitive.*

**Proof.** Using the previously stated results, we have

$$\text{ON} \leq 7 \sum_{e \in E} C_P(e) + 2 \cdot \text{OPT} \leq 7(12m + 4) \sum_{e \in E} C_{\text{OPT}}(e) + 2 \cdot \text{OPT} = (84m + 30)\text{OPT},$$

where the first inequality follows from Lemma 2.3, and the second is due to Lemma 2.4. Since  $m = 2(\lfloor \log n \rfloor + 1)$ , it follows that  $\text{ON} = O(\log n)\text{OPT}$ . ■

We conclude this section by arguing that our analysis is tight. Namely, we demonstrate that there are input sequences for which the total distance traveled by the server in Algorithm Moving Partition is  $\Omega(\log n)$  times the minimum possible.

**Theorem 2.6.** *Algorithm Moving Partition is  $\Omega(\log n)$ -competitive.*

**Proof.** Suppose we are given an evenly-spaced line metric on  $\{1, \dots, n\}$ . Let  $\rho_0 = 1$  be the initial position of the server,  $k$  be the size of the sorting buffer, and  $\langle (n^{k/2m} 1^{k/2m})^m \rangle$  be the input sequence given by the adversary. In Algorithm Moving Partition, the server makes  $m$  round-trips of the form  $1 \rightsquigarrow n \rightsquigarrow 1$ . An optimal solution, on the other hand, accumulates all input requests, and then serves them while making a single trip to  $n$ . Hence, since  $m = 2(\lfloor \log n \rfloor + 1)$ , it follows that Algorithm Moving Partition is  $\Omega(\log n)$ -competitive. ■

### 3 Continuous Line Metrics

In this section, we present a deterministic online algorithm that attains a competitive ratio of  $O(\log N \log \log N)$  for the sorting buffer problem on a continuous line metric. We begin by considering an inherently simpler setting, in which certain properties of the input sequence are known in advance. Later on, we show that the dependency on these properties can be eliminated by utilizing online “guessing” techniques.

#### 3.1 A semi-online algorithm

In the following, we deal with a restricted special case of the problem under consideration, in which we have a prior knowledge of two input-related characteristics:  $\tilde{N}$ , an upper bound on the number of requests (i.e.,  $N \leq \tilde{N}$ ), and  $\tilde{D}$ , an upper bound on the maximal distance between the initial server position and any input request (i.e.,  $\sigma_i \in [\rho_0 - \tilde{D}, \rho_0 + \tilde{D}]$ , for every  $1 \leq i \leq N$ ). In Figure 4, we propose a deterministic online algorithm for these particular settings.

Figure 4: Algorithm Discretized Simulation

**Discretization:** Let  $V$  be the set of  $\tilde{N} + 1$  points equally dividing  $[\rho_0 - \tilde{D}, \rho_0 + \tilde{D}]$  into  $\tilde{N}$  disjoint intervals, each of length  $\frac{2\tilde{D}}{\tilde{N}}$ . In addition, let  $\tilde{\sigma} = \langle \tilde{\sigma}_1, \dots, \tilde{\sigma}_N \rangle$  be the discretized input sequence, in which  $\tilde{\sigma}_i$  is the point in  $V$  nearest to  $\sigma_i$ , where ties are broken arbitrarily.

**Simulation:** Apply Algorithm Moving Partition to the input sequence  $\tilde{\sigma}$ . Move the server to clear requests in the exact same order by which they are cleared in Moving Partition.

**Analysis.** Since all algorithms described in this section are assumed to have identical sorting buffer sizes  $k$  and initial server positions  $\rho_0$ , as far as notation is concerned – we may focus on their input sequences. Consequently, we use  $\text{OPT}_\sigma$  to denote the total distance traveled by the server in an optimal algorithm when the input sequence is  $\sigma$ ,  $\text{DS}_\sigma$  to denote the total distance traveled in Algorithm Discretized Simulation when the input sequence is  $\sigma$ , and  $\text{MP}_{\tilde{\sigma}}$  to denote the total distance traveled in Algorithm Moving Partition when the input sequence is  $\tilde{\sigma}$ .

**Lemma 3.1.**  $\text{OPT}_{\tilde{\sigma}} \leq \text{OPT}_{\sigma} + 2\tilde{D}$  and  $\text{DS}_{\sigma} \leq \text{MP}_{\tilde{\sigma}} + 2\tilde{D}$ .

**Proof.** In what follows, we prove the first inequality, noting that the second inequality can be easily established by using nearly identical arguments. To prove  $\text{OPT}_{\tilde{\sigma}} \leq \text{OPT}_{\sigma} + 2\tilde{D}$ , it is sufficient to show that the entire sequence  $\tilde{\sigma}$  can be processed within a traveling distance of at most  $\text{OPT}_{\sigma} + 2\tilde{D}$ . For this purpose, let  $\text{ALG}(\tilde{\sigma})$  denote the algorithm that serves requests from  $\tilde{\sigma}$  in exactly the same order by which the corresponding requests of  $\sigma$  are served in an optimal algorithm whose input sequence is  $\sigma$ . In other words, if the  $j$ -th request  $\rho_j$  served by the latter algorithm is  $\sigma_i$ , then the  $j$ -th request  $\tilde{\rho}_j$  served by  $\text{ALG}(\tilde{\sigma})$  is  $\tilde{\sigma}_i$ . Notice that

$$d(\tilde{\rho}_j, \tilde{\rho}_{j+1}) \leq d(\tilde{\rho}_j, \rho_j) + d(\rho_j, \rho_{j+1}) + d(\rho_{j+1}, \tilde{\rho}_{j+1}) \leq d(\rho_j, \rho_{j+1}) + \frac{2\tilde{D}}{\tilde{N}},$$

where the second inequality holds since  $d(\rho_j, \tilde{\rho}_j) \leq \frac{\tilde{D}}{\tilde{N}}$ , for every  $0 \leq j \leq N$ . It follows that the total distance traveled by the server in  $\text{ALG}(\tilde{\sigma})$  is

$$\sum_{j=0}^{N-1} d(\tilde{\rho}_j, \tilde{\rho}_{j+1}) \leq \sum_{j=0}^{N-1} d(\rho_j, \rho_{j+1}) + N \cdot \frac{2\tilde{D}}{\tilde{N}} \leq \text{OPT}_{\sigma} + 2\tilde{D}.$$

■

**Theorem 3.2.**  $\text{DS}_{\sigma} = O(\log \tilde{N}) \cdot (\text{OPT}_{\sigma} + \tilde{D})$ .

**Proof.** Using the previously stated results, we have

$$\text{DS}_{\sigma} \leq \text{MP}_{\tilde{\sigma}} + 2\tilde{D} \leq c \log \tilde{N} \cdot \text{OPT}_{\tilde{\sigma}} + 2\tilde{D} \leq c \log \tilde{N} \cdot (\text{OPT}_{\sigma} + 2\tilde{D}) + 2\tilde{D}.$$

The first inequality follows from the second claim in Lemma 3.1. The second inequality follows from Theorem 2.5, stating that there exists a constant  $c > 0$  such that  $\text{MP}_{\tilde{\sigma}} \leq c \log \tilde{N} \cdot \text{OPT}_{\tilde{\sigma}}$ . Finally, the last inequality is due to the first claim in Lemma 3.1. ■

### 3.2 A fully-online algorithm

In what follows, we show that the dependency on knowing  $\tilde{N}$  and  $\tilde{D}$  in advance, exhibited by Algorithm Discretized Simulation, can be eliminated by guessing these parameter in online fashion. The specifics of our approach are formally presented in Figure 5.

Figure 5: Algorithm Doubling Simulation

Let  $\tilde{N} = 4$  and  $\tilde{D} = 0$ . In each phase do ...

**The simulation step:** Simulate the execution of Algorithm Discretized Simulation, assuming that  $\tilde{N}$  and  $\tilde{D}$  are upper bounds on the number of requests and the maximal distance between  $\rho_0$  and any input request, respectively. Move the server to clear requests in the exact same order by which they are cleared in Discretized Simulation. This step ends when the next input point  $\tilde{\sigma}$  is the  $(\tilde{N} + 1)$ -th request arrived so far or when  $\tilde{\sigma} \notin [\rho_0 - \tilde{D}, \rho_0 + \tilde{D}]$ .

**The doubling step:**

- Let  $p$  be the current position of the server. Move the server  $p \rightsquigarrow \rho_0 - \tilde{D} \rightsquigarrow \rho_0 + \tilde{D} \rightsquigarrow \rho_0$ , while clearing all pending requests in the sorting buffer along the way.
- If  $\tilde{\sigma}$  is the  $(\tilde{N} + 1)$ -th request arrived so far, set  $\tilde{N} = \tilde{N}^2$ . If  $\tilde{\sigma} \notin [\rho_0 - \tilde{D}, \rho_0 + \tilde{D}]$ , set  $\tilde{D} = 2d(\rho_0, \tilde{\sigma})$ . Then, the phase ends.

**Analysis.** For the purpose of analyzing the performance of Algorithm Doubling Simulation, we introduce the following notation:

- Let  $T$  be the number of phases in the algorithm, and let  $\tilde{N}_t$  and  $\tilde{D}_t$  denote the values of  $\tilde{N}$  and  $\tilde{D}$  at the beginning of phase  $t$ , respectively.
- Let  $\text{OPT}$  denote the total distance traveled by the server in an optimal algorithm, and let  $\text{ON}$  denote the total distance traveled in Algorithm Doubling Simulation.
- Let  $\text{ON}_t^{\text{sim}}$  and  $\text{ON}_t^{\text{dbl}}$  denote the total distance traveled by the server in the simulation and doubling steps of phase  $t$ , respectively. Notice that  $\text{ON} = \sum_{t=1}^T (\text{ON}_t^{\text{sim}} + \text{ON}_t^{\text{dbl}})$ .
- Let  $\varrho_t$  be the sub-sequence of  $\sigma$  that consists of the requests processed in phase  $t$  of the algorithm. Notice that  $\sigma = \langle \varrho_1, \dots, \varrho_T \rangle$ . Additionally, let  $\text{OPT}_t$  denote the total distance traveled by the server in an optimal solution where the initial position of the server is  $\rho_0$  and the input sequence is  $\varrho_t$ .

**Lemma 3.3.** *There are at most  $\lceil \log \log N \rceil$  phases in which  $\tilde{N}$  is incremented.*

**Proof.** Recall that  $\tilde{N}_1 = 4 = 2^2$  and that each time  $\tilde{N}$  is incremented, its value is squared. Thus, after the  $i$ -th time  $\tilde{N}$  is incremented, its value is  $2^{2^{i+1}}$ . Hence, we have  $\tilde{N} \geq N$  within no more than  $\lceil \log \log N \rceil$  increment steps. ■

**Lemma 3.4.**  $\log \tilde{N}_T < 2 \log N$ .

**Proof.** Assume without loss of generality that there is at least one phase in which  $\tilde{N}$  is incremented; otherwise, there are at most 4 requests and Algorithm Doubling Simulation achieves constant factor competitiveness. Let  $2 \leq i \leq T$  be the minimal phase number such that after this phase  $\tilde{N}$  is no longer incremented, i.e.,  $\tilde{N}_{i-1} < N \leq \tilde{N}_i = \tilde{N}_{i+1} = \dots = \tilde{N}_T$ . Recall that  $\tilde{N}_i = \tilde{N}_{i-1}^2$  and thus,  $\log \tilde{N}_T = \log \tilde{N}_i = 2 \log \tilde{N}_{i-1} < 2 \log N$ . ■

**Lemma 3.5.**  $\text{ON}_t^{\text{dbl}} \leq 5\tilde{D}_t$  for every  $1 \leq t \leq T$ .

**Proof.** Recall that, during the doubling step of phase  $t$ , the server travels  $p \rightsquigarrow \rho_0 - \tilde{D}_t \rightsquigarrow \rho_0 + \tilde{D}_t \rightsquigarrow \rho_0$ , where  $p$  denotes the server position at the beginning of this step. Therefore,

$$\text{ON}_t^{\text{dbl}} = d(p, \rho_0 - \tilde{D}_t) + d(\rho_0 - \tilde{D}_t, \rho_0 + \tilde{D}_t) + d(\rho_0 + \tilde{D}_t, \rho_0) \leq 5\tilde{D}_t,$$

where the inequality holds since the server never leaves  $[\rho_0 - \tilde{D}_t, \rho_0 + \tilde{D}_t]$  prior to phase  $t + 1$ , and in particular  $p \in [\rho_0 - \tilde{D}_t, \rho_0 + \tilde{D}_t]$ . ■

**Lemma 3.6.**  $\sum_{t=1}^T \tilde{D}_t \leq 2(\log \log N + 2)\text{OPT}$ .

**Proof.** Let  $\ell$  be the number of phases in which the value of  $\tilde{D}$  is incremented. Clearly,  $\sum_{t=1}^T \tilde{D}_t$  is maximized when the increments in  $\tilde{D}$  occur in the first  $\ell$  phases. This implies that

$$\sum_{t=1}^T \tilde{D}_t \leq \sum_{t=1}^{\ell} \frac{\tilde{D}_T}{2^t} + (T - \ell)\tilde{D}_T \leq \tilde{D}_T + \lceil \log \log N \rceil \tilde{D}_T \leq (\log \log N + 2)\tilde{D}_T,$$

where the first inequality holds since each time  $\tilde{D}$  is incremented its value is at least doubled, and the second inequality holds since the number of phases in which  $\tilde{D}$  remains unchanged is at most  $\lceil \log \log N \rceil$ , which follows from Lemma 3.3. Clearly, there is at least one request  $\tilde{\sigma}$  for which  $\tilde{D}_T = 2d(\rho_0, \tilde{\sigma})$ . In addition,  $d(\rho_0, \tilde{\sigma}) \leq \text{OPT}$ , as the server must travel to  $\tilde{\sigma}$  in any feasible solution. Thus,  $\sum_{t=1}^T \tilde{D}_t \leq 2(\log \log N + 2)\text{OPT}$ . ■

**Lemma 3.7.**  $\sum_{t=1}^T \text{OPT}_t \leq \text{OPT} + 6 \sum_{t=1}^T \tilde{D}_t$ .

**Proof.** Consider the execution of an optimal algorithm, and break it up into  $T$  sub-executions such that for every  $1 \leq t \leq T - 1$ , the  $t$ -th sub-execution begins when the first request of  $\varrho_t$  arrives and ends just before the first request of  $\varrho_{t+1}$  arrives. Specifically, it ends after the algorithm serves a request, whose destination is  $q_t$ , that precedes the arrival of the first request of  $\varrho_{t+1}$ . In addition, sub-execution  $T$  begins when the first request of  $\varrho_T$  arrives and ends when the algorithm ends. Now suppose we modify each of these sub-executions in the following way:

- For every  $1 \leq t \leq T - 1$ , when sub-execution  $t$  ends, we move the server  $q_t \rightsquigarrow \rho_0 - \tilde{D}_t \rightsquigarrow \rho_0 + \tilde{D}_t \rightsquigarrow \rho_0$ , and clear all pending requests in the sorting buffer along the way.
- For every  $2 \leq t \leq T$ , when sub-execution  $t$  begins, we move the server  $\rho_0 \rightsquigarrow q_{t-1}$ .

It is easy to verify that the movement of the server in this modified execution is valid and that the total distance traveled is at most  $\text{OPT} + 6 \sum_{t=1}^T \tilde{D}_t$ , which follows from arguments similar to those in Lemma 3.5 and the fact that without loss of generality  $q_t \in [\rho_0 - \tilde{D}_t, \rho_0 + \tilde{D}_t]$ , for every  $1 \leq t \leq T - 1$ . Notice that in this modified execution, it is quite possible that some requests are served more than once. However, one can modify the remaining execution by removing server movements aimed to clear requests that have already been cleared, and obtain a feasible execution, in which the total distance traveled can only decrease. Hence, we may assume that this modified execution is feasible.

We now argue that, for every  $1 \leq t \leq T$ , the total distance traveled in modified sub-execution  $t$  provides an upper bound on  $\text{OPT}_t$ . This follows from the observation that at the beginning of modified sub-execution  $t$ , the server is positioned at  $\rho_0$ , and that at the end of this sub-execution, all requests of  $\varrho_t$  were served. Hence,  $\sum_{t=1}^T \text{OPT}_t \leq \text{OPT} + 6 \sum_{t=1}^T \tilde{D}_t$ .  $\blacksquare$

**Theorem 3.8.** *Algorithm Doubling Simulation is  $O(\log N \log \log N)$ -competitive.*

**Proof.** Using the previously stated results, we have

$$\begin{aligned}
\text{ON} &= \sum_{t=1}^T (\text{ON}_t^{\text{sim}} + \text{ON}_t^{\text{dbl}}) \\
&\leq c \sum_{t=1}^T \left( \log \tilde{N}_t \cdot (\text{OPT}_t + \tilde{D}_t) \right) + 5 \sum_{t=1}^T \tilde{D}_t \\
&< 2c \log N \sum_{t=1}^T \text{OPT}_t + (2c \log N + 5) \sum_{t=1}^T \tilde{D}_t \\
&\leq 2c \log N \cdot \text{OPT} + (14c \log N + 5) \sum_{t=1}^T \tilde{D}_t \\
&\leq \left( (2c \log N) + 2(\log \log N + 2)(14c \log N + 5) \right) \text{OPT} \\
&= O(\log N \log \log N) \text{OPT} .
\end{aligned}$$

The first inequality is obtained by combining Lemma 3.5 and Theorem 3.2, stating that there exists a constant  $c > 0$  such that  $\text{ON}_t^{\text{sim}} \leq c \log \tilde{N}_t \cdot (\text{OPT}_t + \tilde{D}_t)$  for every  $1 \leq t \leq T$ . The second inequality holds since  $\log \tilde{N}_t \leq \log \tilde{N}_T < 2 \log N$  for every  $1 \leq t \leq T$ , which follows from Lemma 3.4. The third inequality follows from Lemma 3.7. Finally, the last inequality is due to Lemma 3.6.  $\blacksquare$

## 4 A Lower Bound for any Deterministic Algorithm

In this section, we establish a lower bound of 2.154 on the competitive ratio of any deterministic online algorithm for the sorting buffer problem. We begin by introducing the notion of laziness, which reduces the objective of proving a lower bound for any deterministic online algorithm to that of proving a lower bound for the family of deterministic lazy algorithms.

**Definition 4.1.** A *lazy* algorithm for the sorting buffer problem is an algorithm that satisfies the following properties:

1. The server stores newly read requests as long as the buffer is not full.
2. When the buffer holds a request that relates to the current server position, the server clears it immediately.

**Theorem 4.2.** *The competitive ratio of any deterministic online algorithm is at least  $\frac{2+\sqrt{3}}{\sqrt{3}} \approx 2.154$ , even for evenly-spaced line metrics.*

**Proof.** The forthcoming arguments will be based on the fact that every sorting buffer algorithm can be made lazy without increasing the total distance traveled by the server. Having this observation in mind, suppose we are given the line metric schematically described in Figure 6. More formally, the left-to-right order of the points is  $p, p_0, p_1, \dots, p_{k-1}$ , with  $d(p, p_0) = \alpha(k-1)$  and  $d(p_i, p_{i+1}) = 1$  for every  $0 \leq i \leq k-2$ , where  $\alpha > 0$  is a parameter whose value will be determined later. We assume, for simplicity, that  $\alpha(k-1)$  is integral. Hence, by adding dummy points the specified metric can be viewed as an evenly-spaced line metric on  $\alpha(k-1) + k$  points.

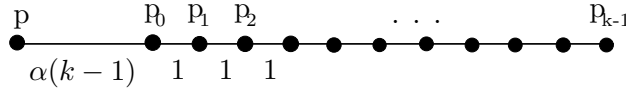


Figure 6: The lower bound instance.

Let  $p_0$  be the initial position of the server,  $k$  be the size of the sorting buffer, and  $\sigma_{k-1} = \langle p^{k-1} p_1 p_2 \dots p_{k-1} p p_{k-1} \rangle$  be the input sequence given by the adversary. We identify any lazy deterministic algorithm with the maximal index  $i$  for which, given the sequence  $\sigma_{k-1}$ , the server initially travels from  $p_0$  to  $p_1, \dots, p_i$ , and then travels to  $p$ . For all algorithms identified with  $i < k-1$ , the adversary changes the input sequence to  $\sigma_i = \langle p^{k-1} p_1 p_2 \dots p_{i+1}^k p \rangle$ , that is, the postfix  $\langle p_{i+2} \dots p_{k-1} p p_{k-1} \rangle$  of  $\sigma_{k-1}$  is replaced by  $\langle p_{i+1}^{k-1} p \rangle$ . We now consider two cases, depending on which sequence was picked by the adversary:

**Case I: The input sequence was  $\sigma_{k-1}$ .** The total distance traveled by the server is at least  $(3+2\alpha)(k-1)$ , since it moves  $p_0 \rightsquigarrow p_{k-1} \rightsquigarrow p \rightsquigarrow p_{k-1}$ . However, the optimal distance is at most  $(1+2\alpha)(k-1)$ , as all requests can be cleared by traveling  $p_0 \rightsquigarrow p \rightsquigarrow p_{k-1}$ . Thus, the competitive ratio of any lazy deterministic online algorithm identified with  $k-1$  is at least  $\frac{3+2\alpha}{1+2\alpha}$ .

**Case II: The input sequence was  $\sigma_i$ , for  $i < k-1$ .** The total distance traveled by the server is at least  $3\alpha(k-1) + 4i + 2$ , since it moves  $p_0 \rightsquigarrow p_i \rightsquigarrow p \rightsquigarrow p_{i+1} \rightsquigarrow p$ , whereas an optimal solution travels  $p_0 \rightsquigarrow p_{i+1} \rightsquigarrow p$ , to obtain a total distance of  $\alpha(k-1) + 2i + 2$ . Thus, the competitive ratio of any lazy deterministic online algorithm identified with  $i$  is at least

$$\frac{3\alpha(k-1) + 4i + 2}{\alpha(k-1) + 2i + 2} \geq \frac{3\alpha(k-1) + 4k - 6}{\alpha(k-1) + 2k - 2} = \frac{3\alpha + 4}{\alpha + 2} - \frac{2}{(\alpha + 2)(k-1)},$$

where the inequality holds since the left-hand side is minimized when  $i = k-2$ . It follows that for any  $\epsilon > 0$  we can pick a sufficiently large value of  $k$  so that

$$\frac{3\alpha + 4}{\alpha + 2} - \frac{2}{(\alpha + 2)(k-1)} \geq \frac{3\alpha + 4}{\alpha + 2} - \epsilon.$$

Therefore, the competitive ratio of any lazy deterministic online algorithm on an evenly-spaced line metric is at least

$$\max_{\alpha > 0} \min \left\{ \frac{3 + 2\alpha}{1 + 2\alpha}, \frac{3\alpha + 4}{\alpha + 2} - \epsilon \right\} .$$

By optimizing the value of  $\alpha$  (i.e., setting  $\alpha^* = \frac{\sqrt{3}-1}{2}$ ), we obtain a lower bound of  $\frac{2+\sqrt{3}}{\sqrt{3}} - \epsilon$  for any  $\epsilon > 0$ . However, recall that we assumed  $\alpha(k-1)$  to be integral. Since  $\alpha^*$  is irrational and  $k$  is an integer, it follows that this assumption does not hold. Nevertheless, this difficulty can be resolved by standard approximation of an irrational number by a rational number, losing an extra additive factor of  $\epsilon$  in the lower bound. ■

## References

- [1] R. Bar-Yehuda and J. Laserson. Exploiting locality: Approximating sorting buffers. In *Proceedings of the 3rd International Workshop on Approximation and Online Algorithms*, pages 69–81, 2005.
- [2] Y. Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, pages 184–193, 1996.
- [3] Y. Bartal. On approximating arbitrary metrics by tree metrics. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 161–168, 1998.
- [4] M. Englert, H. Röglin, and M. Westermann. Evaluation of online strategies for reordering buffers. In *Proceedings of the 5th International Workshop on Experimental Algorithms*, pages 183–194, 2006.
- [5] M. Englert and M. Westermann. Reordering buffer management for non-uniform cost models. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming*, pages 627–638, 2005.
- [6] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 448–455, 2003.
- [7] R. Khandekar and V. Pandit. Online sorting buffers on line. In *Proceedings of the 23rd Annual Symposium on Theoretical Aspects of Computer Science*, pages 584–595, 2006.
- [8] J. S. Kohrt and K. Pruhs. A constant approximation algorithm for sorting buffers. In *Proceedings of the 6th Latin American Symposium on Theoretical Informatics*, pages 193–202, 2004.
- [9] H. Räcke, C. Sohler, and M. Westermann. Online scheduling for sorting buffers. In *Proceedings of the 10th Annual European Symposium on Algorithms*, pages 820–832, 2002.
- [10] A. Silberschatz, P. B. Galvin, and G. Gagne. *Applied operating system concepts*. John Wiley and Sons, Inc., first edition, 2000. Disc scheduling is discussed in Section 13.2.
- [11] A. S. Tanenbaum. *Modern Operating Systems*. Prentice Hall PTR, second edition, 2001. Disc scheduling is discussed in Section 5.4.