Reordering Buffer Management with a Logarithmic Guarantee in General Metric Spaces

Matthias Kohler[1] **and Harald Räcke**[2]

1 **Department of Informatics, Technical University of Munich, Munich, Germany**
kohler@in.tum.de
2 **Department of Informatics, Technical University of Munich, Munich, Germany**
raecke@in.tum.de

Abstract

In the reordering buffer management problem a sequence of requests arrive online in a finite metric space, and have to be processed by a single server. This server is equipped with a request buffer of size k and can decide at each point in time, which request from its buffer to serve next. Servicing of a request is simply done by moving the server to the location of the request. The goal is to process all requests while minimizing the total distance that the server is travelling inside the metric space.

In this paper we present a deterministic algorithm for the reordering buffer management problem that achieves a competitive ratio of $O(\log \Delta + \min\{\log n, \log k\})$ in a finite metric space of n points and aspect ratio Δ. This is the first algorithm that works for general metric spaces and has just a logarithmic dependency on the relevant parameters. The guarantee is memory-robust, i.e., the competitive ratio decreases only slightly when the buffer-size of the optimum is increased to $h = (1 + \epsilon)k$. For memory robust guarantees our bounds are close to optimal.

1998 ACM Subject Classification F.1.2 [Modes of Computation] Online Computation

Keywords and phrases Online algorithms, reordering buffer, metric spaces, scheduling

Digital Object Identifier 10.4230/LIPIcs.ICALP.2017.33

1 Introduction

In the reordering buffer management problem a sequence of requests arrive online in a finite metric space, and have to be processed by a single server. This server is equipped with a request buffer and can decide at each point in time, which request from its buffer to serve next. Servicing of a request is simply done by moving the server to the location of the request. The goal is to process all requests while minimizing the total distance that the server is traveling inside the metric space.

This simple, abstract model can be used for modeling context switching costs that occur in various applications in many different areas ranging from production engineering through computer graphics to information retrieval [7, 10, 17, 21]. In the online version of the problem the server does not see future requests but has to make its decision based on past requests and the requests it currently holds in the request buffer. The worst case ratio between the cost of the online algorithm and the cost of an optimal offline algorithm is called the *competitive ratio*.

We say a guarantee on the competitive ratio for the reordering buffer management problem is *memory robust* if the guarantee degrades gracefully as the buffer-size of the optimum algorithm is increased over the buffer-size of the online algorithm. More precisely, the ratio

44th International Colloquium on Automata, Languages, and Programming (ICALP 2017).
Editors: Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl;
Article No. 33; pp. 33:1–33:12


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

between the cost of the online algorithm with a buffer-size of k, and the cost of an optimum offline algorithm with a buffer-size of $h \geq k$ should be at most $O(ch/k)$.

The main result of this paper is a deterministic algorithm for the reordering buffer management problem that achieves a competitive ratio of $O(h(\log \Delta + \min\{\log n, \log k\})/k)$ in a finite metric space of n points and aspect ratio Δ. The algorithm is also $O(h)$-competitive because any reasonable algorithm achieves this competitive ratio (see Lemma 13 in the appendix). This is the first algorithm that works for general metric spaces and has just a logarithmic dependency on the relevant parameters. The algorithm and its analysis are also very simple.

It has been shown that even on a uniform metric ($\Delta = 1$) the competitive ratio of an online algorithm (deterministic or randomized) must be $\Omega(\log k)$ against an optimum algorithm with buffer-size $h \geq (1+\epsilon)k$ [1, 12]. Hence, an $O(\log k)$ term in the competitive ratio is unavoidable for memory robust algorithms.

Biénkowski et al. [9] have shown that for a sub-linear dependency on the buffer-size there needs to be another term in the competitive ratio apart from k for memory-robust algorithms. In particular, they give an instance on a line metric with n equidistant points for which any online algorithm looses a factor of $\Omega(\min\{k, \log n\})$ against an optimum algorithm with slightly larger buffer ($h = (1+\epsilon)k$). For this instance the number of points n is equal to the aspect ratio Δ. So a logarithmic dependency on the aspect ratio seems reasonable.

Englert and Räcke [12] present a deterministic, memory-robust algorithm for tree metrics of hop-diameter D that obtains a competitive ratio of $O(\frac{h}{k}(\log D + \log k))$. They then use the technique of approximating arbitrary metrics by tree-metrics due to Fakcharoenphol, Rao, and Talwar [14] to obtain a randomized $O(\frac{h}{k} \log n \cdot \log h)$-competitive algorithm for general metrics.

As a whole these results are uncomparable to our results. There exist metrics where the aspect ratio Δ is very small, but there are a lot of points, resulting in very poor guarantees from the result by Englert and Räcke. However, if one considers a star with edges of different length, the aspect ratio could be very high, but the hop-diameter in the tree is just 2, which makes the guarantee given by the result in [12] stronger than ours. One advantage of our result is that for general metrics the dependency on our parameters is logarithmic, while Englert and Räcke have the product of two logarithms. This product cannot easily be removed as any (memory-robust) algorithm that relies on the FRT-approximation will loose one logarithm because of FRT, and another because an online solution on a tree will have a logarithmic competitive ratio.

In Section 5 we deal with the question whether it is possible to trade the dependency on $\log \Delta$ in our competitive ratio for something else, like e.g. $\log n$, which does not depend on the aspect ratio. Our algorithm is memory restricted in the sense that it makes its decisions only depending on the content of the buffer, and on the content of an additional memory that contains k bits. We show, that for such a scenario there exist instances with an aspect ratio Δ, on which every memory-restricted algorithm with k bits is $\Omega(\sqrt{\log \Delta})$-competitive. This extends a lower bound due to Khandekar and Pandit [20] for memoryless algorithms.

1.1 Further Related Work

Most previous work on the reordering buffer management problem considers the case of uniform metrics. Räcke et al. [22] introduced the problem and developed a deterministic algorithm with competitive ratio $O(\log^2 k)$, which was subsequently improved to $O(\log k)$ by Englert and Westermann [13]. The analysis of both these algorithms can be slightly modified to give a memory-robust guarantee.

The first paper that used an analysis technique that is not memory-robust (and can therefore beat the $\Theta(\log k)$-guarantee) was due to Avigdor-Elgrabli and Rabani, who presented a deterministic algorithm with competitive ratio $O(\log k/\log\log k)$. This in turn was improved to a guarantee of $O(\sqrt{\log k})$ by Adamaszek et al. [2], which is close to optimal due to a lower bound of $\Omega(\sqrt{\log k/\log\log k})$ shown in the same paper. For randomized algorithms Avigdor-Elgrabli and Rabani present an $O(\log\log k)$-competitive algorithm [6]. This is optimal due to a corresponding lower bound proved by Adamaszek et al. [2].

For star metric spaces the result by Englert and Westermann [13] obtains a deterministic competitive ratio of $O(\log k)$. The result by Adamaszek et al. [2] gives an $O(\sqrt{\log k})$ guarantee for the case that $\Delta = O(\text{poly}(k))$. A straightforward extension to arbitrary values of Δ gives a competitive ratio of $O(\sqrt{\log k + \log\Delta})$. For the randomized case Avigdor-Elgrabli et al. [4] give an $O((\log\log(k\Delta))^2)$-competitive algorithm, i.e., an algorithm with a slight dependency on the aspect ratio of the metric space.

Gamzu and Segev [15] analyze the reordering buffer problem for n points on a line as this can be used to model the disc scheduling problem. They present a deterministic algorithm with a competitive ratio of $O(\log n)$.

In the offline case it has been shown that finding an optimal solution to the problem is NP-hard even on uniform metrics [3, 11]. Avigdor-Elgrabli and Rabani have given a constant factor approximation [5]. Im and Moseley gave an $O(\log\log(k\Delta))$-approximation for the star-metric [18] and subsequently improved this to $O(\log\log\log(k\Delta))$ [19]. Barman et al. [8] gave a bicriteria approximation algorithm that achieves an approximation guarantee of $O(\log n)$ when the buffer of the online algorithm is a constant factor *larger* than the buffer of the optimum algorithm. This works in general metric spaces.

1.2 The Model

An input sequence σ of requests has to be processed, where each request σ_i corresponds to some point in a finite metric space $M = (V, d)$. We use $n = |V|$ to denote the number of distinct points in M, and Δ to denote its *aspect ratio*, i.e., the ratio between the largest and smallest distance between two points. We assume w.l.o.g. that the minimum non-zero distance between two points is 1.

A *reordering buffer* that can store k requests can be used to rearrange the input sequence into an output sequence σ' in the following way. Initially, the buffer contains the first k requests of σ. In every time step t an algorithm has to select a request r from the buffer and append it to the output sequence, i.e., the algorithm sets σ'_t to r. If there are still requests waiting in the input sequence, the next such request takes r's place in the buffer; otherwise this place stays empty. The process is repeated until all requests from σ have been appended to the output sequence.

An online algorithm ALG has to make its decision based on the requests in the buffer and on the requests previously seen, but not based on future requests that are still to come. Suppose an algorithm ALG generates a request sequence σ' when giving a request sequence σ as input. The (true) cost $\text{ALG}_{true}(\sigma)$ is defined as

$$\text{ALG}_{true}(\sigma) = \sum_{i=1}^{\ell-1} d(\sigma'_i, \sigma'_{i+1}) \ ,$$

where ℓ is the length of the input sequence. Note that this means that the server may start its processing at the first request without incurring any cost for traveling to this location.

Throughout the paper we use a slightly different notation w.r.t. the optimum algorithm OPT for processing σ. Firstly, we assume that OPT has a larger buffer-size $h \geq k$. Secondly,

we denote the (true) cost of this algorithm with $\mathrm{OPT}(\sigma) = \mathrm{OPT}_{true}(\sigma)$. The reason is that for an online algorithm we will introduce an approximate (simplified) version of $\mathrm{ALG}_{true}(\sigma)$, which will be denoted with $\mathrm{ALG}(\sigma)$. Hence, for the online algorithm we need the differentiation between $\mathrm{ALG}(\sigma)$ and $\mathrm{ALG}_{true}(\sigma)$, whereas this is not required for the optimum algorithm.

2 The Algorithm

A *block-oriented* algorithm for the reordering buffer management problem serves requests in blocks. Whenever the buffer gets full, the algorithm identifies a set S of requests from the buffer and serves these requests. Additional requests that arrive while serving requests in S are ignored, and will not be considered until all requests in S have been handled. We call such a set S of requests chosen by the algorithm a *block*. The process of choosing and servicing blocks of requests is repeated until the end of the input sequence is reached. The requests in the buffer at this time form the last *block* of the algorithm.

For a block-oriented algorithm we can write down the (approximate) cost of the algorithm just in terms of the sequence $S_1, S_2, S_3, \ldots$ of generated blocks. This is done as follows. For a block S_i, we use $C(S_i)$ to denote the *cost of the block*, which is defined as the length of a shortest path that connects all requests in S_i (note that computing $C(S_i)$ is NP-hard). For two blocks S_i and S_j, we define the *distance* $d(S_i, S_j)$ between the two blocks, by

$$d(S_i, S_j) = \min_{r_i \in S_i, r_j \in S_j} d(r_i, r_j) \ ,$$

i.e., the distance of the closest pair $(r_i, r_j) \in S_i \times S_j$.

Suppose that for a request sequence σ a block-oriented algorithm generates a sequence $S_1, S_2, \ldots, S_\ell$ of blocks. We define the *cost* $\mathrm{ALG}(\sigma)$ of the algorithm by

$$\mathrm{ALG}(\sigma) = \sum_{i=1}^{\ell} C(S_i) + \sum_{i=1}^{\ell-1} d(S_i, S_{i+1}) \ . \tag{1}$$

We refer to the first term in Equation 1 as the *block cost* $\mathrm{ALG}_{bc}(\sigma)$ of the algorithm, and to the second term as the *connection cost* $\mathrm{ALG}_{cc}(\sigma)$. The following lemma shows that this definition of cost is close to the true cost of the algorithm. The fact that we can specify the cost of the algorithm just in terms of the generated blocks will greatly simplifies our analysis.

▶ **Lemma 1.** *We can implement any block-oriented algorithm* ALG *such that* $\mathrm{ALG}(\sigma) \leq \mathrm{ALG}_{true}(\sigma) \leq 3\,\mathrm{ALG}(\sigma)$.

Proof. In the following we describe how to serve all requests in a block S_i, and how to move to the next block S_{i+1}. Suppose the server is initially located at a request from S_i (for the first block S_1 we can assume this because according to our model the server may start at an arbitrary location). Since the requests in block S_i are known completely before serving its first request, we can efficiently compute an MST that covers all requests in S_i. The cost of traversing the elements by following the edges of the MST is at most $2C(S_i)$. Let (r_i, r_{i+1}) denote the request pair in $S_i \times S_{i+1}$ with minimum distance. We move, from our current location (after serving the last request from S_i) along a shortest path to r_{i+1}. The cost for this step is at most $C(S_i) + d(r_i, r_{i+1})$.

Repeating the above step for all blocks gives a total true cost $\mathrm{ALG}_{true}(\sigma)$ of at most $3\,\mathrm{ALG}_{bc}(\sigma) + \mathrm{ALG}_{cc}(\sigma) \leq 3\,\mathrm{ALG}(\sigma)$. ◀

In order to complete the description of the algorithm we need to describe how to choose a *good* block of requests for service when the buffer becomes full. For this we need a few definitions. For a set S of requests, and a value δ, $0 \leq \delta \leq 1$ we define the δ-fraction cost $C_\delta(S)$ of S as the minimum cost for servicing a δ-fraction of the elements from S. Formally,

$$C_\delta(S) = \min_{U \subseteq S, |U| \geq \delta \cdot |S|} C(U) \ .$$

Our algorithm StableSet is based on the following notion of a *large, stable set*. Intuitively, the cost for servicing the elements of a stable set S does not reduce by too much even if a large fraction of elements from S is removed.

▶ **Definition 2.** A set S of requests is (α, β, γ)-stable if the following holds

1. $C_{1-\alpha}(S) \geq \beta \cdot C(S)$ (stability constraint),
2. $|S| \geq \gamma k$ (size constraint).

In Section 4 we prove the following lemma showing that we can efficiently find stable sets with good parameters.

▶ **Lemma 3.** *Let V denote a set of k requests covering at most $\ell \leq k$ distinct locations in a metric space with aspect ratio Δ. There exists a polynomial time algorithm that finds an (α, β, γ)-stable subset $S \subseteq V$ with $\alpha \geq 1/(1 + \log \Delta + \log \ell)$, $\beta \geq 1/8$, and $\gamma \geq 1/e$.*

With these definitions our algorithm StableSet becomes very simple. When the buffer becomes full, choose a stable subset S from the elements of the buffer according to the algorithm implicit in Lemma 3. Then service this block of requests according to the algorithm in Lemma 1. This is repeated until the end of the input sequence is reached. The elements that still remain in the buffer form the last block of the algorithm.

3 Analysis

In the following we first describe a general approach to obtain a lower bound on the cost $\mathrm{OPT}(\sigma)$ of the optimal solution. Let $X_1, \ldots, X_\ell$ denote subsets of requests from the input sequence. We say that a subset X_i is *partially scheduled* by OPT at time t, if the first t requests in OPT's output sequence contain at least one but not all elements from X_i. The following claim gives a lower bound on the optimum cost.

▶ **Claim 4.** *Let $X_1, \ldots, X_\ell$ denote (not necessarily disjoint) subsets of requests from the input sequence, and suppose that at each point in time there are at most s subsets X_i that are partially scheduled by* OPT. *Then* $\mathrm{OPT}(\sigma) \geq \frac{1}{s} \sum_i C(X_i)$.

Proof. We associate an interval $[start(i), end(i)]$ with each set X_i, where $start(i)$ denotes the position of the first element of X_i that appears in OPT's output sequence, and $end(i)$ denotes the position of the last such element. Clearly, the cost of OPT for serving elements that lie between $start(i)$ and $end(i)$ is at least $C(X_i)$.

We can color the intervals with s colors such that intervals with the same color do not intersect. This holds because the interval graph corresponding to the set of intervals has a maximum clique size of s. Such interval graphs can be colored with s colors by a Greedy algorithm. Since sets X_i from the same color class do not interleave in OPT's output sequence the cost for serving all elements of a color-class is at least $\sum_{i \in I} C(X_i)$, where I denotes the index set of the color-class. As there must exist a color-class with cost at least $\frac{1}{s} \sum_i C(X_i)$ the claim follows. ◀

3.1 Analyzing Block Cost

The following lemma gives the bound on the block cost induced by an algorithm that uses stable sets.

▶ **Lemma 5.** *Let $S_1, S_2, \ldots, S_\ell$ denote the sequence of blocks generated by a block-oriented algorithm, where all but the last block are (α, β, γ)-stable, where $k \geq \frac{6}{\alpha\gamma}$. Then* $\mathrm{ALG}_{bc}(\sigma) \leq \left(\frac{6h}{\alpha\beta\gamma k} + 1\right) \cdot \mathrm{OPT}(\sigma)$.

Proof. Let $z = \lceil \alpha\gamma k/3 \rceil$. We obtain a set X_i $(i < \ell)$ by taking S_i and removing the first z and the last z requests from it that appear in OPT's output sequence. Then the cardinality of X_i is at least

$$|X_i| = |S_i| - 2z = |S_i| - 2\lceil \alpha\gamma k/3 \rceil \geq |S_i| - 2\alpha\gamma k/3 - 2 \geq |S_i| - \alpha\gamma k \geq (1-\alpha)|S_i| \ ,$$

where the second inequality holds for $k \geq \frac{6}{\alpha\gamma}$, and the final inequality holds due to the size constraint for block S_i. The stability constraint for S_i gives us that $C(X_i) \geq \beta C(S_i)$.

We show that at most $(h+k)/z$ sets X_i can be partially scheduled by OPT at any given time. Fix a time t. We define a partially scheduled set X_i to be of Type I if not all of S_i has already appeared in the input sequence, otherwise, we define it to be of Type II. For sets of Type II, OPT must hold the last z requests of S_i (according to the order given by OPT's output sequence) in its buffer, as these have already appeared. For sets of Type I, ALG must hold the first z requests of S_i in its buffer, as these have already appeared but ALG only starts removing elements from S_i after all of S_i has appeared. This means there can at most be k/z partially scheduled sets of Type I, and at most h/z partially scheduled sets of Type II. Applying Claim 4 gives that

$$\mathrm{OPT}(\sigma) \geq \frac{z}{h+k} \sum_{i=1}^{\ell-1} C(X_i) \geq \frac{\alpha\beta\gamma k}{6h} \sum_{i=1}^{\ell-1} C(S_i) \ . \tag{2}$$

Combining the definition of block-cost, Equation 2, and the fact that $\mathrm{OPT}(\sigma) \geq C(S_\ell)$ gives

$$\mathrm{ALG}_{bc}(\sigma) = \sum_{i=1}^{\ell} C(S_i) \geq \left(\frac{6h}{\alpha\beta\gamma k} + 1\right) \cdot \mathrm{OPT}(\sigma) \ ,$$

as desired. ◀

3.2 Analyzing Connection Cost

▶ **Lemma 6.** *Let $S_1, S_2, \ldots, S_\ell$ denote the sequence of blocks generated by a block-oriented algorithm, where all but the last block have cardinality at least γk. Then* $\mathrm{ALG}_{cc}(\sigma) \leq \left(\frac{5h}{\gamma k} + 1\right) \cdot \mathrm{OPT}(\sigma)$.

Proof. We use $\mathrm{ALG}'_{cc}(\sigma)$ to denote the connection cost, where we ignore the cost for connecting the last two blocks $S_{\ell-1}$ and S_ℓ. For every pair of consecutive blocks S_i, S_{i+1}, $i \leq \ell - 2$ we generate $\lceil \gamma k \rceil$ request-pairs by matching $\lceil \gamma k \rceil$ requests from S_i to $\lceil \gamma k \rceil$ requests from S_{i+1} in an arbitrary manner. Let X_i^r, $i \in \{1, \ldots, \ell-2\}, r \in \{1, \ldots, \lceil \gamma k \rceil\}$ denote the request pairs generated this way. We have

▶ **Fact 7.** $\sum_{i,r} C(X_i^r) \geq \gamma k \cdot \mathrm{ALG}'_{cc}(\sigma)$.

To see this, observe that for a request-pair X_i^r we have $C(X_i^r) \geq d(S_i, S_{i+1})$, as the request pair connects sets S_i and S_{i+1}. Since for every i we have at least γk requests the fact holds. The following fact allows us to apply Claim 4.

▶ **Fact 8.** *At any given time, there exist at most* $5h$ *request pairs from sets* X_i^r *that are partially scheduled by* OPT.

Proof. Fix a time step t. Suppose we have a request pair that is partially scheduled by OPT at time t. We say that it is of Type I if it has not been scheduled by ALG at all (by time step t); it is of Type II if ALG has already scheduled both requests of the pair; and it is of Type III, otherwise.

There can be at most $2h$ request pairs of Type II, because OPT must hold the second request of such a pair in its buffer, and any request can belong to at most two pairs. There can be at most $2k$ request pairs of Type I, because ALG must hold the first request of such a pair in its buffer, as this request has already appeared but ALG has not scheduled it. Finally, observe that there are at most k pairs that are partially scheduled by ALG at any point in time. Hence, the total number of partially scheduled requests of Type III is at most k.

Altogether there exists at most $3k + 2h \leq 5h$ request pairs that are partially scheduled by OPT. ◀

Combining Claim 4 with the above fact gives $\mathrm{OPT}(\sigma) \geq \frac{1}{5h} \sum_{i,r} C(X_i^r)$. Together with Fact 7 we obtain

$$\mathrm{ALG}_{cc}(\sigma) \leq \mathrm{ALG}'_{cc}(\sigma) + \mathrm{OPT}(\sigma) \leq \frac{1}{\gamma k} \sum_{i,r} C(X_i^r) + \mathrm{OPT}(\sigma) \leq \left(\frac{5h}{\gamma k} + 1\right) \cdot \mathrm{OPT}(\sigma) \ ,$$

as desired. The first inequality uses the fact that the cost for connecting the two last blocks is at most $\mathrm{OPT}(\sigma)$. ◀

3.3 Proof of the Main Result

Combining the analysis of the block cost and the connection cost gives our main theorem.

▶ **Theorem 9.** *A block-oriented algorithm that only chooses* (α, β, γ)*-stable blocks is* $O(\frac{h}{k} \cdot (\alpha\beta\gamma)^{-1})$*-competitive, against an optimal algorithm with buffer size* $h \geq k$*. Using the stable set computation from Lemma 3 gives a competitive ratio of* $O(h(\log \Delta + \min\{\log n + \log k\})/k)$ *in a metric space of* n *points and aspect ratio* Δ*.*

Proof. For the case that $k \geq \frac{6}{\alpha\gamma}$ we can simply combine the bounds in Lemma 5 and Lemma 6. For the case that $k \leq \frac{6}{\alpha\gamma}$ we use the fact that any algorithm is $O(h)$-competitive which gives the result since $O(h) = O(\frac{h}{k} \cdot k) = O(\frac{h}{k}(\alpha\beta\gamma)^{-1})$. ◀

4 Finding Stable Sets

In this section we present an algorithm for finding stable sets.

▶ **Lemma 3.** *Let* V *denote a set of* k *requests covering at most* $\ell \leq k$ *distinct locations in a metric space with aspect ratio* Δ*. There exists a polynomial time algorithm that finds an* (α, β, γ)*-stable subset* $S \subseteq V$ *with* $\alpha \geq 1/(1 + \log \Delta + \log \ell)$*,* $\beta \geq 1/8$*, and* $\gamma \geq 1/e$*.*

Proof. Set $\beta' := 1/2$ and $\alpha := 1/(1 + \log_2 \Delta + \log_2 \ell)$. For a subset S of requests we define $\mathrm{MST}_{1\text{-}\alpha}(S)$ to be a minimum spanning tree among at least $\lceil (1-\alpha)|S| \rceil$ requests from S. It is NP-hard to find such an MST but there is a 2-approximation algorithm that returns a tree $T_{1-\alpha}$ that spans $\lceil (1-\alpha)|S| \rceil$ requests and has cost $\mathrm{cost}(T_{1-\alpha}) \leq 2\,\mathrm{cost}(\mathrm{MST}_{1\text{-}\alpha}(S))$ [16].

Our algorithm for finding a stable set proceeds as follows. Initially it sets $S := V$. Then it (approximately) checks whether S is stable. For this it computes an approximation

$T_{1-\alpha}$ to $\text{MST}_{1\text{-}\alpha}(S)$ according to the algorithm by Garg [16]. Then it checks whether $\text{cost}(T_{1-\alpha}) \geq \beta' \,\text{cost}(\text{MST}(S))$. If this is the case the set S is returned. Otherwise the algorithm sets $S := V(T_{1-\alpha})$, where $V(T_{1-\alpha})$ is the vertex set of tree $T_{1-\alpha}$, and repeats the process. In the following we prove that the set S returned by the algorithm fulfills the desired constraints. We start with the stability constraint:

▶ **Fact 10.** *For each subset $U \subseteq S$, $|U| \geq (1-\alpha)|S|$, we have $C(U) \geq \frac{1}{8} \cdot C(S)$.*

Proof. We have

$$C(U) \geq \text{cost}(\text{MST}_{1\text{-}\alpha}(S)) \geq \frac{1}{2}\,\text{cost}(T_{1-\alpha}) \geq \frac{\beta'}{2}\,\text{cost}(\text{MST}(S)) \geq \frac{1}{8}C(S) \ .$$

The first step follows because an optimum path for $C(U)$ is also a spanning tree on at least $\lceil(1-\alpha)|S|\rceil$ vertices. The second step holds because of the approximation guarantee of Garg's algorithm. The third step is due to the termination condition of our procedure for finding a stable set, and the last step holds because an MST is a 2-approximation for $C(S)$. ◀

It remains to prove the size constraint. For this we require a bound on the number of iterations.

▶ **Fact 11.** *The algorithm performs at most $r_{\max} \leq \log_2(\ell\Delta) + 1$ unsuccessful iterations.*

Proof. In every unsuccessful iteration the cost of the minimum spanning tree over the set S decreases by factor $\beta' = 1/2$. The cost can be at most $\ell\Delta$ at the start, and if the cost drops below one, all remaining requests are located at a single vertex, which leads to a stable set. ◀

In every unsuccessful iteration the cardinality of the set S decreases by a $(1-\alpha)$ factor. Hence, the final cardinality is at least $k(1-\alpha)^{r_{\max}} \geq k/e$. This completes the proof of the lemma. ◀

5 Lower Bound for Memory Restricted Algorithms

In [20] Khandekar and Pandit defined a memoryless reordering buffer management algorithm as an algorithm that bases its decisions only on the content of the buffer and not on some further information that may be stored in its memory. They showed that such algorithms are severely limited by giving a lower bounds of $\Omega(k)$ on the competitive ratio. In terms of the aspect ratio their lower bound example gives $\Omega(\log\Delta/\log\log(\Delta))$.

In this section we extend their result and show that an algorithm that only bases its decision on the buffer-content and on further k bits of memory may experience a competitive ratio of $\Omega(\sqrt{\log\Delta})$. This means, if the memory used by the algorithm does not depend on the aspect ratio, the aspect ratio must appear in the competitive ratio in some form (unless, of course, the competitive ratio is a trivial bound like $O(k)$).

Since our block-oriented algorithm only needs to mark all requests that belong to the current block, it can be implemented with k bits of memory. Hence, one reason that the aspect ratio appears in our competitive ratio is the structure of the algorithm that makes it memory-restricted.

▶ **Lemma 12.** *There exists an input sequence with aspect ratio $\Delta \leq (k(k+1)2^m)^k$, for which any deterministic reordering buffer management algorithm with m bits of memory has competitive ratio $\Omega(k)$. For $m = k$ this gives a lower bound of $\Omega(\sqrt{\log\Delta})$ on the competitive ratio.*

Proof. The instance consists of a metric space over $k+1$ vertices $\{v_0, \ldots, v_k\}$, where the distance between two distinct vertices v_i and v_j is $d(v_i, v_j) = \lambda^i + \lambda^j$. The vertices can be viewed as the leaves of a star, where the vertex v_i is connected to the center of the star via an edge of length λ^i. λ will be chosen later.

Let $\mathrm{ALG}_t \in \{v_0, \ldots, v_k\}$ denote the position of the online server in the metric space after the t-th request has been served. Initially, there is a request at every vertex v_i, $i \neq 0$, and we assume contrary to the definition of our model in Section 1.2 that the online algorithm has to start at vertex v_0 (i.e., $\mathrm{ALG}_0 = v_0$). This slight change in the model does not affect our asymptotic results. The input sequence is chosen adversarily: after serving the request at ALG_t a new request at ALG_{t-1} appears. This means that whenever the online algorithm is making a decision on the next request to serve, there is a request located at every vertex v_i different from the current position of the ALG-server.

There are $k+1$ possible states of the buffer; one state for every position of the online server. In addition, the m bits of memory give rise to 2^m memory-states. The state of the algorithm is a combination of the buffer-state and the memory-state. This means in total there are $z := (k+1)2^m$ different states that the algorithm may be in. Depending on its state $\mathcal{S}$ the algorithm deterministically chooses a vertex v_{next}, and serves the request located at this vertex. Then a new request appears at its previous position, and the algorithm is in some new state $\mathcal{S}'$.

We model the servicing of our adversarial sequence σ by a deterministic algorithm, as a path on a state graph G that contains one vertex for every possible state $\mathcal{S}$, and a directed edge $(\mathcal{S}, \mathcal{S}')$ if $\mathcal{S}'$ is the successor state to state $\mathcal{S}$. We assign a weight to every edge in G as follows. If $\mathcal{S}$ corresponds to a state where the server is located at v_i and $\mathcal{S}'$ corresponds to a state with the server at position v_j, we assign a length of $d(v_i, v_j)$ to edge $(\mathcal{S}, \mathcal{S}')$. By this definition the servicing of the request sequence corresponds to a path P on the state graph and the length of this path is the cost of the online algorithm. Note that the path will actually contain a cycle C, and asymptotically the cost of the online algorithm is determined by the cost for serving the cycle.

Let $v_{i_{\max}}$ denote the vertex with largest index that corresponds to some state along the cycle, let $n_{\max}$ denote the number of states along the cycle that correspond to this position, and let n_C denote the total number of vertices along the cycle. The cost of the online algorithm for serving the cycle is at least

$$\mathrm{cost}_{\mathrm{ALG}}(C) \geq 2n_{\max}\lambda^{i_{\max}} \ ,$$

as it enters and leaves the vertex $v_{i_{\max}}$ at least $n_{\max}$ times. An optimum algorithm can serve the cycle differently. It only holds requests at location $v_{i_{\max}}$ in its buffer. All other requests are served immediately as they arrive. Then it only has to pay for the edge to $v_{i_{\max}}$ every k-th time. Hence, the optimum (average) cost for serving the cycle is at most

$$\mathrm{cost}_{\mathrm{OPT}}(C) \leq 2n_C\lambda^{i_{\max}-1} + 2n_{\max}\lambda^{i_{\max}}/k \ .$$

This gives

$$\frac{\mathrm{cost}_{\mathrm{ALG}}(C)}{\mathrm{cost}_{\mathrm{OPT}}(C)} \geq \frac{n_{\max}\lambda^{i_{\max}}}{n_C\lambda^{i_{\max}-1} + n_{\max}\lambda^{i_{\max}}/k} \geq \frac{n_{\max}}{z/\lambda + n_{\max}/k} \geq \frac{n_{\max}}{1+n_{\max}}k = \Omega(k) \ .$$

Here, we use the fact that $n_C \leq z$ (the number of states) for the second inequality, and we choose $\lambda = kz$ for the third inequality. The ratio of the costs on the cycle gives an asymptotic bound on the competitive ratio, as the cycle dominates the cost. With our choice of λ we get that the aspect ratio Δ is $\Delta \leq \lambda^k = (k(k+1)2^m)^k$. ◀

Acknowledgments. We thank Matthias Englert for making us aware that the competitive ratio of a memory restricted algorithm should depend on the aspect ratio.

References

1. Amjad Aboud. Correlation clustering with penalties and approximating the reordering buffer management problem. *Master's thesis, Computer Science Department, The Technion – Israel Institute of Technology*, 2008.
2. Anna Adamaszek, Artur Czumaj, Matthias Englert, and Harald Räcke. Almost tight bounds for reordering buffer management. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*, pages 607–616, 2011.
3. Yuichi Asahiro, Kenichi Kawahara, and Eiji Miyano. NP-hardness of the sorting buffer problem on the uniform metric. *Discrete Applied Mathematics*, 160(10-11):1453–1464, 2012. doi:10.1016/j.dam.2012.02.005.
4. Noa Avigdor-Elgrabli, Sungjin Im, Benjamin Moseley, and Yuval Rabani. On the randomized competitive ratio of reordering buffer management with non-uniform costs. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 78–90, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
5. Noa Avigdor-Elgrabli and Yuval Rabani. A constant factor approximation algorithm for reordering buffer management. In *Proceedings ofproc 24th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 973–984, 2013. doi:10.1137/1.9781611973105.70.
6. Noa Avigdor-Elgrabli and Yuval Rabani. An optimal randomized online algorithm algorithm for reordering buffer management. In *Proceedings of the 54th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1–10, 2013.
7. Noa Avigdor-Elgrabli and Yuval Rabani. An improved competitive algorithm for reordering buffer management. *ACM Trans. Algorithms*, 11(4):1–15, June 2015. doi:10.1145/2663347.
8. Siddharth Barman, Shuchi Chawla, and Seeun Umboh. A bicriteria approximation for the reordering buffer problem. In *Proceedings of the 20th Annual European Symposium on Algorithms (ESA)*, pages 157–168, 2012. doi:10.1007/978-3-642-33090-2_15.
9. Marcin Bienkowski, Martin Böhm, Lukasz Jez, Pawel Laskos-Grabowski, Jan Marcinkowski, Jirí Sgall, Aleksandra Spyra, and Pavel Veselý. Logarithmic price of buffer downscaling on line metrics. *CoRR*, abs/1610.04915, 2016.
10. Dan Blandford and Guy Blelloch. Index compression through document reordering. In *Proceedings of the Data Compression Conference*, DCC'02, pages 342–351, Washington, DC, USA, 2002. IEEE Computer Society.
11. Ho-Leung Chan, Nicole Megow, René Sitters, and Rob van Stee. A note on sorting buffers offline. *Theoretical Computer Science*, 423:11–18, 2012.
12. Matthias Englert and Harald Räcke. Reordering buffers with logarithmic diameter dependency for trees. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SIDA)*, pages 1224–1234, 2017.
13. Matthias Englert and Matthias Westermann. Reordering buffer management for non-uniform cost models. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 627–638, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
14. Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.
15. Iftah Gamzu and Danny Segev. Improved online algorithms for the sorting buffer problem on line metrics. *ACM Trans. Algorithms*, 6(1):15:1–15:14, December 2009.

16 Naveen Garg. Saving an ϵ: A 2-approximation for the k-MST problem in graphs. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 396–402, 2005.

17 Kai Gutenschwager, Sven Spiekermann, and Stefan Voß. A sequential ordering problem in automotive paint shops. *International Journal of Production Research*, 42(9):1865–1878, 2004. doi:10.1080/00207540310001646821.

18 Sungjin Im and Benjamin Moseley. New approximations for reordering buffer management. In *Proceedings of 25th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1093–1111, 2014. doi:10.1137/1.9781611973402.81.

19 Sungjin Im and Benjamin Moseley. Weighted reordering buffer improved via variants of knapsack covering inequalities. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 737–748, 2015. doi:10.1007/978-3-662-47672-7_60.

20 Rohit Khandekar and Vinayaka Pandit. Online sorting buffers on line. In *Proceedings of the 23rd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 584–595, 2006.

21 Jens Krokowski, Harald Räcke, Christian Sohler, and Matthias Westermann. Reducing state changes with a pipeline buffer. In *Proceedings of the 9th International Fall Workshop Vision, Modeling, and Visualization*, 2004.

22 Harald Räcke, Christian Sohler, and Matthias Westermann. Online scheduling for sorting buffers. In *Proceedings of the 10th Annual European Symposium on Algorithms (ESA)*, pages 820–832, 2002.

A Reasonable Algorithms

We define an algorithm for the reordering buffer management problem to be *reasonable* if at any point in time it chooses a request r from the buffer as the next request to be served, and then moves from its current location to r along a shortest path.

▶ **Lemma 13.** *Any reasonable algorithm for the reordering buffer management problem has competitive ratio* $2(h+k)$.

Proof. Suppose a reasonable online algorithm generates an output sequences $s_1, s_2, \ldots, s_\ell$. Its cost $\mathrm{ALG}_{true}(\sigma)$ is then $\sum_{i=1}^{\ell-1} d(s_i, s_{i+1})$. We define sets of consecutive requests: $X_i := \{s_i, s_{i+1}\}$. In the following we prove that at any point in time there can at most be $2(h+k)$ sets X_i that are partially scheduled by OPT. The result then follows by applying Claim 4.

Fix a time t. We order the elements within a request-pair X_j according to the order in which the elements are scheduled by OPT, and will refer to them as the first and second request, respectively. Suppose a request pair $X_j, j \leq t$ is partially scheduled by OPT at time t. This means that the second request of the pair must stay in OPT's buffer between steps t and $t+1$ because both requests have already appeared by time t. Note that this holds even for the case $j = t$, because the element s_{t+1} that is output by ALG at time $t+1$ must have appeared on or before time t. Any request is only contained in at most two pairs. Consequently, there can be at most $2h$ request pairs $X_j, j \leq t$, that are partially scheduled by OPT at time t.

Now, consider a request pair X_j, $j > t$ that is partially scheduled by OPT at time t. The first request of the pair is scheduled by OPT at time t or before, but ALG schedules both requests at time $t+1$ or later. Hence, the first request of the pair must be stored by ALG between steps t and $t+1$. This means we can have at most $2k$ request pairs $X_j, j > t$

that are partially scheduled by OPT at time t. In total we get at most $2(h+k)$ partially scheduled pairs. Applying Claim 4 gives

$$\mathrm{OPT}(\sigma) \geq \frac{1}{2(h+k)} \sum_i C(X_i) \ ,$$

and, hence, $\mathrm{ALG}_{true}(\sigma) \leq 2(h+k) \cdot \mathrm{OPT}(\sigma)$, as desired. ◀