View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Warwick Research Archives Portal Repository

WARWICK
THE UNIVERSITY OF WARWICK

Original citation:
Englert, Matthias and Räcke, Harald (2017) Reordering buffers with logarithmic diameter dependency for trees. In: 28rd ACM-SIAM Symposium on Discrete Algorithms, Barcelona, Spain, 16-19 Jan 2017. Published in: SODA '17 Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms 1224-1234.

Permanent WRAP URL:
http://wrap.warwick.ac.uk/84475

Copyright and reuse:
The Warwick Research Archive Portal (WRAP) makes this work of researchers of the University of Warwick available open access under the following conditions. Copyright © and all moral rights to the version of the paper presented here belong to the individual author(s) and/or other copyright owners. To the extent reasonable and practicable the material made available in WRAP has been checked for eligibility before being made available.

Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge. Provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way.

Publisher's statement:
First Published SODA '17 Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms published by the Society for Industrial and Applied Mathematics (SIAM). Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

A note on versions:
The version presented in WRAP is the published version or, version of record, and may be cited as it appears here.

For more information, please contact the WRAP Team at: wrap@warwick.ac.uk

warwick.ac.uk/lib-publications

Downloaded 06/20/17 to 137.205.202.240. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Reordering Buffers with Logarithmic Diameter Dependency for Trees

Matthias Englert* Harald Räcke†

Abstract

In the reordering buffer problem a sequence of items located in a metric space arrive online, and have to be processed by a single server moving within the metric space. At any point in time, the first k still unprocessed items from the sequence are available for processing and the server has to select one of these items and process it by visiting its location. The goal is to process all items while minimizing the total distance the server moves.

Englert, Räcke, Westermann (STOC'07) gave a deterministic $O(D \cdot \log k)$-competitive online algorithm for weighted tree metrics with hop-diameter D. We improve the analysis of this algorithm and significantly improve the dependency on D. Specifically, we show that the algorithm is in fact $O(\log D + \log k)$-competitive. Our analysis is quite robust. Even when an optimal algorithm, to which we compare the online algorithm, is allowed to choose between the first $h > k$ unprocessed items, the online algorithm is still $O(h \cdot (\log D + \log h)/k)$-competitive. For $h = (1 + \varepsilon) \cdot k$, with constant $\varepsilon > 0$, this is optimal.

Our results also imply better competitive ratio for general metric spaces, improving the randomized $O(\log n \cdot \log^2 k)$ result for n-point metric spaces from STOC'07 to $O(\log n \cdot \log k)$.

1 Introduction

In the reordering buffer problem, a sequence of items located in a metric space arrive online, and have to be processed by a single server moving within the metric space. There exists a request buffer that can hold up to k requests and gives the server some flexibility in choosing the next request to serve. More precisely, at any point in time, the first k still unprocessed items from the sequence are available for processing and the server has to select one of these items and process it by visiting its location. The goal is to process all items while minimizing the total distance that the server is moving.

This model has been introduced by Räcke et al. [26] for modeling the context switching cost that occurs in applications in many different areas ranging from production engineering through computer graphics to information retrieval [6, 14, 21, 25]. In this paper we focus on the online version of the problem in which the server does not see future requests in the input stream but has to make its decision online only depending on past requests and on the items currently in the 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*.

Englert, Räcke, Westermann [16] gave a deterministic $O(D \cdot \log k)$-competitive online algorithm for weighted tree metrics with hop-diameter D. In this paper we improve the analysis of this algorithm and significantly improve the dependency on D. We show that the algorithm is in fact $O(\log k + \log D)$-competitive. Our analysis is quite robust, and also extends to the case when we compare the performance of the online algorithm to the performance of an optimum algorithm with *larger* buffer size $h \geq k$. We prove a competitive ratio of $O(\frac{h}{k}(\log h + \log D))$, i. e., the competitive ratio increases gracefully as h increases.

Theorem 1.1. *On tree metrics with hop-diameter D,* PAY*, with a buffer of size k, is $O(\frac{h}{k}(\log h + \log D))$-competitive against an optimal offline algorithm with a buffer of size h.*

For $h = (1 + \varepsilon) \cdot k$, with constant $\varepsilon > 0$, our analysis is optimal. Therefore, any further improvement for the case $h = k$ must involve a proof that does not share the robustness property of the proof presented here. This follows from two results: Firstly, Aboud [1] has shown that on a uniform metric (a star) the gap between two *optimal* algorithms (one with buffer-size k, the other with buffer-size $h = 4k$) can be as large as $\Omega(\log k)$. In Appendix A, we give a slightly different proof, inspired by the input sequence generation in [17], that shows that the same bound can in fact be shown for any h that is any constant factor larger than k. Hence, any online

*Department of Computer Science and Centre for Discrete Mathematics and its Applications (DIMAP), University of Warwick. M.Englert@warwick.ac.uk.

†Department of Informatics, Technical University Munich. raecke@in.tum.de

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 06/20/17 to 137.205.202.240. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

algorithm (deterministic or randomized) also must have competitive ratio at least $\Omega(\log k)$ when compared to an optimal algorithm with a buffer that is larger by a constant factor. Secondly, Bieńkowski et al. [13] gives an instance on a line with D equidistant points, on which the gap between two optimal algorithms (buffer sizes $h = (1+\varepsilon)\cdot k$ and k, respectively) is $\Omega(\log D)$ (as long as $k \geq \log D$). Again, any online algorithm must therefore have competitive ratio $\Omega(\log D)$ when compared to an optimal algorithm with a buffer that is larger by a constant factor.

We then show that for the special case of hierarchically well separated trees (HSTs) the competitive ratio improves to $O(\frac{h}{k}\cdot\log h)$, or simply $O(\log k)$ if $h = k$. Note that this bound is independent of the diameter of the tree. Combining this result with the results on probabilistically approximating arbitrary metrics by tree metrics [11, 12, 19], we obtain a randomized scheduling strategy for general metric spaces that achieves a competitive ratio of $O(\log k\cdot\log n)$ in expectation against an oblivious adversary. Here n denotes the number of distinct points in the metric space. This improves the previous best result for general metric spaces [16] by a $\Theta(\log k)$-factor. If the metric space is not known in advance one can e.g. use Bartal [11] to construct a tree embedding as the points in the metric space appear. This gives an overall guarantee of $O(\log\Delta\log n\log k)$, where Δ denotes the aspect ratio of the metric space.

1.1 Further Work.

Most previous work on the reording buffer problem focuses on the online problem in uniform star networks. Räcke et al. [26] introduced the problem and developed a deterministic algorithm with competitive ratio $O(\log^2 k)$. This was first improved to $O(\log k)$ [18], and later to $O(\log k/\log\log k)$ [6], before Adamaszek et al. [2] obtained a bound of $O(\sqrt{\log k})$, which is close to optimal due to a lower bound of $\Omega(\sqrt{\log k/\log\log k})$ shown in the same paper. Avigdor-Elgrabli and Rabani [8] showed that randomization can give an exponential improvement to the competitive ratio by developing an algorithm with competitive ratio $O(\log\log k)$. This is optimal due to a corresponding lower bound proved by Adamaszek et al. [2].

Khandekar and Pandit [24] analyze the reordering buffer problem for D uniformly-spaced points on a line with the motivation that this scenario models the disc scheduling problem. They present a randomized algorithm with a competitive ratio of $O(\log^2 D)$ in expectation against an oblivious adversary. Gamzu and Segev [20] improve this by presenting a deterministic $\Theta(\log D)$-competitive strategy that also works if the points are unevenly spaced. In addition, they give, for the line metric, a lower bound of ≈ 2.154 on the competitive ratio of any deterministic algorithm.

There has been a large amount of further work on hardness and approximation of the offline problem as well as other variants of the online problem for different metrics. See, for example, [3], [4], [5], [7], [9], [10], [15], [22], and [23].

2 The Algorithm

We study the algorithm PAY which was introduced in [16]. Initially, the reordering buffer is filled with the first k items from the input sequence. After that, PAY alternates between a selection and a processing phase.

The selection phase identifies a set of items to be visited by the server. It works as follows. Each item stored in the buffer generates "payment" at a unit rate. This payment can be placed on edges. An edge e is *fully paid*, if the amount of payment $\mathsf{pay}(e)$ on the edge is equal to the length $\ell(e)$ of the edge. The payment generated by an item in a time interval $[t, t+dt)$ is placed on the first edge on the path from the item to PAY's server that is not fully paid. This continuous process of generating and placing payment is stopped once PAY's server is connected, through fully paid edges, to at least one item. At this point, PAY's server will be located in a connected subgraph induced by fully paid edges.

In the processing, phase the items located in this connected subgraph are processed. This is done as follows. Let p denote the position of the PAY-server at the beginning of the processing phase, and let p' denote a position in the connected subgraph that is furthest from p. The PAY-server visits all items in the connected subgraph and chooses p' as its new location. In addition, the payment on all edges in the subgraph (but not on other edges) is cleared, i.e., $\mathsf{pay}(e) := 0$ for these edges. Finally, the now empty slots in the buffer are filled up again by the next items in the input sequence. If the buffer contains k items again, we go back to carry out a new selection phase. Otherwise, the algorithms stops (clearing the remaining items in the buffer in an optimal way).

We only use the notion of continuous time for the selection phase. That means that if the i-th selection phase last from time t to time t', then the $(i+1)$-th selection phase starts at time t'. That is, the processing phase in between happens instantaneous and time does not progress during it. We also note that while we describe the selection phase as a continuous process, it can be easily discretized and implemented efficiently.

3 Proof of Theorem 1.1

To compare PAY to an optimal offline algorithm, we fix such an optimal offline algorithm OPT and we imagine

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 06/20/17 to 137.205.202.240. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

running PAY and OPT in parallel on the same input. PAY has a buffer that can store up to k and OPT has a buffer that can store up to $h \geq k$ items.

More precisely, initially OPT's and PAY's buffer contain the first h and k items of the input sequence, respectively. Then PAY performs a selection phase followed by a processing phase which processes, say, x items. After that, OPT will also process x items and fill up its buffer with the next x items in the input sequence.

We start with a basic observation. Let cost_i denote the sum of lengths of edges in the connected subgraph that contains PAY's server in the i-th processing phase.

Observation 1. *The total cost of PAY, i.e., the total distance PAY's server travels, is at most* $2 \cdot \sum_i \mathsf{cost}_i$.

Ideally, we would like to perform the following analysis: Fix an edge $e = \{u, v\}$. This edge splits the tree into two subtrees T_u and T_v containing u and v respectively. Imagine OPT's server is located in T_v. How often can e contribute to some cost_i, i.e., be part of the connected subgraph, before OPT traverses e as well? Bounding this would give a bound on $\sum_i \mathsf{cost}_i$ in terms of the optimal cost and, due to the observation above, also give us a bound on PAY's overall cost.

Unfortunately, this approach fails for multiple reasons. The first problem is the following: When a connected subraph is processed, PAY's server moves from some initial position p to a new position p'. If, for each processing phase, p and p' are always both contained in T_u but the connected subgraph always contains the edge e, this tells us very little about when OPT needs to visit T_u. (We do know that each time a new item is processed in T_u and therefore after at most $h + k$ such phases OPT will have h items located in T_u and none in T_v forcing it to use edge e. But we are aiming for a much better bound than $O(h + k)$.)

As we will see in our remaining analysis, the situation is much better if the starting point p lies in T_v instead of T_u. In other words, the edges in the connected subgraph of some processing phase that are problematic are the ones that lie on the path from the initial position p of the PAY server to the position q of the OPT server. Therefore, we introduce a new cost $\mathsf{reduced\text{-}cost}_i$ of the i-th processing phase which is equal to cost_i minus the total length of edges in the connected subgraph that lie on the path from p to q. This way we will be able to ignore the problematic edge costs, because the following claim implies that it is sufficient to find a good bound on $\sum_i \mathsf{reduced\text{-}cost}_i$.

Claim 1. $\sum_i \mathsf{cost}_i \leq 2 \cdot \mathrm{OPT} + 3 \cdot \sum_i \mathsf{reduced\text{-}cost}_i$.

Proof. Suppose we are at the start of some processing phase i. PAY's server is at some location p which lies in a connected subgraph induced by fully paid edges. As before, for simplicity we will just call this *the* connected subgraph from now on. After processing all items in this connected subgraph, PAY's server is located at a point p' which is furthest away from p in the connected subgraph. Let q be the last vertex in the connected subgraph on the path from p to OPT's server. Note that $\mathsf{reduced\text{-}cost}_i$ is equal to cost_i minus the length of the path from p to q.

Let x be the point at which the paths from p to p' and from p to q split. Let $\mathsf{path\text{-}cost}^1_i$ denote the length of the path from p to x and let $\mathsf{path\text{-}cost}^2_i$ denote the length of the path from x to q. We have $\mathsf{reduced\text{-}cost}_i = \mathsf{cost}_i - \mathsf{path\text{-}cost}^1_i - \mathsf{path\text{-}cost}^2_i$ and will now argue that $\sum_i \mathsf{path\text{-}cost}^1_i \leq 2 \cdot \mathrm{OPT} + \sum_i \mathsf{reduced\text{-}cost}_i$ and $\mathsf{path\text{-}cost}^2_i \leq \mathsf{reduced\text{-}cost}_i$. Note that this implies the claim since

$$\begin{aligned}\sum_i \mathsf{cost}_i &= \sum_i (\mathsf{reduced\text{-}cost}_i + \mathsf{path\text{-}cost}^1_i + \mathsf{path\text{-}cost}^2_i) \\ &\leq \sum_i \mathsf{reduced\text{-}cost}_i + 2 \cdot \mathrm{OPT} \\ &\quad + \sum_i \mathsf{reduced\text{-}cost}_i + \sum_i \mathsf{reduced\text{-}cost}_i \ .\end{aligned}$$

We start with the easier second part. The length of the path from x to q is at most as long as the length of the path from x to p', because otherwise p' was not a furthest point from p in the connected subgraph. Therefore, it must also be the case that $\mathsf{path\text{-}cost}^2_i \leq \mathsf{reduced\text{-}cost}_i$.

Now for the more involved first part. Consider any edge $e = \{u, v\}$ on the path from p to x. The removal of this edge would result in two subtrees T_u and T_v that are attached to vertex u and v, respectively. Now, w.l.o.g. p is located in T_u. But then p', q, and OPT's server must be located in T_v. Suppose the next processing phase in which edge e is fully paid is some processing phase $j > i$. Either OPT has moved its server over edge e in between processing phase i and j, or, in processing phase j, e cannot lie on the path from PAY's to OPT's server. Note that in the latter case e will contribute to $\mathsf{reduced\text{-}cost}_j$.

It may happen that no next processing phase j exists in which e is fully paid, because we have reached the end of the input sequence. However, this can only happen once for each edge and, furthermore, only for edges that have to be traversed by OPT at least once (because the input sequence must contain at least one item in T_u and one in T_v). Therefore the total cost neglected in this way is bounded by OPT. Altogether, we get $\sum_i \mathsf{path\text{-}cost}^1_i \leq \mathrm{OPT} + \mathrm{OPT} + \sum_i \mathsf{reduced\text{-}cost}_i$. □

If $\sum_i \mathsf{reduced\text{-}cost}_i \leq \mathrm{OPT}$, the claim, combined with Observation 1, already implies that PAY is 10-

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 06/20/17 to 137.205.202.240. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

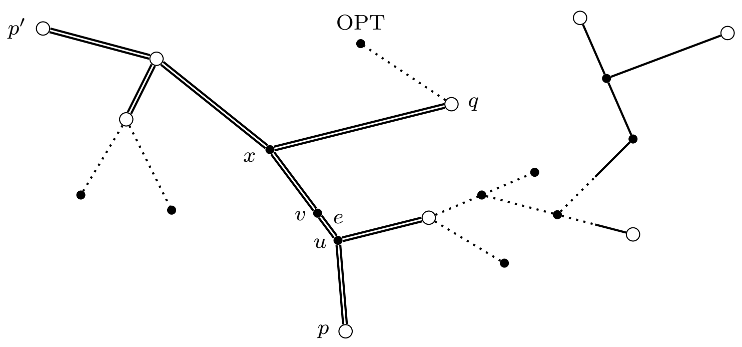

Figure 1: White nodes are nodes at which items are located. The solid part of the edges indicates payment. The double edges are fully paid and are part of the connected sub-graph. In this processing phase, the server would start at p and end at p'. Edge e would not contribute to reduced-cost since it lies between p and the OPT-server.

competitive. Therefore, we assume $\sum_i \mathsf{reduced\text{-}cost}_i >$ OPT from now on. In this case, the claim implies

$$(3.1) \qquad \sum_i \mathsf{cost}_i \le 5 \cdot \sum_i \mathsf{reduced\text{-}cost}_i \,.$$

In the following, we will work with a more wasteful inequality instead.

$$(3.2) \qquad \sum_i \mathsf{cost}_i \le 10 \cdot \sum_i \mathsf{reduced\text{-}cost}_i \,.$$

While this is not necessary for our analysis for general trees, it will be helpful in the analysis for hierarchically well separated trees in Section 4.

Ideally we would now like to find an upper bound on the number of times an edge e contributes to some $\mathsf{reduced\text{-}cost}_i$ term in relation to the number of times OPT traverses e. However, this cannot be done directly either and requires a further type of amortization. To this end, we introduce the concept of *discount*. In our algorithm, each item generates payment and places it on edges. In addition to this, we now let items also generate discount which is placed on edges. The discount generation is only used in the analysis and not in the algorithm itself. It therefore can be based on the entire input sequence and OPT's actions.

Recall that each item in PAY's buffer generates payment dt on the first non-fully paid edge towards the PAY server. Suppose item I is in the PAY buffer for d_I time units after having arrived at time a_I. Further suppose item I generates a total payment of $p(I, e)$ on edge e. We define $\alpha(I, e) := p(I, e)/d_I$. That is $\alpha(I, e)$ indicates what fraction of the total payment generated by I, is generated on edge e.

We now let each item stored either in OPT's or PAY's buffer (but not in both) generate discount on each edge e between the item's location and the PAY-server. In each time interval of length dt an item places

$$k \cdot \frac{\alpha(I, e) + 1/D}{160h} \cdot dt$$

discount on such an edge. Let $\mathcal{I}_e^{\mathrm{OPT}}(i)$ be the set of items stored in OPT's buffer that generate discount on edge e during selection phase i and let $\mathcal{I}_e^{\mathrm{PAY}}(i)$ be the set of items stored in PAY's buffer that generate discount on edge e during selection phase i.

Finally, we need one more type of discount. Let $q(I, e)$ be the payment generated by item I on edge e *after* time $a_I + 159d_I/160$ (note that in total over all edges this is only $d_I/160$). Right before being processed by PAY, item I allocates an *extra discount* of $4q(I, e)$ to edge e.

Claim 2. *The total amount of discount placed by all items is at most 5% of the total payment $\sum_I d_I = \sum_I \sum_e p(I, e) = \sum_i \mathsf{cost}_i$ generated by all items.*

Proof. The total discount generated by an item I during a time interval dt is at most

$$\sum_e k \cdot \frac{\alpha(I, e) + 1/D}{160h} \cdot dt = \sum_e k \cdot \frac{p(I, e)/d_I + 1/D}{160h} \cdot dt$$
$$= \frac{k}{160h} + \sum_e \frac{k}{160hD} \cdot dt \le \frac{k}{80h} \cdot dt \ .$$

There are at most $h + k \le 2h$ items stored in OPT's or PAY's buffer at any point in time. Combined, they generate discount of at most $k/40 \cdot dt$, while k items in PAY's buffer generate payment of $k \cdot dt$ at the same time.

Finally, each item I places an additional amount of extra discount $4\sum_e q(I, e) = 4(d_I - 159d_I/160) = d_I/40$. But the same item also generates payment of $\sum_e p(I, e) = d_I$ over its lifetime in PAY's buffer. ☐

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 06/20/17 to 137.205.202.240. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Note that due to Equation 3.2, we can also conclude that the total discount placed by all items is at most $\sum_i \textsf{reduced-cost}_i/2$.

Just like payment, discount is removed from edges in the connected subgraph after the processing phase. We now further limit the number of edges that we consider to contribute cost to processing phase i. Specifically, consider the connected subgraph of processing phase i. Let $\textsf{discounted-cost}_i$ denote the total length of edges in the connected subgraph which are neither

1. on the path between the initial position of the PAY-server and the position of the OPT-server nor
2. carry a discount of $\ell(e)$ or more at the beginning of the processing phase.

Since the total discount placed on edges is at most $\sum_i \textsf{reduced-cost}_i/2$, we can make the following observation.

Observation 2.

$$\begin{aligned}\sum_i \textsf{discounted-cost}_i &\\ &\geq \sum_i \textsf{reduced-cost}_i - \frac{1}{2}\sum_i \textsf{reduced-cost}_i\\ &= \frac{1}{2}\sum_i \textsf{reduced-cost}_i \ .\end{aligned}$$

Fix an edge $e = \{u, v\}$ and let T_u and T_v be the two subtrees obtained by deleting e. Assume w.l.o.g. that OPT's server is in T_v. We want to analyze for how many i, the edge e can contribute to $\textsf{discounted-cost}_i$ before OPT's server traverses e to enter T_u. In the following we will show that the number of times this can happen is bounded by $O(\frac{h}{k} \cdot (\log h + \log D))$, which will conclude the proof of the theorem. Let

$$\text{rate}_e^{\text{OPT}}(i) = \sum_{I \in \mathcal{I}_e^{\text{OPT}}(i)} k \cdot \frac{\alpha(I, e) + 1/D}{160h}$$

be the rate of discount generation on e during selection phase i (which is followed by processing phase i) by items in OPT's buffer. Note that if selection phase i lasts for time T, items in OPT's buffer will generate a total discount of $T \cdot \text{rate}_e^{\text{OPT}}(i)$ on e. Similarly let $\text{rate}_e^{\text{PAY}}(i) = \sum_{I \in \mathcal{I}_e^{\text{PAY}}(i)} k \cdot (\alpha(I, e) + 1/D)/(160h)$ be the rate of discount generation on e during selection phase i by items in PAY's buffer.

Lemma 3.1. *Suppose the i-th processing phase is one for which edge $e = \{u, v\}$ contributes to* $\textsf{discounted-cost}_i$ *and let processing phase $i' < i$ be the last processing phase before i in which edge e was contained in the connected subgraph to be processed. If such an i' does not exist, let i' be the first processing phase. If OPT does not traverse edge e between processing phase i' and processing phase $i + 1$, either*

$$\text{rate}_e^{\text{OPT}}(i + 1) > \Big(1 + \frac{k}{102400h}\Big) \cdot \text{rate}_e^{\text{OPT}}(i)$$

or

$$\text{rate}_e^{\text{PAY}}(i + 1) < \Big(1 - \frac{k}{102400h}\Big) \cdot \text{rate}_e^{\text{PAY}}(i) \ .$$

Proof. As before, let T_v and T_u be the two subtrees obtained by deleting edge e. When the i-th processing phase starts, PAY and OPT must be contained in the same subtree since e contributes to $\textsf{discounted-cost}_i$ and therefore cannot lie on the path between the PAY-server and the OPT-server. Assume w.l.o.g. that both servers are located in T_v. In fact, they must also be located in T_v during all selection phases $i' + 1, \ldots, i$. For PAY this follows from the fact that the last processing phase in which e was contained in the connected subgraph was i'. For OPT this follows from the explicit assumption that OPT does not traverse edge e between processing phase i' and processing phase $i + 1$.

Let $\mathcal{P}$ be the set of items that generated payment on e in at least one of the selection phases $i' + 1, \ldots, i$. We observe that all items in $\mathcal{P}$ are located in T_u since the PAY server is located in T_v. Another important observation is that all items in $\mathcal{P}$ will be processed by PAY in processing phase i. These items generate payment on e and e is in the connected subgraph of processing phase i. Thus, these items are also part of the connected subgraph of processing phase i. Now we partition $\mathcal{P} = \mathcal{P}^{\text{OPT}} \cup \mathcal{P}^{\text{PAY}}$ into the subset $\mathcal{P}^{\text{OPT}} \subseteq \mathcal{P}$ of items that are also stored in OPT's buffer and the subset $\mathcal{P}^{\text{PAY}} \subseteq \mathcal{P}$ of items that are exclusively stored in PAY's buffer at the beginning of processing phase i.

Each selection phase of the algorithm has a duration that indicates how much payment each item in PAY's buffer generates during the phase. Let T be the total combined duration of selection phases $i' + 1, \ldots, i$.

Let $O \subseteq \mathcal{P}$ be the subset of items I for which $d_I > 160 \cdot T$, i.e., items that have been in the buffer for a significantly longer duration than just these selection phases. For any such item we have $p(I, e) = q(I, e)$. This is because $I \in \mathcal{P}$ only generates payment on e during selection phases $i' + 1, \ldots, i$ and these phases make up at most 1/160-th of the total lifetime of the item I.

Claim 3. *The following three inequalities hold.*

1. $\sum_{I \in O} p(I, e) < \ell(e)/4$.
2. $\text{rate}_e^{\text{OPT}}(i) < \ell(e)/T$.

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 06/20/17 to 137.205.202.240. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

3. $\text{rate}_e^{\text{PAY}}(i) < \ell(e)/T$.

Proof. All three statements follow from the fact that edge e must have received less than $\ell(e)$ discount. Specifically:

1. If $\sum_{I\in O} p(I,e) = \sum_{I\in O} q(I,e) \geq \ell(e)/4$, we get a contradiction. All items in $I \in O$ allocate extra discount $4 \cdot q(I,e)$ to edge e. Therefore the total amount of discount on e would be at least $\ell(e)$. However, in that case e would not contribute to $\mathsf{discounted\text{-}cost}_i$ as required by the lemma.

2. If $\text{rate}_e^{\text{OPT}}(i) \geq \ell(e)/T$, the items in $\mathcal{I}_e^{\text{OPT}}(i)$ generate at least $\ell(e)$ discount on e during selection phases $i'+1, \ldots, i$. Therefore e would not contribute to $\mathsf{discounted\text{-}cost}_i$.

3. If $\text{rate}_e^{\text{PAY}}(i) \geq \ell(e)/T$, the items in $\mathcal{I}_e^{\text{PAY}}(i)$ generate at least $\ell(e)$ discount on e during selection phases $i'+1, \ldots, i$. Therefore e would not contribute to $\mathsf{discounted\text{-}cost}_i$. □

Observe that $\ell(e) = \sum_{I\in\mathcal{P}} p(I,e) = \sum_{I\in\mathcal{P}^{\text{OPT}}} p(I,e) + \sum_{I\in\mathcal{P}^{\text{PAY}}} p(I,e)$. Now, we distinguish two cases depending on whether the items in $\mathcal{P}^{\text{PAY}}$ or $\mathcal{P}^{\text{OPT}}$ give the larger contribution.

- In the case that $\sum_{I\in\mathcal{P}^{\text{OPT}}} p(I,e) \geq \ell(e)/2$, we have $\sum_{I\in\mathcal{P}^{\text{OPT}}\setminus O} p(I,e) \geq \ell(e)/4$. We observe that

$$\begin{aligned}\sum_{I\in\mathcal{P}^{\text{OPT}}} \alpha(I,e) &\geq \sum_{I\in\mathcal{P}^{\text{OPT}}\setminus O} \alpha(I,e)\\ &= \sum_{I\in\mathcal{P}^{\text{OPT}}\setminus O} \frac{p(I,e)}{d_I}\\ &\geq \sum_{I\in\mathcal{P}^{\text{OPT}}\setminus O} \frac{p(I,e)}{160T} \geq \frac{\ell(e)}{640T} .\end{aligned}$$

  Because OPT does not visit subtree T_u and because PAY processes all items in $\mathcal{P} \supseteq \mathcal{P}^{\text{OPT}}$, items in $\mathcal{P}^{\text{OPT}}$ are exclusively stored in OPT's buffer during selection phase $i+1$. Therefore, $\mathcal{I}_e^{\text{OPT}}(i+1) = \mathcal{I}_e^{\text{OPT}}(i) \cup \mathcal{P}^{\text{OPT}}$. Hence

$$\begin{aligned}\text{rate}_e^{\text{OPT}}(i+1) &\geq \text{rate}_e^{\text{OPT}}(i) + \frac{k}{160h}\sum_{I\in\mathcal{P}^{\text{OPT}}} \alpha(I,e)\\ &\geq \text{rate}_e^{\text{OPT}}(i) + \frac{k}{160h}\cdot\frac{\ell(e)}{640T}\\ &> \text{rate}_e^{\text{OPT}}(i) + \frac{k}{160h}\cdot\frac{\text{rate}_e^{\text{OPT}}(i)}{640}\\ &= \text{rate}_e^{\text{OPT}}(i)\cdot\Big(1+\frac{k}{102400h}\Big) .\end{aligned}$$

- If $\sum_{I\in\mathcal{P}^{\text{PAY}}} p(I,e) \geq \ell(e)/2$, then by arguments symmetric to the ones in the first case it follows that

$$\text{rate}_e^{\text{PAY}}(i+1) < \text{rate}_e^{\text{PAY}}(i)\cdot\Big(1-\frac{k}{102400h}\Big) .$$

This concludes the proof of the lemma. □

In addition to the previous lemma, we observe that for any selection phase i, we have $\text{rate}_e^{\text{PAY}}(i+1) \leq \text{rate}_e^{\text{PAY}}(i)$ and $\text{rate}_e^{\text{OPT}}(i+1) \geq \text{rate}_e^{\text{OPT}}(i)$ unless OPT visits subtree T_u by traversing e after processing phase i. This holds because the set of items in T_u that are exclusively stored in PAY's buffer cannot increase and the set of items in T_u that are exclusively stored in OPT's buffer cannot decrease unless OPT visits T_u.

Lemma 3.1 tells us that as long as OPT does not traverse edge e, $\text{rate}_e^{\text{OPT}}(i)$ increases by some factor or $\text{rate}_e^{\text{PAY}}(i)$ decreases by some factor if e contributes to $\mathsf{discounted\text{-}cost}_i$. Combined with the following lower and upper bounds on $\text{rate}_e^{\text{OPT}}(i)$ and $\text{rate}_e^{\text{PAY}}(i)$, this will conclude the proof of the theorem.

Claim 4. *For any edge e and selection phase i,*

- *$\text{rate}_e^{\text{OPT}}(i)$ and $\text{rate}_e^{\text{PAY}}(i)$ are upper bounded by $k/80$ and*
- *$\text{rate}_e^{\text{OPT}}(i)$ and $\text{rate}_e^{\text{PAY}}(i)$ are either 0 or at least $k/(160hD)$.*

Proof. The following equation gives the upper bound on $\text{rate}_e^{\text{OPT}}(i)$.

$$\begin{aligned}\text{rate}_e^{\text{OPT}}(i) &= \sum_{I\in\mathcal{I}_e^{\text{OPT}}(i)} k\cdot\frac{\alpha(I,e)+1/D}{160h}\\ &\leq \sum_{I\in\mathcal{I}_e^{\text{OPT}}(i)} k\cdot\frac{1+1/D}{160h}\\ &\leq k\cdot\frac{1+1/D}{160} \leq \frac{k}{80} .\end{aligned}$$

The upper bound on $\text{rate}_e^{\text{PAY}}(i)$ follows by the same reasoning.

For the second statement observe that $\text{rate}_e^{\text{OPT}}(i) = 0$ if $\mathcal{I}_e^{\text{OPT}}(i) = \emptyset$. However, if $\mathcal{I}_e^{\text{OPT}}(i) \neq \emptyset$, then

$$\text{rate}_e^{\text{OPT}}(i) = \sum_{I\in\mathcal{I}_e^{\text{OPT}}(i)} k\cdot\frac{\alpha(I,e)+1/D}{160h} \geq \frac{k}{160hD} .$$

Again, the same reasoning provides the required bound on $\text{rate}_e^{\text{PAY}}(i)$. □

Claim 5. *Let $i_{\text{start}}, \ldots, i_{\text{end}}$ denote a sequence of consecutive phases during which* OPT *does not traverse*

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 06/20/17 to 137.205.202.240. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

edge e. Then there are at most $O(\frac{h}{k}(\log h + \log D))$ many $i \in \{i_{\text{start}}, \dots, i_{\text{end}}\}$ for which e contributes to $\mathsf{discounted\text{-}cost}_i$.

Proof. Whenever e contributes, either $\text{rate}_e^{\text{OPT}}(i)$ increases by a $(1 + \frac{k}{102400h})$-factor or $\text{rate}_e^{\text{PAY}}(i)$ decreases by a $(1 - \frac{k}{102400h})$-factor. Because of the upper and lower bounds on the rates from Claim 4, this can happen at most

$$\frac{\log(\frac{k/80}{k/(160hD)})}{\log(1 + \frac{k}{102400h})} = O\Big(\frac{h}{k}(\log h + \log D)\Big)$$

times for $\text{rate}_e^{\text{OPT}}(i)$, and at most

$$\frac{\log(\frac{k/(160hD)}{k/80})}{\log(1 - \frac{k}{102400h})} = O\Big(\frac{h}{k}(\log h + \log D)\Big)$$

times for $\text{rate}_e^{\text{PAY}}(i)$. $\square$

From the above claim we get that $\sum_i \mathsf{discounted\text{-}cost}_i \leq O(\frac{h}{k}(\log h + \log D)) \cdot \text{OPT}$. Combining this with Observation 2 and Equation 3.2 concludes the proof of Theorem 1.1.

4 Hierarchically Well Separated Trees

Our improved analysis for trees also implies an improved result for hierarchically well separated trees (HSTs). A 2-HST is a rooted tree such that

- all leaf vertices are on the same level, i.e., have the same hop-distance from the root,
- all edges on the same level have the same length, and
- the length of an edge connecting a level i vertex to a level $i+1$ vertex is half the length of an edge connecting a level $i-1$ vertex to a level i vertex.

We further assume that items can only be located at leaf vertices.

In [16], it is shown that PAY is $O(D \cdot \log k)$-competitive for trees. For 2-HSTs this bound is then improved to $O(\log^2 k)$.[1] In the same way, we can use our result from the previous section to obtain an upper bound of $O(\log k)$ on the competitive ratio of PAY for 2-HSTs which improves the previous $O(\log^2 k)$ bound. Given the analysis of the previous section, this follows from the same techniques and arguments that were used in [16]. For completeness, we include the arguments in the following. Since we are proving asymptotic statements, we assume throughout that k is sufficiently large.

Since the HST is a rooted tree, the connected subgraph of a processing phase i contains a unique vertex that is on the lowest level (i.e., closest to the root) among all vertices contained in the connected subgraph. Let this vertex be y. Again, let p denote the vertex at which the PAY-server starts in processing phase i and let p' denote the position where it stops. Remember that p and p' must both be leaves. Also observe that y lies on the path between p and p'. The first modification to the proof in the previous section is that we do not allow edges on the path between p and y to contribute to $\mathsf{reduced\text{-}cost}_i$ and $\mathsf{discounted\text{-}cost}_i$ anymore.

Specifically, consider the connected subgraph of processing phase i. Let $\mathsf{reduced\text{-}cost}_i^{\text{HST}}$ denote the total length of edges in the connected subgraph which are neither

1. on the path between p and y nor
2. on the path between p and the position of the OPT-server.

As before, let x denote the vertex at which the paths from p to p' and from p to the location of the OPT-server split. If x lies on the path between y and p', then any edge between p and y was already excluded from contributing to $\mathsf{reduced\text{-}cost}_i$. So now consider the case that x lies on the path between p and y. Then $\mathsf{reduced\text{-}cost}_i^{\text{HST}}$ must be at least as large as the total length of edges on the path between p and y, since all edges on the equally long path from y to p' contribute to $\mathsf{reduced\text{-}cost}_i^{\text{HST}}$. Therefore, overall $\mathsf{reduced\text{-}cost}_i^{\text{HST}} \geq \mathsf{reduced\text{-}cost}_i - \mathsf{reduced\text{-}cost}_i^{\text{HST}}$ and hence

$$10 \cdot \mathsf{reduced\text{-}cost}_i^{\text{HST}} \geq 5 \cdot \mathsf{reduced\text{-}cost}_i \geq \mathsf{cost}_i,$$

where the last inequality was given in Equation 3.1. Thus, we get the corresponding inequality to Equation 3.2.

The second modification to the proof of Theorem 1.1 is a change of the rate at which items generate discount. Previously, each item stored either in OPT's or PAY's buffer (but not in both) generated discount on each edge e between the item's location and the PAY-server. In each time interval of length dt an item placed

$$k \cdot \frac{\alpha(I, e) + 1/D}{160h} \cdot dt$$

discount on such an edge. We change the $1/D$ term in this expression depending on whether e is one of the longer or one of the shorter edges on which I might

[1] There are other versions of HSTs but they all can be suitably approximated by the 2-HSTs defined here. The restriction for items to arrive at leaf vertices can be removed using the same techniques. See [16] for more details.

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 06/20/17 to 137.205.202.240. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

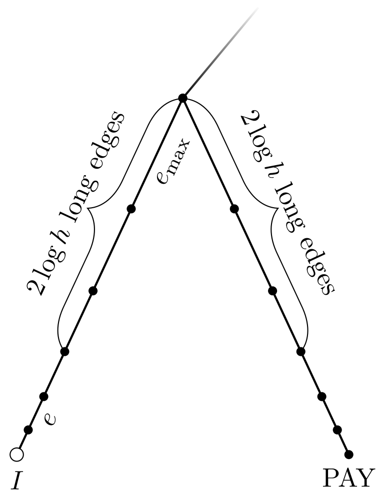

Figure 2: The part of an HST that is the path between an item I and the PAY-server. e is a short edge. $e_{\max}$ is the longest edge on the path that does not lie on the path from the PAY-server to the root of the tree.

generate discount. Specifically, for a selection phase i consider the edges on the path between the location of the item and the PAY-server. We call the longest $4\log h$ edges on this path *long edges* for selection phase i and item I and the remaining edges *short edges.* In each time interval of length dt during the selection phase, the item now generates discount

$$k \cdot \frac{\alpha(I,e) + 1/(8\log h)}{160h} \cdot dt$$

on long edges e. Let $e_{\max}$ be the longest edge on the path. Then, in each time interval of length dt during the selection phase, the item now generates discount

$$\left(\frac{k}{160h} \cdot \alpha(I,e) + k \cdot \frac{\ell(e)}{\ell(e_{\max})}\right) \cdot dt$$

on short edges e. However, $\mathrm{rate}_e^{\mathrm{OPT}}(i)$ and $\mathrm{rate}_e^{\mathrm{PAY}}(i)$ are now defined as if *all* edges where long edges. That is,

$$\mathrm{rate}_e^{\mathrm{OPT}}(i) := \sum_{I \in \mathcal{I}_e^{\mathrm{OPT}}(i)} k \cdot \frac{\alpha(I,e) + 1/(8\log h)}{160h}$$

and

$$\mathrm{rate}_e^{\mathrm{PAY}}(i) := \sum_{I \in \mathcal{I}_e^{\mathrm{PAY}}(i)} k \cdot \frac{\alpha(I,e) + 1/(8\log h)}{160h} \enspace .$$

Now, we show that the claim that relates the total discount to the total payment (Claim 2) still holds for this new discount generation scheme.

Claim 6. *The total amount of dicount placed by all items is at most 5% of the total payment generated by all items.*

Proof. The total discount $\mathcal{D}(I)$ generated by an item I during a time interval dt is at most

$$\frac{k}{160h} \sum_{e \in E_{\mathrm{long}} \cup E_{\mathrm{short}}} \alpha(I,e)dt + \sum_{e \in E_{\mathrm{long}}} \frac{k}{160h} \cdot \frac{1}{8\log h} dt + \sum_{e \in E_{\mathrm{short}}} k \frac{\ell(e)}{\ell(e_{\max})} dt \enspace ,$$

where E_{long} and E_{short} denote the set of long and short edges, respectively, on its path to the PAY-server. We can simplify this to

$$\mathcal{D}(i) \le \frac{k}{160h} dt + \sum_{e \in E_{\mathrm{long}}} \frac{k}{160h} \cdot \frac{1}{8\log h} dt + k \frac{\sum_{e \in E_{\mathrm{short}}} \ell(e)}{\ell(e_{\max})} dt \le \frac{k}{80h}\, dt \enspace .$$

The last step follows because $|E_{\mathrm{long}}| \le 4\log h$ and $\sum_{e \in E_{\mathrm{short}}} \ell(e) \le 4 \cdot \ell(e_{\max})/h^2 \le \ell(e_{\max})/(320h)$, which holds (for sufficiently large h) because the edges are geometrically increasing which means that the longest short edge must be a lot shorter than the longest long edge $e_{\max}$.

Since, in every phase at most $h + k \le 2h$ items generate discount we obtain that the total discount generated in this way (without extra discount) is only

$$2h \frac{k}{80h} T \le kT/40 \enspace ,$$

where T is the combined total time of all selection phases of the online algorithm.

The generation of extra discount did not change. Therefore the total extra discount generated over all items is at most $kT/40$, as in the proof of Claim 2. Since the algorithm generates a total payment of kT the claim follows. □

It now may happen that the actual discount generation rate is lower than what $\mathrm{rate}_e^{\mathrm{OPT}}(i)$ or $\mathrm{rate}_e^{\mathrm{PAY}}(i)$ indicate. This could potentially cause issues in Lemma 3.1 where we rely on this discount generation. Nevertheless, the following lemma shows that such problems do not arise.

Lemma 4.1. *Suppose that an edge $e = \{u,v\}$ contributes to* $\mathsf{discounted\text{-}cost}_i^{HST}$ *for the i-th processing phase. Let processing phase $i' < i$ be the last processing phase before i in which edge e was contained in the connected subgraph to be processed. If such an i' does*

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 06/20/17 to 137.205.202.240. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

not exist, let i' be the first processing phase. Further assume that OPT does not traverse edge e between processing phase i' and processing phase $i+1$. Then for every $j \in [i'+1, i]$, the true discount generation rate on e in selection phase j by items exclusively stored in OPT's and PAY's buffer is not less than $\text{rate}_e^{\text{OPT}}(j)$ *and* $\text{rate}_e^{\text{PAY}}(j)$*, respectively.*

Proof. We first observe that e cannot lie on the path from the PAY-server to the root of the tree during any selection phase in $[i'+1, i]$. Otherwise, e would not contribute to $\mathsf{discounted\text{-}cost}_i^{\text{HST}}$ according to our definition. Secondly, the OPT-server and the PAY-server must be on the same side of e during every selection phase in $[i'+1, i]$. This is because, due to the assumptions in the lemma, they cannot traverse e during this interval and if they were located on different sides, e would not contribute to $\mathsf{discounted\text{-}cost}_i^{\text{HST}}$. Further we note that any item generating discount on e during *some* selection phase in $[i'+1, i]$, must either be exclusively stored in OPT's buffer or exclusively stored in PAY's buffer during *each* selection phase in $[i'+1, i]$. This is because e must lie on the path from that item to the PAY-server (and therefore also on the path to the OPT-server) in such a selection phase. However, because neither server crosses e between processing phase i' and selection phase i, the item must in fact be exclusively stored in the OPT or PAY buffer for the entire duration of this interval.

Now assume for contradiction that the lemma is not true. Then there must be an item I for which e is a short edge during some selection phase $j \in [i'+1, i]$ (see Figure 2 for an illustration). e may be a short edge for multiple selection phases which involve different maximum edges $e_{\max}$ with respect to e. We are interested in those selection phases for which this maximum edge is the longest. We choose j to be the first such selection phase in $[i'+1, i]$.

Note that this implies that $e_{\max}$ separates e and the item I from the PAY-server in selection phase j, because e cannot lie on the path from the PAY-server to the root of the tree. Furthermore, in the preceding processing phase, $e_{\max}$ was part of the connected subgraph because otherwise $e_{\max}$ also would be a long edge for I in selection phase $j-1$, which contradicts our choice of j (note that in this case $j-1 > i'$ as the PAY-server cannot traverse edge e without traversing $e_{\max}$). We conclude that, at the start of selection phase j, edge $e_{\max}$ has no payment.

The edge $e_{\max}$ separates the PAY-server from edge e during selection phase j and this must remain true until the first processing phase $j' \geq j$ in which $e_{\max}$ is contained in the connected subgraph again. This also implies that $j' \leq i$, because e is in the connected subgraph of processing phase i.

At most k items can generate payment at any single point in time. Therefore the total combined duration of selection phases $j, \ldots, j'$ must be at least $\ell(e_{\max})/k$. Item I generates discount on e in every selection phase in $[j, j']$. Therefore, during the same time, I generates discount of at least $\ell(e_{\max})/k \cdot k \cdot \ell(e)/\ell(e_{\max}) \geq \ell(e)$ on edge e. Therefore, the discount on e in processing phase i is also at least $\ell(e)$ and e would not contribute to $\mathsf{discounted\text{-}cost}_i^{\text{HST}}$, which contradicts the assumptions of the lemma. □

Finally, we observe that with the new discount generation, Claim 4 improves as follows.

CLAIM 7. *For any edge e and selection phase i,*

- $\text{rate}_e^{\text{OPT}}(i)$ *and* $\text{rate}_e^{\text{PAY}}(i)$ *are upper bounded by* $k/80$ *and*
- $\text{rate}_e^{\text{OPT}}(i)$ *and* $\text{rate}_e^{\text{PAY}}(i)$ *are either 0 or at least* $k/(1280h \log h)$*.*

Using this improved claim in combination with the still valid Lemma 3.1, improves the bound in Claim 5 from $O(\frac{h}{k}(\log h + \log D))$ to $O(\frac{h}{k}(\log h + \log\log h))$, which concludes our analysis for HSTs.

5 Conclusions

We have shown a competitive ratio of $O(\frac{h}{k}(\log h + \log D))$ on tree-metrics of hop-diameter D against an adversary with buffer size $h \geq k$. Our results are tight for $h = (1+\varepsilon) \cdot k$. However, an important open question is whether the results can be improved for $h = k$. On one hand one might want to improve the dependency on k from $\log k$ to, e.g., $\sqrt{\log k}$ (as in the case for uniform metrics). On the other hand, it would be interesting to investigate whether the dependency on D can be improved even further or removed altogether. Even for the seemingly simple case of line-metrics (where D, in principle, is unbounded) this seems difficult and to the best of our knowledge the best upper bound that is independent of D for this case is just the trivial $O(k)$ (if D is large). Another question is what bounds can be achieved when $h < k$. This is a typical resource augmentation setting. For example, how large does k need to be, compared to h, to achieve a constant competitive 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

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 06/20/17 to 137.205.202.240. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

buffer management. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC)*, pages 607–616, 2011.

[3] Anna Adamaszek, Marc P. Renault, Adi Rosén, and Rob van Stee. Reordering buffer management with advice. In *Proceedings of the 11th Workshop on Approximation and Online Algorithms (WAOA)*, pages 132–143, 2013.

[4] 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.

[5] 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, 2015.

[6] Noa Avigdor-Elgrabli and Yuval Rabani. An improved competitive algorithm for reordering buffer management. In *Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 13–21, 2010.

[7] Noa Avigdor-Elgrabli and Yuval Rabani. A constant factor approximation algorithm for reordering buffer management. In *Proceedings of the 24th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 973–984, 2013.

[8] 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.

[9] Yossi Azar, Matthias Englert, Iftah Gamzu, and Eytan Kidron. Generalized reordering buffer management. In *Proceedings of the 31st Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 87–98, 2014.

[10] Siddharth Barman, Shuchi Chawla, and Seeun Umboh. A bicriteria approximation for the reordering buffer problem. In *Proceedings of the 20th European Symposium on Algorithms (ESA)*, pages 157–168, 2012.

[11] Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 184–193, 1996.

[12] Yair Bartal. On approximating arbitrary metrics by tree metrics. In *Proceedings of the 30th ACM Symposium on Theory of Computing (STOC)*, pages 161–168, 1998.

[13] Marcin Bieńkowski, Martin Böhm, Łukasz Jeż, Paweł Laskoś-Grabowski, Jan Marcinkowski, Jiří Sgall, Aleksandra Spyra, and Pavel Veselý. Logarithmic price of buffer downscaling on line metrics. *CoRR*, abs/1610.04915, 2016.

[14] Dan Blandford and Guy Blelloch. Index compression through document reordering. In *Proceedings of the Data Compression Conference (DCC)*, pages 342–351, 2002.

[15] Ho-Leung Chan, Nicole Megow, René Sitters, and Rob van Stee. A note on sorting buffers offline. *Theor. Comput. Sci.*, 423:11–18, 2012.

[16] Matthias Englert, Harald Räcke, and Matthias Westermann. Reordering buffers for general metric spaces. *Theory of Computing*, 6(1):27–46, 2010.

[17] Matthias Englert, Heiko Röglin, and Matthias Westermann. Evaluation of online strategies for reordering buffers. *ACM Journal of Experimental Algorithmics*, 14, 2009.

[18] 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, 2005.

[19] Jittat Fakcharoenphol, Satish B. 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.

[20] Iftah Gamzu and Danny Segev. Improved online algorithms for the sorting buffer problem on line metrics. *ACM Trans. Algorithms*, 6(1), 2009.

[21] 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.

[22] Sungjin Im and Benjamin Moseley. New approximations for reordering buffer management. In *Proceedings of the 25th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1093–1111, 2014.

[23] 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.

[24] Rohit Khandekar and Vinayaka Pandit. Online and offline algorithms for the sorting buffers problem on the line metric. *Journal of Discrete Algorithms*, 2008.

[25] 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 (VMV)*, pages 217–224, 2004.

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

A Appendix

In the following we show that, on the uniform metric, an optimal offline algorithm with buffer size k may have cost that is by an $\Omega(\ln h)$ factor larger than the cost of an optimal offline algorithm with buffer size $h = k/(1-\varepsilon)$, for any constant $\varepsilon > 0$.

The problem for the uniform metric, where two items have either distance 0 or 1 from one another, is the same as a tree of height one in which items arrive at the leaf

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 06/20/17 to 137.205.202.240. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

nodes of the tree. Another, more common interpretation is to view the items as being colored. Two items with the same color have distance 0 and two items with different colors have distance 1. In the following, we will use the latter interpretation. An algorithm incurs a cost of 1 for every color change, that is, switching from removing items of one color to removing items of another color from the buffer.

We construct a sequence σ of items such that an optimal offline algorithm with a buffer of size h can process all colors in order of appearance with only one color change per color, which is clearly optimal. An optimal offline algorithm with buffer-size $k < h$ will require more color changes.

Our input sequence consists of phases $j = 0, \dots, N$, for some large $N \gg h$.

- In phase 0, 1 item of color c_1 arrives, followed by $\lfloor \frac{h}{i \ln h} \rfloor$ items of color c_{i+1} for each $i \geq 1$.
- In phase $j > 0$, 1 item of color c_j arrives. Then, $\lfloor \frac{h}{i \ln h} \rfloor - \lfloor \frac{h}{(i+1) \ln h} \rfloor$ items of color c_{j+i+1} arrive for each $i \geq 1$.

This sequence has the property that by the end of phase j, $\lfloor \frac{h}{i \ln h} \rfloor$ items of color c_{j+i+1}, for $i \geq 1$ have arrived.

A buffer of size $h' = 1 + \sum_{i \geq 1} \lfloor \frac{h}{i \ln h} \rfloor$ is necessary and sufficient to process this sequence with cost equal to the number of colors $N + 1 + \lfloor h / \ln h \rfloor = O(N)$. Such an algorithm holds all items of colors strictly larger than c_{j+1} in the buffer at the end of phase j (as well as one item of color c_{j+1} which can be removed from the buffer one step before the item of color c_{j+1} which arrives in phase $j + 1$ is removed from the buffer). There are h' such items. Note that $h \geq h' \geq h \cdot (1 - \frac{1 + \ln \ln h}{\ln h})$, for sufficiently large h.

An algorithm processing the sequence with a buffer of size $k < h'$ will have at least $h' - k$ of these items removed from the buffer before the end of phase j. Let ϕ_j be the number of these items removed from the buffer before the end of phase j. Summing over all phases, we must have $\Phi := \sum \phi_j \geq N \cdot (h' - k)$.

Switching to a color c_{j+i+1} in phase j removes at most $\lfloor \frac{h}{i \ln h} \rfloor$ items from the buffer and thus contributes at most $i \cdot \lfloor \frac{h}{i \ln h} \rfloor \leq \frac{h}{\ln h}$ to Φ. Over all phases and colors, we therefore must make at least $\Phi \cdot \frac{\ln h}{h}$ such color changes. For $k \leq (1 - \epsilon) \cdot h$ this gives

$$\begin{aligned} N \cdot (h' - k) \cdot \frac{\ln h}{h} &\geq \left(\epsilon - \frac{1 + \ln \ln h}{\ln h} \right) \cdot \ln h \cdot N \\ &= \Theta(\ln h) \cdot N \end{aligned}$$

color changes. Combined with the fact that an optimal algorithm with a buffer of size h can process the sequence at cost $O(N)$, this proves our claim.

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited