IBM Research Report

Online Sorting Buffers On Line

Vinayaka Pandit
IBM Research Division
IBM India Research Lab
Block I, I.I.T. Campus, Hauz Khas
New Delhi - 110016. India.

Rohit Khandekar
University of California, Berkeley

IBM Research Division
Almaden - Austin - Beijing - Delhi - Haifa - T.J. Watson - Tokyo - Zurich

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 modelled 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 request 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 $O(\log^3 n)$ 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 determinisitc strategy which makes scheduling decisions based only on the content of the buffer has a competitive ratio of $\Omega(k)$.

1 Introduction

Disc scheduling is one of the fundamental problems 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 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 modelled 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. This 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 minimzing the overall cost of head movement. We call this problem the Sorting Buffers problem (SBP) on a line metric to be consistent with 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 SSF, STF, and CSCAN have been proposed. However, this problem has so far been not studied 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 give a 3/2 approximation for the problem by exploiting its relation to a special case of the asymmetric travelling salesman problem called ATSP-Δ. They also show that the problem can be solved optimally in polynomial time if the reachability function is linear. They leave the online problem with buffers, which is essentially SBP, 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. 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 pair-wise 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 $\Omega(\sqrt{n})$ 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. The main observation in constructing the dynamic program is the observation that the buffer can pick from one of the $\binom{i}{k}$ choices in buffering k requests from the first i requests when it receives the $(i+1)$th request. 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 and 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, Nearest-First, and Scan 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 buffer contents 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 $O(\log^3 n)$ 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 $1 \le \alpha \le \log n$.

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 cost of our algorithm is within the factor of $O(\log^3 n)$ of the optimum.

Our algorithm easily generalizes to the following metrics. We call a edge-weighted rooted tree "β-growing" if for any edge e in the tree, the total weight of the subtree below e is at most β times the weight of e. For a metric defined by the shortest paths in an β-growing tree, our algorithm has a competitive ratio of $O(\beta h)$ where h is the height of the tree. Since $O(\log n)$-growing trees of depth $O(\log n)$ approximate a line metric within a factor of $O(\log n)$, we get an overall competitive ratio of $O(\log^3 n)$ for a line metric.

Our algorithm also generalizes naturally to the metric spaces with the property that for any subset of points, the cost of the minimum spanning tree on those points is comparable to the maximum pair-wise distance for those points. (*** check!)

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 labelled 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 $i-k$ requests among the first i requests for $i \ge k$. Thus we can keep at most k requests pending at any time. The ouput 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 $i+k$ requests arrived, i.e., $\pi(i) \le i+k$. The cost of the schedule is the total distance that the server has

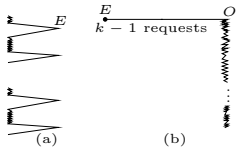

Figure 1: Two common ways of constructing worst case inputs

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 $i-k$ requests for $i \geq k$. In the offline version, the entire input sequence is known upfront.

2.1 The disc model

The disc is modelled as an arrangement of tracks numbered $1, \ldots, n$. The time taken to move a disc head from track i to track j is assumed to be $|i-j|$. The Disc Scheduling problem is the sorting buffers problem on this line metric space $(\{1, \ldots, n\}, d)$ where $d(i,j) = |i-j|$. We observe that any schedule which respects buffer constraints has an $O(k)$ approximatio ratio. To see this, note that, for every $2k$ consecutive crossings of a track t by the given schedule, any schedule using a buffer of size k has to cross t atleast once. When considered over all the tracks, it implies an $O(k)$ approximation.

3 Why Natural Strategies Fail

In this section, we show lower bounds for the competitive ratios for some strategies considered for other scheduling problems. We also show two ways of generating worst case inputs for deterministic strategies.

Consider the FIFO strategy and let the head be at track 0. The input repeats the following sequence many times over: $\{X, -1(k \text{ times}) - 1, 0(k+1 \text{ times})\}$ where X is a large integer. For every k trips to the X that the FIFO strategy makes, the optimal makes just one trip by buffering the k requests and serving the 0 and -1 requests as they come. Thus, FIFO is $\Omega(k)$-competitive.

Consider the "nearest first strategy", also called STF. Let the head be at track 0. Let the input be X (k times) followed by a sequence of $\{1, 0, 1, 0, \ldots 1, 0, \ldots\}$ where $X > 3$. The STF strategy keeps the requests for the track X in the buffer and shuffles between 0 and 1. When the sequence is very long, the optimal would be to first clear all the requests at X and then save a factor of $O(k)$ movement between 0 and 1. So, STF is also $\Omega(k)$-competitive.

We could show a lower bound of $\Omega(k)$ for many deterministic strategies, some involving complicated decision making. Two common ways of generating worst case inputs emerged. The first way, as in the case of FIFO schedule, is to generate an input in which the strategy fails to buffer far off requests. Consider Figure 1(a). The deterministic strategy fails to buffer the requests at the far end E whereas the optimal travels to the far end after buffering k such requests. The second way, as in the case of STF, is to force the strategy to keep the same set of requests in the buffer for long time. Consider Figure 1(b). Suppose there are $k-1$ requests at the end E. Let the sequence of nearby requests at O be very long. Optimal schedule will clear the requests at E and save an $\Omega(k)$ on the long sequence. The deterministic strategy will be forced to keep the requests at E pending and thus incur a cost of $\Omega(k)$ of the optimal.

3.1 Memoryless Deterministic Algorithms

We call a deterministic strategy to be *memoryless* if it makes its scheduling decisions based purely on the set of pending requests in the buffer. In this section, we prove the following theorem.

Theorem 3.1 *The lower bound on the approximation ratio for deterministic memoryless strategies is* $\Omega(k)$.

Proof. Consider $k+1$ points, denoted by S, on a straight line at distances of $1, k, \ldots, k^k$ from the origin. Let A be any deterministic memoryless strategy. Start with a request at each point. Whenever the strategy moves the head from p_i to p_j, release a new request at p_i. Thus, at all times, the pending requests and the head location together span all the points in S. Construct a directed graph G on the points in S as follows: add an edge (p_i, p_j) if the head moves from p_i to p_j when

there is a pending request at each of the points in S. Observe that the out-degree of every node is one and G must have a cycle.

Suppose there is a cycle of length two between $p_i, p_j \in S$. Let the head be at p_i (with pending requests at all other points). Consider a long sequence of the form $p_i, p_j, p_i, p_j, \ldots,$. The strategy A will traverse the cycle keeping the other $k-1$ requests in the buffer. The optimum algorithm will instead serve the other $k-1$ requests and use the buffer to save $\Omega(k)$ factor of the trips between p_i and p_j. Observe the similarity with the situation in Figure 1(b).

Suppose all the cycles are of length greater than two. Consider a cycle C between points $p_1, p_2, \ldots p_c \in S$ where $c > 2$. The points in S are such that, there must exist an edge $e \in C$ whose length is $O(k)$ times more than the total length of all the other edges in C. Let the edge e be from p_i to p_j. Let the head be at p_i with requests on all other points. Consider a long repeating sequence of $p_i, p_j, \ldots, p_c, p_1, \ldots, p_{i-1}$. If there are X repetitions of the cycle, then the strategy A will make X trips of the cycle C. The optimal on the other hand chooses to first serve all the requests except the one at p_j. After that, it clears the requests as they appear while buffering the requests to p_j. Thus, it makes $X/\Omega(k)$ trips to p_j. Thus, its overall cost is a factor $\Omega(k)$ less than that of A. Observe the similarity with the situation in Figure 1(a).

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

4 Algorithms for Sorting Buffers

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 does not use the buffer at all and serves any request as soon as it arrives. It is again easy to see that this algorithm is $O(k)$-competitive.

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

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 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 prove 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 $d_e \geq 0$ assigned to the edges $e \in T$. Let $p_1, \ldots, p_N$ denote the input sequence of points in the tree. Refer to Figure 2. 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 supernode 0 and shrink R_e to form a supernode 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 travelling on edge e. Thus

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

is a lower bound on the cost OPT of the optimum of the original tree instance. Again the First-In-First-Out algorithm that does not use the buffer and serves the requests as soon as they arrive 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 4.1.

Lemma 4.2 *For any $1 \leq l \leq k$, we have $\mathsf{LB}(k) \geq \mathsf{LB}(\lceil k/l \rceil)/2l$.*

[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.

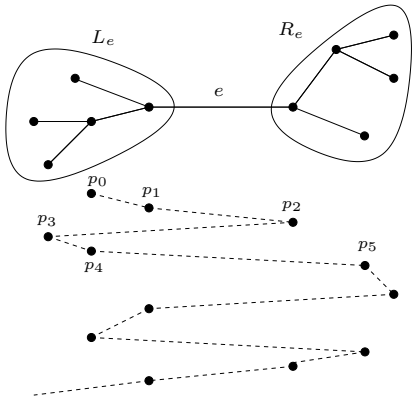

Figure 2: Lower bound contributed by edge e in the tree

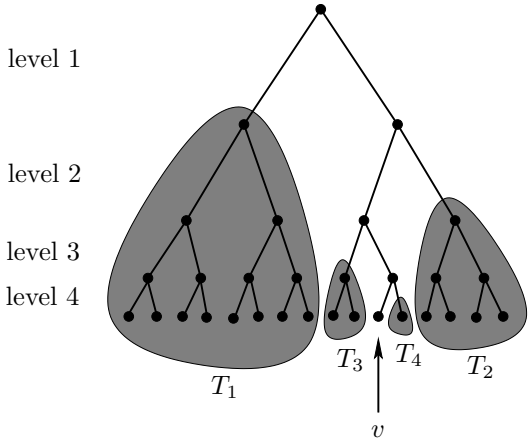

Figure 3: Partition of the leaves in a phase of the algorithm

4.2 An algorithm on a binary tree

Consider rooted a 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 3 shows such a tree with $n = 8$ leaves. Let each edge in level i have cost $n/2^i$. Consider a metric defined on the leaves of this tree. 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 3. 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 these sub-buffers overflows. Suppose that the jth sub-buffer overflows. The algorithm then clears all the pending requests in the subtrees $T_j, T_{j+1}, \ldots, T_h$. To serve these requests, the server performs a Eulerian traversal of the trees $T_j, \ldots, T_h$ and in the end resides at an arbitrary leaf of the tree T_j. The algorithm then goes to the next phase.

4.2.2 Analysis.

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

Lemma 4.3 *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 4.4 *The total distance travelled by the server in the algorithm is $O(\mathsf{OPT} \cdot \log^2 n)$. (*** is there an additive term?)*

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 4.2 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 now argue that $C_e \leq \mathsf{LB}_e(k/h)$. Refer to Figure 2. 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 since 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. Therefore we have $\sum_e C_e \leq \sum_e \mathsf{LB}_e(k/h) = \mathsf{LB}(k/h)$ and the proof is complete. ∎

Lemma 4.5 *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 4.4 and is omitted. The algorithm presented above generalizes naturally to the following metrices.

Definition 4.1 *An edge-weighted rooted tree is called* β-growing *if for any edge e in the tree, the total weight of the subtree below the parent of e is at most β times the weight of the edge e.*

Note that the binary tree considered in Section 4.2 is $O(\log n)$-growing tree. The proof of the following theorem is very similar to that of Theorem 4.4 and is omitted.

Theorem 4.6 *Consider a metric on the leaves of an β-growing tree where the distances are defined by the shortest path lengths. There is a deterministic online algorithm for SBP on this metric with a competitive ratio of $O(\beta h)$ where h denotes the height of the tree. Furthermore for any $1 \leq \alpha \leq h$, the competitive ratio can be improved to $O(\alpha^{-1}\beta h)$ using a buffer size of αk.*

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 4.2 (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 4.3 *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 4.7 *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 3, 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 that 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 $O(\log^3 n)$ 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.

5 Conclusions

No complexity results are known for even the special case of sorting buffers on uniform metrics. This includes hardness results for the offline version and lower bounds on competitive ratios for online algorithms. Any result in this direction will be very interesting. The $O(\log^2 k)$ approximation algorithm for the sorting buffers on uniform metrics is deterministic. It looks unlikely that a deterministic strategy will give non-trivial approximation ratios for the disc scheduling problem. An $O(k)$ lower bound on the competitive ratio of any deterministic strategy would be interesting.

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.