Online Sorting Buffers on Line

Rohit Khandekar* Vinayaka Pandit†

September 16, 2005

Abstract

We consider the online scheduling problem for sorting buffers on a line metric. This problem is motivated by an application to disc scheduling. The input to this problem is a sequence of requests. Each request is a block of data to be written on a specified track of the disc. The disc is modeled as a number of tracks arranged on a line. To write a block on a particular track, the scheduler has to bring the disc head to that track. The cost of moving the disc head from a track to another is the distance between those tracks. A sorting buffer that can store at most k requests at a time is available to the scheduler. This buffer can be used to rearrange the input sequence. The objective is to minimize the total cost of head movement while serving the requests. On a disc with n uniformly-spaced tracks, we give a randomized online algorithm with a competitive ratio of $O(\log^3 n)$ in expectation against an oblivious adversary. This algorithm also yields a competitive ratio of $O(\alpha^{-1} \log^3 n)$ if we are allowed to use a buffer of size αk for any $1 \leq \alpha \leq \log n$. This is the first non-trivial approximation for the sorting buffers problem on a line metric. Our technique is based on probabilistically embedding the line metric into hierarchically well-separated trees. We show that any deterministic strategy which makes scheduling decisions based only on the contents of the buffer has a competitive ratio of $\Omega(k)$.

Category: Algorithms and Data Structures.

1 Introduction

Disc scheduling is a fundamental problem in the design of storage systems. Standard text books on operating systems contain detailed discussion on disc scheduling. They discuss various heuristics for scheduling the movement of the disc head. It is to be noted that the difference in the performance of different scheduling strategies can be seen only when a buffer which can hold more than one request in order to rearrange the sequence is available (See section 13.2 in [9]). In this paper, we consider the problem of efficiently scheduling the head movement of a disc when a buffer of limited size is available to rearrange an online sequence of requests. Overlooking certain operational details, the disc is modeled as a number of tracks arranged on a straight line. The input is a sequence of requests. Each request is a block of data and specifies the track on which the data needs to be written. Available to the scheduler is a buffer which can store at most k requests at a time. The buffer can be used to rearrange the input sequence. To write a block of data onto a track, the disc head has to be moved to that track. The cost of moving the head from track i to track j is assumed to be $|i - j|$. The goal of the scheduler is to serve all the requests while

*University of California, Berkeley. email: rkhandekar@gmail.com

†IBM India Research Laboratory, New Delhi. email: pvinayak@in.ibm.com

minimizing the overall cost of head movement. We call this problem the Sorting Buffers problem (SBP) on a line metric to be consistent with the previous work. Note that, in the absence of the sequencing restriction, the optimal schedule can be found offline by simply sorting the requests.

The Disc Scheduling problem is well studied in the design of storage systems. Several popular heuristics like Shortest Seek First (SSF), Shortest Time First(STF), and CSCAN have been proposed. In standard storage terminology, *seek time* is defined as the time required for the head to move to the track in which data is to be written. The *rotational latency* is the time required for the requested sector to pass under the head. We observe that our formulation of the problem is based on seek time only. SSF strategy, at each step, schedules the request with the shortest seek time among all the pending requests. STF strategy, at each step, schedules the request with the minimum of seek time plus rotational latency among all the pending requests. In the CSCAN schedule, the head starts from one end of the disc and travels to the other end, servicing all the requests on a track while passing it. After reaching the other end, it moves back to its starting point and repeats. Note that the CSCAN may violate buffer constraint. However, this problem has not been studied so far from the point of view of approximation guarantees. Andrews et al. [1] studied a related disc scheduling problem in the offline setting. They consider a model in which a convex reachability function determines how long it takes for the head to move between two tracks. Given a set of requests, they consider the problem of minimizing the time required to serve all the requests. Note that, unlike the SBP, their problem does not impose any sequencing restriction. They gave a $\frac{3}{2}$ approximation for the disc scheduling problem with convex reachability function in which there is no buffer constraint. They show that the problem can be solved optimally in polynomial time if the reachability function is linear. They leave the online problem with buffer constraint as an open problem.

1.1 Previous work

The SBP can in fact be defined on any metric space. The input is then a sequence of requests each of which corresponds to a point in the metric space. To serve a request after its arrival, the server has to visit the corresponding point. The cost of moving the server from a point to another is the distance between those points. The sorting buffer which can store at most k requests can be used to rearrange the sequence and the goal is to minimize the total movement of the server to serve all the requests. Let N denote the total number of requests. It is easy to see that if $k = N$ and there is one request to each point, then this problem is essentially the Hamiltonian path problem on the given metric. Thus the offline version of SBP on general metrics is NP-hard. On a line metric, however, it is not known if the offline version is NP-hard for a general k. To the best of our knowledge, no non-trivial lower or upper bounds on the approximation (resp. competitive) ratio of either deterministic or randomized offline (resp. online) algorithms are known for a line metric.

The SBP on a uniform metric (in which all pairwise distances are 1) has been studied before. Racke et al. [8] presented a deterministic online algorithm, called *Bounded Waste* that has $O(\log^2 k)$ competitive ratio. They also showed that some natural strategies like First-In-first-Out, Least-Recently-Used, etc. have $\Omega(\sqrt{k})$ competitive ratio. Kohrt and Pruhs [7] also considered the uniform metric but with different optimization measure. Their objective was to maximize the reduction in the cost from that of the schedule without a buffer. They presented a 20-approximation algorithm for this version. This ratio was later improved to 9 by Bar-Yehuda and Laserson [2]. It is not known if the offline version of SBP on the uniform metric is NP-hard.

The offline version of the sorting buffers problem on the uniform metric as well as the line metric can be solved optimally using dynamic programming in $O(N^{k+1})$ time where N is the number of requests in the sequence. This follows from the observation that the algorithm can pick k requests to hold in the buffer from first i requests in $\binom{i}{k}$ ways when the $(i+1)$th request arrives. Suppose there is a constraint that a request has to be served within D time steps of it being released, then, the dynamic program can be modified to compute the optimal schedule in $O(D^{k+1})$ time.

The dial-a-ride problem with finite capacity considered by Charikar and Raghavachari [5] is related to our problem. In this problem, the input is a sequence of requests each of which is a source-destination pair in a metric space on n points. For each request, an object has to be transferred from the source to the destination. The goal is to serve all the requests using a vehicle of capacity k so that total length of the tour is minimized. The non-preemptive version requires that once an object is picked, it can be dropped only at its destination while in the preemptive version the objects can be dropped at intermediate locations and picked up later. Charikar and Raghavachari [5] give approximation algorithms for both preemptive and non-preemptive versions using Bartal's metric embedding result. Note, however, that the disc scheduling problem enforces a sequencing constraint that is not imposed by the dial-a-ride problem. In the disc scheduling, if we are serving the ith request, then it is necessary that at most k requests from first to $(i-1)$th request be outstanding. Whereas, in the dial-a-ride problem, the requests can be served in any order as long as they meet the capacity requirement. Therefore the techniques used by them can not be used directly for the disc scheduling problem. They give $O(\log n)$ approximation for the preemptive case and $O(\sqrt{k}\log n)$ approximation for the non-preemptive case. The capacitated vehicle routing problem considered by Charikar et al. [4] is a variant in which all objects are identical and hence an object picked from a source can be dropped at any of the destinations. They give the best known approximation ratio of 5 for this problem and survey the previous results.

1.2 Our results

We first show in Section 3 that natural strategies such as First-In-First-Out and Nearest-First are not appropriate for this problem and that they have a competitive ratio of $\Omega(k)$. We also show that any deterministic algorithm that takes decisions just based on the (unordered) set of requests which are currently in the buffer has a competitive ratio of $\Omega(k)$.

Next in Section 4, we provide the first non-trivial competitive ratio for the online SBP on a line metric. For a line metric $\{1,\ldots,n\}$ with distance between i and j being $|i-j|$, we present a randomized online algorithm with a competitive ratio of $O(\log^3 n)$ in expectation against an oblivious adversary. This algorithm also yields a competitive ratio of $O(\alpha^{-1}\log^3 n)$ if we are allowed to use a buffer of size αk for any $1 \leq \alpha \leq \log n$.

Our algorithm is based on the probabilistic embedding of the line metric into the so-called hierarchical well-separated trees (HSTs) first introduced by Bartal [3]. Bartal proved that any metric on n points can be probabilistically approximated within a factor of $O(\log n \log\log n)$ by metrics on HSTs. This factor was later improved to $O(\log n)$ by Fakcharoenphol et al. [6]. In fact it is easy to see that the line metric $\{1,\ldots,n\}$ can be probabilistically approximated within a factor of $O(\log n)$ by the metrics induced by binary trees of depth $1+\log n$ such that the edges in level i have length $n/2^i$. We provide a simple lower bound on the cost of the optimum on a tree metric by counting how many times it must cross a particular edge in the tree. Using this lower bound, we prove that the expected cost of our algorithm is within the factor of $O(\log^3 n)$ of the optimum.

Our algorithm generalizes naturally to other "line-like" metric spaces. More precisely, consider a metric such that for every subset of points in the metric space, the cost of the minimum spanning tree on that subset is within α times the maximum pairwise distance in that subset. For such a metric on n points, our algorithm is $O(\alpha \log^2 n \log D)$ competitive in expectation, where D is the aspect ratio, i.e., the ratio of the maximum pairwise distance to the minimum pairwise distance in the metric. The factor of $\log D$ is coming from the height of $\log D$ of an HST approximation of such a metric space and the two factors of $\log n$ are due to approximations by HSTs. Since for a line metric on n uniformly spaced points, we have $\alpha = 1$ and $D = n$, we obtain an overall $O(\log^3 n)$ bound.

2 The Sorting Buffers Problem

Let (V, d) be a metric on n points. The input to the Sorting Buffers problem consists of a sequence of N requests. The ith request is labeled with a point $p_i \in V$. There is a server which is initially located at a point $p_0 \in V$. To serve ith request, the server has to visit p_i after its arrival. There is a sorting buffer which can hold up to k requests at a time. The first k requests arrive initially. The $(i+1)$th request arrives after we have served at least $i+1-k$ requests among the first i requests for $i \geq k$. Thus we can keep at most k requests pending at any time. The output is such a legal schedule (or order) of serving the requests. More formally, the output is given by a permutation π of $1, \ldots, N$ where the ith request to be served is denoted by $\pi(i)$. Since we can keep at most k requests pending at a time, a legal schedule must satisfy that the ith request to be served must be among first $i+k-1$ requests arrived, i.e., $\pi(i) \leq i+k-1$. The cost of the schedule is the total distance that the server has to travel, i.e., $C_\pi = \sum_{i=1}^{N} d(p_{\pi(i)}, p_{\pi(i-1)})$ where $\pi(0) = 0$ corresponds to the starting point. The Online Sorting Buffers problem (SBP) is to find a legal schedule π that minimizes C_π where the $(i+1)$th request is revealed only after serving at least $i-k+1$ requests for $i \geq k$. In the offline version, the entire input sequence is known up-front.

2.1 The disc model

Overlooking certain operational details, a disc is modeled as an arrangement of tracks numbered $1, \ldots, n$. The time taken to move the head from track i to track j is assumed to be $|i-j|$. The Disc Scheduling problem is the SBP on the line metric space $(\{1, \ldots, n\}, d)$ where $d(i,j) = |i-j|$. It is not known if the offline problem is NP-hard. We argue in Section 4.1 that the algorithm that serves the requests in the order they arrive is an $O(k)$-approximation. To the best of our knowledge, no algorithm with a better guarantee was known before.

3 Why Some Natural Strategies Fail

Many natural deterministic strategies suffer from one of the two following drawbacks.

1. Some strategies block large part of the sorting buffer with requests that are kept pending for a long time. Doing this, the effective buffer size drops well below k, thereby yielding a bad competitive ratio. For example, consider the Nearest-First (also called Shortest-Trip-First (STF)) strategy that always serves the request nearest to the current head location. Suppose that the initial head location is 0. Let the input

sequence be $3, \ldots, 3, 1, 0, 1, 0, \ldots$ where there are $k-1$ requests to 3. The STF strategy keeps the $k-1$ requests to 3 pending and serves the 1s and 0s alternatively using an effective buffer of size 1. The optimum schedule, on the other hand, gets rid of the requests to 3 by making a single trip to 3 and uses the full buffer to serve the remaining sequence. It is easy to see that if the sequence of 1s and 0s is large enough, the cost of STF is $\Omega(k)$ times that of the optimum.

2. Some other strategies, in an attempt to free the buffer slots, travel too far too often. The optimum, however, saves the distant requests and serves about k of them at once by making a single trip. Consider, for example, the First-In-First-Out (FIFO) strategy that serves the requests in the order they arrive. Suppose again that the initial head location is 0 and the input sequence is a repetition of the following block of requests: $n, 1, \ldots, 1, 0, \ldots, 0$ where there are k requests to 1 and k requests to 0 in each block. The FIFO strategy makes a trip to n for each block while the optimum serves 1s and 0s for $k-1$ blocks and then serves the accumulated $k-1$ requests to n by making a single trip to n. Note that, in doing this, the effective buffer size for the optimum reduces from k to 1. However, for the sequence of k 1s followed by k 0s, having a buffer of size k is no better than having buffer of size 1. It is now easy to see that if $n = k$, then FIFO is $\Omega(k)$ worse than the optimum.

Thus a good strategy must necessarily strike a balance between clearing the requests soon to free the buffer and not traveling too far too often. Obvious combinations of the two objectives, like making decisions based on the ratios of the distance traveled to the number of requests served also fail on similar input instances. We refer the reader to Racke et al. [8] for more discussion.

3.1 Memoryless Deterministic Algorithms

We call a deterministic algorithm *memoryless* if it makes its scheduling decisions based purely on the set of pending requests in the buffer. Such an algorithm can be completely specified by a function ρ such that for every possible (multi-)set S of k requests in the buffer, the algorithm picks a request $\rho(S) \in S$ to serve next. The Nearest-First strategy is an example of a memoryless algorithm. In this section, we prove the following theorem.

Theorem 1 *Any memoryless algorithm is $\Omega(k)$-competitive.*

Proof. We consider $k+1$ points $S = \{1, k, k^2, \ldots, k^k\}$ on a line. Consider any deterministic memoryless algorithm A. Suppose the head is initially at 1. We start with a request at each of the points $k, \ldots, k^k$. Whenever A moves the head from location p_i to location p_j, we release a new request at p_i. Thus, at all times, the pending requests and the head location together span all the $k+1$ points in S. Construct a directed graph G on S as vertices as follows. Add an edge from p_i to p_j if from the configuration with the head at p_i (with pending requests at all other points), A moves the head to p_j. Observe that the out-degree of every vertex is one and that G must have a cycle reachable from 1.

Suppose there is a cycle of length two between points $p_i, p_j \in S$ that is reachable from 1. In this case, we give an input sequence as follows. We first follow the path from 1 to p_i so that the head now resides at p_i. We then give a long sequence of the requests of the form $p_j, p_i, p_j, p_i, \ldots$. A serves p_js and p_is alternatively, keeping the other $k-1$ requests pending in the buffer. The optimum algorithm will instead clear the other $k-1$ requests first and use the full buffer to save an $\Omega(k)$ factor in the cost. Note that this situation is "blocking-the-buffer" (item 1) in the discussion on why some strategies fail.

Suppose, on the other hand, all the cycles reachable from 1 are of length greater than two. Consider such a cycle C on $p_1, p_2, \dots p_c \in S$ where $c > 2$ and $p_1 > p_2, \dots, p_c$. Note that the edges (p_1, p_2) and (p_c, p_1) have lengths that are $\Omega(k)$ times the total lengths of all the other edges in C. We now give an input sequence as follows. We first make A bring the head at p_1 and then repeat the following block of requests several times: $p_2, p_3, \dots, p_c$. For each such block, A makes a trip of C while the optimum serves $p_2, \dots, p_c$ repeatedly till it accumulates $k-1$ requests to p_1 and then clears all of them in one trip to p_1. Thus overall it saves an $\Omega(k)$ factor in the cost over A. Note that this situation is "too-far-too-often" (item 2) in the discussion on why some strategies fail.

Thus, any deterministic memoryless strategy has a competitive ratio of $\Omega(k)$. ∎

Note that $n = k^k$ in the above example. Thus the lower bound we proved in terms of n is $\Omega(\frac{\log n}{\log \log n})$. We do not know how to prove a better lower bound in terms of n.

4 Algorithm for Sorting Buffers on a Line

4.1 A lower bound on OPT for a tree metric

Consider first an instance of SBP on a two-point metric $\{0, 1\}$ with $d(0, 1) = 1$. Let us assume that $p_0 = 0$. There is a simple algorithm that behaves optimally on this metric space. It starts by serving all requests at 0 till it accumulates k requests to 1. It then makes a transition to 1 and keeps serving requests to 1 till k requests to 0 are accumulated. It then makes a transition to 0 and repeats. It is easy to see that this algorithm is optimal. Consider the First-In-First-Out algorithm that uses a buffer of size 1 and serves any request as soon as it arrives. It is easy to see that this algorithm is $O(k)$-competitive since it makes $O(k)$ trips for every trip of the optimum algorithm.

We use $\mathsf{OPT}(k)$ to denote both the optimum algorithm with a buffer of size k and its cost. The following lemma states that $\mathsf{OPT}(k)$ scales linearly with the buffer size.

Lemma 1 *For a two-point metric, for any $1 \le l \le k$, we have $\mathsf{OPT}(k) \ge \mathsf{OPT}(\lceil k/l \rceil)/2l$.*

Proof. Let $p_0 = 0$. By the time $\mathsf{OPT}(\lceil k/l \rceil)$ makes l trips to 1, we know that $\mathsf{OPT}(k)$ must have accumulated at least k requests to 1. Therefore $\mathsf{OPT}(k)$ must travel to 1 at least once. Note that the l (round) trips to 1 cost at most $2l$ for $\mathsf{OPT}(\lceil k/l \rceil)$. We can repeat this argument for every trip of $\mathsf{OPT}(k)$ to conclude the lemma. ∎

Using the above observations, we now present a lower bound on the optimum cost for the sorting buffers problem on a tree metric.[1] Consider a tree T with lengths $d_e \ge 0$ assigned to the edges $e \in T$. Let $p_1, \dots, p_N$ denote the input sequence of points in the tree. Refer to Figure 1. Fix an edge $e \in T$. Let L_e and R_e be the two subtrees formed by removing e from T. We can shrink L_e to form a super-node 0 and shrink R_e to form a super-node 1 to obtain an instance of SBP on a two-point metric $\{0, 1\}$ with $d(0, 1) = d_e$. Let LB_e denote the cost of the optimum on this instance. It is clear that any algorithm must spend at least LB_e for traveling on edge e. Thus

$$\mathsf{LB} = \sum_{e \in T} \mathsf{LB}_e \tag{1}$$

[1] Recall that a tree metric is a metric on the set of vertices in a tree where the distances are defined by the path lengths between the pairs of points.

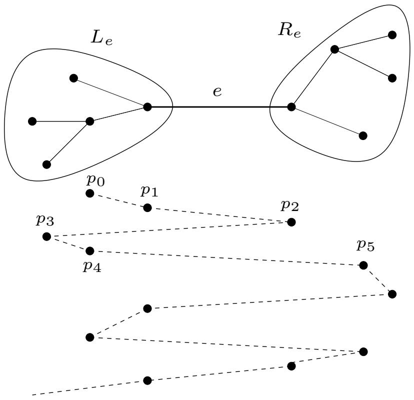

Figure 1: Lower bound contributed by edge e in the tree

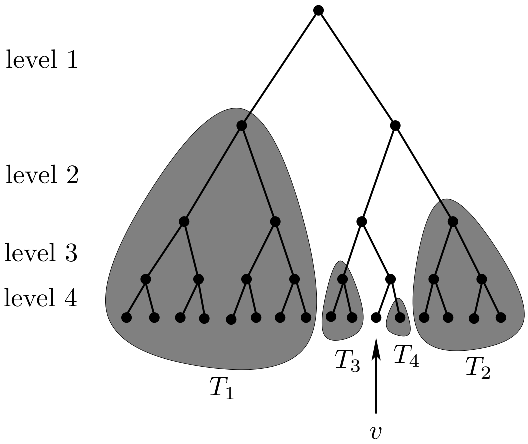

Figure 2: Partition of the leaves in a phase of the algorithm

is a lower bound on the cost OPT of the optimum on the original tree instance. Again, the algorithm that serves the requests in the order they arrive in, is $O(k)$-competitive.

Let $\mathsf{LB}(k)$ denote the above lower bound on $\mathsf{OPT}(k)$, the optimum with a buffer of size k. The following lemma follows from Lemma 1.

Lemma 2 *For any $1 \leq l \leq k$, we have $\mathsf{LB}(k) \geq \mathsf{LB}(\lceil k/l \rceil)/2l$.*

4.2 An algorithm on a binary tree

Consider a rooted binary tree T on $n = 2^h$ leaves. The height of this tree is $h = \log n$. The edges are partitioned into levels 1 to h according to their distance from the root; the edges incident to the root are in level 1 while the edges incident to the leaves are in level h. Figure 2 shows such a tree with $n = 8$ leaves. Let each edge in level i have cost $n/2^i$. Consider a metric on the leaves of this tree defined by the path lengths. In this section, we present a deterministic online algorithm for SBP on this metric that has a competitive ratio of $O(\log^2 n)$. Since the First-In-First-Out algorithm has a competitive ratio of $O(k)$, we assume that $k > h = \log n$. We also assume for simplicity that h divides k.

4.2.1 Algorithm.

The algorithm goes in phases. Suppose that in the beginning of a phase, the server is present at a leaf v as shown in Figure 2. We partition the leaves other than v into h subsets as shown in the figure. Consider the path P_v from v to the root. Let T_i be the tree hanging to the path P_v at level i for $1 \leq i \leq h$. Let V_i be the set of leaves in T_i. We think of the sorting buffer of size k as being divided into h sub-buffers of size k/h each. We associate the ith sub-buffer with V_i, i.e., we accumulate all the pending requests in V_i in the ith sub-buffer. The algorithm maintains the following invariant.

> Invariant. Each of the h sub-buffers has at most k/h requests, i.e., there are at most k/h pending requests in any V_i.

We input new requests till one of the sub-buffers overflows. Suppose that the $j$the sub-buffer overflows. The algorithm then clears all the pending requests in the subtrees $T_j, T_{j+1}, \ldots, T_h$ by performing an Eulerian tour of the trees $T_j, \ldots, T_h$. At the end of the tour, the head is at an arbitrary leaf of T_j. The algorithm, then, enters the next phase.

4.2.2 Analysis.

To prove the correctness of the algorithm we have to argue that at most k requests are pending at any point in the algorithm. This is in fact guaranteed by the invariant.

Lemma 3 *The invariant is satisfied in the beginning of any phase.*

Proof. Initially, when no requests have arrived, the invariant is trivially satisfied. Suppose that it is satisfied in the beginning a phase and that the jth sub-buffer overflows in that phase. The division into trees and sub-buffers changes after the move. However since the server resides in T_j, all the trees $T_1, \ldots, T_{j-1}$ and their corresponding sub-buffers remain unchanged. Also since the algorithm clears all the pending requests in the trees $T_j, \ldots, T_h$, the jth to hth sub-buffers after the move are all empty. Thus the invariant is also satisfied in the beginning of the next phase. ∎

Next we argue that the algorithm is $O(\log^2 n)$ competitive.

Theorem 2 *The total distance traveled by the server in the algorithm is $O(\mathsf{OPT} \cdot \log^2 n)$.*

Proof. For an edge $e \in T$, let $\mathsf{LB}_e(k)$ be the lower bound on $\mathsf{OPT}(k)$ contributed by e as defined in Section 4.1. Let $\mathsf{LB}(k) = \sum_e \mathsf{LB}_e(k)$. Let $\mathsf{LB}_e(k/h)$ and $\mathsf{LB}(k/h)$ be the corresponding quantities assuming a buffer size of k/h. We know from Lemma 2 that

$$\mathsf{OPT}(k) \geq \mathsf{LB}(k) \geq \mathsf{LB}(k/h)/2h = \mathsf{LB}(k/h)/2\log n.$$

To prove the lemma, next we argue that the total cost of the algorithm is $O(\mathsf{LB}(k/h) \cdot \log n)$.

To this end, consider a phase t. Suppose jth sub-buffer overflows in this phase. Let v be the leaf corresponding to the current position of the server and let u be the vertex on the path from v to the root between the levels j and $j-1$. In this phase, the algorithm spends at most twice the cost of subtree below u. Let e be the parent edge of tree T_j, i.e., the edge that connects T_j to u. Note that the cost of e is $n/2^j$ while the total cost of the subtree below u is $2(\log n - j) \cdot n/2^j = O(\log n \cdot n/2^j)$. With a loss of factor $O(\log n)$, we charge the cost of clearing the requests in $T_j \cup \cdots \cup T_h$ to the cost paid in traversing e in this phase. We say that the phase t transfers a charge of $n/2^j$ to e.

Now fix an edge $e \in T$. Let C_e denote the total charge transfered to e from all the phases. We show that $C_e \leq \mathsf{LB}_e(k/h)$. Refer to Figure 1. Let L_e and R_e be the two subtrees formed by removing e from T. Now C_e is the total cost paid by our algorithm for traversing e in the phases which transfer a charge to e. Note that in these phases, we traverse e to go from L_e to R_e or vice-versa only when there are at least k/h pending requests on the other side. Thus C_e is a at most $\mathsf{LB}_e(k/h)$, the lower bound contributed by e assuming a buffer size of k/h. Thus, $\sum_e C_e \leq \sum_e \mathsf{LB}_e(k/h) = \mathsf{LB}(k/h)$ and the proof is complete. ∎

Lemma 4 *For any $1 \leq \alpha \leq \log n$, there is an $O(\alpha^{-1} \log^2 n)$ competitive algorithm for SBP on the binary tree metric defined above if the algorithm is allowed to use a buffer of size αk.*

The algorithm is similar to the above one except that it assigns a sub-buffer of size $\alpha k/h$ to each of the h subtrees. The proof that it has the claimed competitive ratio is similar to that of Theorem 2 and is omitted.

4.3 An algorithm on a line metric

Our algorithm for a line metric is based on the probabilistic approximation of the line metric by a binary tree metric considered in the previous section. We first define some notions.

Definition 1 (Bartal [3]) *A set of metric spaces $\mathcal{S}$ over a set of points V, α-probabilistically approximates a metric space M over V, if*

- *every metric space in $\mathcal{S}$ dominates M, i.e., for each $N \in \mathcal{S}$ and $u, v \in V$ we have $N(u,v) \geq M(u,v)$, and*
- *there exists a distribution over the metric spaces $N \in \mathcal{S}$ such that for every pair $u, v \in V$, we have $\mathbb{E}[N(u,v)] \leq \alpha M(u,v)$.*

Definition 2 *A r-hierarchically well-separated tree (r-HST) is a edge-weighted rooted tree with the following properties.*

- *The weights of edges between a node to any of its children are same.*
- *The edge weights along any path from root to a leaf decrease by at least a factor of r.*

Bartal [3] showed that any connected edge-weighted graph G can be α-probabilistically approximated by a family r-HSTs where $\alpha = O(r \log n \log\log n)$. Fakcharoenphol, Rao and Talwar [6] later improved this factor to $\alpha = O(r \log n)$. It is very easy to see that the following lemma holds. We provide the proof for completeness.

Lemma 5 *A line metric on uniformly-spaced n points can be $O(\log n)$ probabilistically approximated by a family of binary 2-HSTs.*

Proof. Assume for simplicity that $n = 2^h$ for some integer h. Let M be a metric on $\{1, \ldots, n\}$ with $M(i,j) = |i - j|$. Consider a binary tree T on $2n$ leaves. Label the leaves from left to right as $l_1, \ldots, l_{2n}$. Partition the edges into levels as shown in Figure 2, i.e., the edges incident to the root are in level 1 and those incident to the leaves are in level $1 + \log n$. Assign a weight of $n/2^i$ to each edge in level i. Now pick r uniformly at random from the set $\{0, 1, \ldots, n-1\}$. Let N be the metric induced on the leaves $l_{r+1}, l_{r+2}, \ldots, l_{r+n}$ and consider a bijection from $\{1, \ldots, n\}$ to $\{l_{r+1}, \ldots, l_{r+n}\}$ that maps i to l_{r+i}.

It is easy to see that under this mapping, the metric space N dominates M. Now consider any pair i and $i+1$. It is easy to see that $\mathbb{E}[N(l_{r+i}, l_{r+i+1})] = O(\log n)$. By linearity of expectation, we have that for any pair $1 \leq i, j \leq n$, we have $\mathbb{E}[N(l_{r+i}, l_{r+j})] = O(\log n \cdot |i-j|)$. Therefore this distribution over the binary 2-HSTs forms $O(\log n)$-approximation to the line metric as desired. ∎

It is now easy to extend our algorithm on binary 2-HSTs to the line metric. We first pick a binary 2-HST from the distribution that gives $O(\log n)$ probabilistic approximation to the line metric. Then we run our deterministic online $O(\log^2 n)$-competitive algorithm on this binary tree. It is easy to see that the resulting algorithm is a randomized online algorithm that achieves a competitive ratio of $O(\log^3 n)$ in expectation against an oblivious adversary. It is necessary that the adversary be oblivious to our random choice of the 2-HST metric. Again for $1 \leq \alpha \leq \log n$, we can improve the competitive ratio to $O(\alpha^{-1} \log^3 n)$ by using a buffer of size αk.

4.4 Extension to "line-like" metrics

Our algorithm generalizes naturally to other "line-like" metric spaces. Consider a metric such that for every subset of points in the metric space, the cost of the minimum spanning tree on that subset is within α times the maximum pairwise distance in that subset. For such a metric on n points, we next argue that our algorithm is $O(\alpha \log^2 n \log D)$ approximation, where D is the aspect ratio, i.e., the ratio of the maximum pairwise distance to the minimum pairwise distance in the metric. Since for a line metric on n uniformly spaced points, we have $\alpha = 1$ and $D = n$, we obtain an overall $O(\log^3 n)$ approximation.

Now for the algorithm on such metrics, we first approximate the given metric by 2-HSTs of height $h = O(\log D)$. Our algorithm on such an HST assigns a buffer of size k/h to each subtree. We get a factor of $\log D$ from here. Recall that at the end of each phase, we clear requests in a subtree and charge this cost to the lower bound contributed by a top-edge of the subtree. We now argue that the cost of such a clearing step is within $O(\alpha \log n)$ factor of the cost of the top-edge.

To see this, we first observe that the cost of the top-edge is in fact within a constant factor of the maximum pairwise distance (diameter) of the subset S of points of the original metric space assigned in the subtree. This follows from the construction of the HSTs in Fakcharoenphol et al. [6]. Now observe that the expected cost of the subtree is $O(\log n)$ times of the cost of the minimum spanning tree on S which is, by our assumption, within a factor of α of the diameter of S. Thus the overall expected cost of the subtree is $O(\alpha \log n)$ times the cost of the top-edge. The rest of the analysis is similar and is omitted. We get an overall approximation guarantee of $O(\alpha \log^2 n \log D)$.

5 Conclusions

No complexity results are known for sorting buffers on a uniform metric or a line metric. This includes hardness results for the offline versions and lower bounds on competitive ratios for online algorithms. We showed some lower bounds on the competitive ratios for some simple strategies on a line metric. Any further results in this direction will be very interesting. The $O(\log^2 k)$ approximation algorithm of Racke et al. [8] for the uniform metric is deterministic. It looks unlikely that a deterministic strategy will give a non-trivial approximation ratio for a line metric. An $\Omega(k)$ lower bound on the competitive ratio of any deterministic algorithm for a line metric would be interesting. One may be able to prove that our algorithm for a line metric is in fact $O(\log^2 n)$ (or even $O(\log n)$) competitive. Any polylog k approximation guarantee will also be interesting to prove. Now that the uniform metric and the line metric are handled, can we prove some non-trivial results for general HSTs, thereby proving some results for a general metric?

Acknowledgments

We would like to thank Peter Sanders for introducing us to this problem. Thanks to Varsha Dani and Tom Hayes for many useful discussions and for discovering a simple proof of the lower bound for memoryless algorithms. Thanks also to Kunal Talwar for suggesting extensions in the direction of "line-like" metrics.

References

[1] M. Andrews, M. Bender, and L. Zhang. New algorithms for disc scheduling. *Algorithmica*, 32(2):277–301, 2002.

[2] R. Bar-Yehuda and J. Laserson. 9-approximation algorithm for the sorting buffers problem. In *3rd Workshop on Approximation and Online Algorithms*, 2005.

[3] Y. Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *IEEE Symposium on Foundations of Computer Science*, pages 184–193, 1996.

[4] M. Charikar, S. Khuller, and B. Raghavachari. Algorithms for capacitated vehicle routing. *SIAM Journal of Computing*, 31:665–682, 2001.

[5] M. Charikar and B. Raghavachari. The finite capacity dial-a-ride problem. In *IEEE Symposium on Foundations of Computer Science*, pages 458–467, 1998.

[6] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *35th Annual ACM Symposium on Theory of Computing*, pages 448–455, 2003.

[7] J. Kohrt and K. Pruhs. A constant approximation algorithm for sorting buffers. In *LATIN 04*, pages 193–202, 2004.

[8] H. Racke, C. Sohler, and M. Westermann. Online scheduling for sorting buffers. In *Proceedings of the European Symposium on Algorithms*, pages 820–832, 2002.

[9] A. Silberschatz, P. Galvin, and G. Gagne. *Applied Operating System Concepts*, chapter 13. Mass-Storage Structure, pages 435–468. John Wiley and Sons, first edition, 2000. Disc Scheduling is discussed in Section 13.2.