Weighted Reordering Buffer Improved via Variants of Knapsack Covering Inequalities

Sungjin Im[1] and Benjamin Moseley[2](✉)

[1] Department of Electrical Engineering and Computer Science, University of California, Merced, CA 95344, USA
sim3@ucmerced.edu

[2] Washington University in St. Louis, St. Louis, MO 63130, USA
bmoseley@wustl.edu

Abstract. We consider the weighted Reordering Buffer Management problem. In this problem a set of n elements arrive over time one at a time and the elements can be stored in a buffer of size k. When the buffer becomes full, an element must be output. Elements are colored and if two elements are output consecutively and they have different colors then a switching cost is incurred. If the new color output is c, the cost is w_c. The objective is to reorder the elements to minimize the total switching cost in the output sequence.

In this paper, we give an improved randomized $O(\log\log\log k\gamma)$-approximation for this problem where γ is the ratio of the maximum to minimum weight of a color, improving upon the previous best $O(\log\log k\gamma)$-approximation. Our improvement builds on strengthening the standard linear program for the problem with non-standard knapsack covering inequalities. In particular, by leveraging the structure of these inequalities, our algorithm manages to render several random procedures more powerful and combine them effectively, thereby giving an exponential improvement upon the previous work.

1 Introduction

Buffer management theory focuses on studying how a buffer, typically of limited size, can be used to support an application. Due to the numerous applications of buffers, such as in networking and memory management, a rich and diverse theory has been developed. One well studied problem is the Reordering Buffer Management problem. In this problem, there is a set of n elements that arrive over time and the elements are colored. It is assumed that one element arrives at each time from 1 to n. There is a buffer of size k where the arriving elements can be stored in, and when the buffer becomes full an element must be output. If an element is output that has the same color as the previous element output, then there is no cost for outputting the element. Otherwise when the color changes, if the element output has color c then a cost of w_c is incurred. The goal is to reorder elements in the buffer to minimize the total cost incurred. Note that if

S. Im—Supported in part by NSF grant CCF-1409130.

$w_c = 1$ for all colors c, then the goal is just to minimize the number of times the color changes in the output sequence.

The Reordering Buffer Management problem, which was elegantly formulated in [20], seeks to understand the fundamental tradeoff between the limited buffer size and the context switching cost. This simple, yet powerful, model captures several practical problems seen in paint shops, graphics rendering, as well as network buffering. For example, consider a server that forwards messages to clients. When switching from sending messages from one client to another, a cost is paid representing the overhead of the context switch. One may desire to limit the number of times the server switches between clients by buffering messages and reordering them to minimize the context switches. See [3,10,19,20] for more applications of this model. Besides the practical importance of the model, the model has been well studied theoretically. The main theoretical interest comes from the simplicity of the model and the fact that, yet being simple, the model is algorithmically challenging. Indeed, the model has been extensively studied both online and offline [1,2,4,6–8,12,13,20]. However, even though this problem has been rigorously studied for over a decade, the complexity of the problem is not well understood.

The challenges of the model emerge even when the weights of the colors are uniform. The uniform weight problem is known to be NP-Hard [4,12]. Further, algorithms that initially would seem to be ideal candidates for the problem fail to have a small approximation ratio. For example, simple algorithms such as Largest Color First, which outputs color that has the largest number of elements in the buffer, First-In First-Out and Least Recently Used all have strong lower bounds on their approximation ratios [20]. Due to this, previous work has focused on developing more sophisticated algorithms for the problem.

Initially an $O(\log^2 k)$-approximation was shown for the problem when the weights are all uniform [20]. This been improved through a sequence of woks [2,6,8,13]. Recently, the complexity of the unweighted case has been resolved up to constant factors and $O(1)$-approximation algorithms are known [7,16]. For the weighted version of the problem, the currently best known approximation is a randomized $O(\log \log k\gamma)$-approximation where here γ is the ratio of the maximum to minimum weight of a color [16]. Several algorithms are known which use resource augmentation where the algorithm is given a larger buffer than the optimal solution [12,20]. A key open question in the area is determining the right approximation ratio for the non-uniform weight version of the problem.

Results: In this work we improve upon the best known approximation ratio for the non-uniform weighted Reordering Buffer Management problem. We develop an algorithm that exponentially improves upon the best previously known algorithm's approximation guarantee of $O(\log \log k\gamma)$. Our main result is the following.

Theorem 1. *There exists a randomized $O(\log \log \log k\gamma)$-approximation algorithm for the weighted Reordering Buffer Management problem where γ is the ratio of the maximum to minimum weight of a color.*

For the online version of the problem, an $\Omega(\log\log k)$ lower bound is known on randomized algorithms as well as an $O((\log\log k\gamma)^2)$ upper bound [2,5]. This is the first case where the offline problem has been shown to have an approximation ratio better than the best possible competitive ratio. To show the main result, we introduce new linear program rounding techniques. In particular, we add knapsack covering inequalities to the standard linear program for the problem. See [11] for details on knapsack covering inequalities. We extend the definition of traditional knapsack covering inequalities by adding additional parameters to the inequalities, which prove to be very useful. By leveraging the structural properties given to us by these inequalities, we can circumvent barriers faced in previous rounding techniques. The inequalities were used in the context of unweighted reordering buffer in the authors' previous work [16]. However, they were only able to use the inequalities to give a small constant factor improvement for the unweighted case, but did not know how to use them for the weighted case. In this work, we demonstrate the power of our variants of knapsack covering inequalities by giving an exponential improvement for the weighted case. The inequalities will be further discussed in Section 2 and 3. We will give an overview of our algorithm and analysis in Section 3 together with the discussion on how this work is differentiated from the previous work.

Related Work: Besides the mentioned work on the offline buffer reordering problem, the problem has also been considered online. It is known that in the online setting that there is lower bound of $\Omega(\sqrt{\frac{\log k}{\log\log k}})$ on the competitive ratio of deterministic schedulers and $\Omega(\log\log k)$ on randomized schedulers [2]. The work of [2] gave the first $O(\sqrt{\log k})$-competitive deterministic online scheduler, essentially resolving the deterministic case when colors are unweighted. The recent work of [8] has resolved the randomized case when colors are unweighted by giving an $O(\log\log k)$-competitive online algorithm. For the weighted version of the problem, previous the best known online algorithm is a deterministic $O(\sqrt{\log k\gamma})$-competitive algorithm, which has recently been improved to $O((\log\log k\gamma)^2)$ randomized algorithm [5]. The problem has also been considered in the stochastic setting [14].

The Reordering Buffer Management problem has been generalized and been studied in several other settings. Most generalizations consider extending the definition of the cost function when switching colors. The work of [15,17] considers when the cost of switching between two colors forms a line metric and [9,18] considers when the costs form a general metric.

Organization: The paper is organized as follows. In Section 2 we start by introducing the linear programming relaxation we will consider throughout the paper as well as some useful lemmas and a simple randomized sampling procedure. In Section 3 we give a high-level sketch of our algorithm and analysis to show the intuition guiding our work. In Section 4 we formally introduce our algorithm and finally in Section 5 we give the formal proof of the algorithm's guarantees.

2 Preliminaries

In this section, we introduce our linear program, a few useful lemmas as well as a simple sampling scheme that our algorithm will utilize. We begin by introducing our linear program. We call a continuous sequence of elements of the same color in the output sequence a *color block*. We require elements for a color are output in first-in first-out order without loss of generality. Each color block (or simply block) b is a triple (i, t, ℓ) specifying the first element in the color block e_i, the time the block is scheduled t and the length of the sequence ℓ. Note that one can deduce all ℓ elements that are output in the block from the triple, and we let $(i', t') \in b$ if element $e_{i'}$ is output at time t' in the color block. Let B be the set of all possible color blocks in the output sequence. Note that B is polynomial in n. Let E denote the set of all elements.

Below is an integer programming formulation for the problem. The variable x_b specifies if the color block b is in the output sequence. The variable $y_{i,t}$ specifies if element e_i is output at time t and $\beta_{i,t}$ specifies if the element e_i was output at or before time t. We use the notation $E_{b,\le t}$ to denote all the elements in the color block b which were output in b at or before time t. For a color block b let $c(b)$ be the color of the elements in b and, likewise, let $c(e_i)$ denote the color of the element e_i. Let $p(i)$ to denote the element for color $c(e_i)$ which is the previous element of this color that arrives before e_i – that is, the latest arriving element for color $c(e_i)$ that arrives before e_i.

$$\min \quad \sum_{b \in B} w_{c(b)} x_b \tag{IP}$$

$$\text{s.t.} \quad y_{i,t} = \sum_{(i,t) \in b} x_b \qquad \forall i, t \tag{1}$$

$$\sum_{i \in [n]} y_{i,t} = 1 \qquad \forall t \ge k+1 \tag{2}$$

$$\sum_{t \in [k+1, k+n]} y_{i,t} = 1 \qquad \forall i \in [n] \tag{3}$$

$$\beta_{i,t} = \sum_{i, t' \le t} y_{i,t'} \qquad \forall i \in [n], t \in [k+1, k+n] \tag{4}$$

$$\beta_{p(i),t-1} \ge \beta_{i,t} \qquad \forall i \in [n], t \ge k+1 \tag{5}$$

$$\sum_{b \in B \setminus B'} (|E_{b,\le t} \setminus E'|) x_b \ge (t - k - |E'|)(1 - \sum_{b \in B'} x_b) \quad \forall t \in [k+1, k+n], B' \subseteq B, E' \subseteq E \tag{6}$$

$$x_b \in \{0, 1\} \qquad \forall b \in B \tag{7}$$

Constraint (2) ensures that at most one element is output at each time. Constraint (3) ensures that each element is output at some time. Constraints (1) and (4) set the y and β variables according to the x variables. Constraint (5) ensures that elements are output in first-in-first-out order. Finally, the knapsack covering inequality is given in Constraint (6). We obtain an LP relaxation by replacing (7) with $x_b \in [0, 1]$.

Variants of Knapsack Covering Inequalities: The key constraints (6) deserve special attentions. The constraints are over all $B' \subseteq B$ and $E' \subseteq E$; therefore, there are exponentially many such constraints. To get a feel of the constraints, consider the simplest case that $B' = \emptyset$ and $E' = \emptyset$ with a fixed time t. Then the left-hand-side is simply the total number of elements output by time t, where each color block counts the number of elements it outputs by time t, and adds it to the summation. The right-hand-side is $t-k$, hence (6) states that at least $t-k$ elements must be output by time t due to the space limit of k for the buffer. Now consider an arbitrary E' with $B' = \emptyset$. Then, (6) lower bounds the total number of elements that has to be output in individual blocks by time t with the elements in E' excluded. In fact, (6) is a standard knapsack covering inequality if $B' = \emptyset$. Intuitively, this prevents the LP from cheating with elements.

In contrast, the power of having B' does not seem immediate – if there is a $b \in B'$ where $x_b = 1$, then the inequality is trivially satisfied, otherwise it becomes a standard knapsack inequality. However, having B' turns out to be very useful in randomized rounding. Recall that in the Reordering Buffer Problem the costs are determined by color blocks output, hence the complexity cannot be understood well without having a good control over blocks. In the fractional LP solution, we will be able to exclude some fractional color blocks and focus on "good" fractional blocks to derive nice probabilistic properties. The overall analysis is done with carefully chosen E' and B'.

To see why having $B' \neq \emptyset$ is useful, consider adding a color block b which has k^2 elements in it output by time t, but $x_b = \frac{1}{k}$ in an LP solution. From this color block, a total 'volume' of elements output is k. However, the color block is chosen by very little in the LP. By adding b to B', the right hand side decreases by a multiplicative factor of $1 - \frac{1}{k}$ while the left hand side decreases by an additive factor of k. This strengthens the LP. For instance, in the case that E' is chosen such that, $t - k - |E'| \leq k$, then the right hand side only decreases by an additive $k \cdot \frac{1}{k} = 1$, while the left hand side decreases by k. The added power is that the LP cannot output a large volume of elements using color blocks with many elements, but only choosing those color blocks themselves by a small amount.

Finally, we discuss the separation oracle regarding the constraints (6). Unfortunately, we do not know if there is a polynomial-time separation oracle when the constraint is defined over all $B' \subseteq B, E' \subseteq E$. However, there is a very easy separation oracle if either B' or E' is fixed. It turns out that we only need to consider polynomially many different E' for our analysis. That is, even though such a collection of E' is determined by $\{x_b\}$, we only need to look at polynomially many E', and this will allow us to solve the LP in polynomial time to the extent of our need. We defer the proof of solving the LP in polynomial time to a full version of this paper.

Useful Lemmas and Observations: Now we show some lemmas that will be useful throughout the paper. We will refer to x_b as the *height* of the color block

b in the LP solution. The following lemma will allow our algorithm to output a color at time t if the elements in the algorithm's buffer for the color at time t have been processed by a set of color blocks of substantial height in the LP by time t. This is similar to lemmas used in [2,16] and is standard for the problem. The proof is omitted.

Lemma 1. *Consider any color block b output by our algorithm A which starts at time t and ends at time t''. Let $t' \geq k+1$ be the earliest time before t such that A scheduled no element of color $c(b)$ during $[t', t)$. Suppose that the LP has a set of color blocks S of total height at least ϵ (i.e. $\sum_{b' \in S} x_{b'} \geq \epsilon$) that each have processed at least one element in b by time t – in particular, such a set S exists if the first element e_i in b is processed by at least ϵ by time t in the* LP. *Then there is a set of color blocks of total height at least ϵ for color $c(e_i)$ in the* LP*'s solution that end during $(t', t'']$.*

The following proposition follows from constraint (1) in the LP.

Proposition 1. *Suppose that the* LP *has a set $\mathcal{I}$ of color c color blocks of color c and total height at least h, all starting no later than some time t. Further, suppose that each of blocks scheduled after time step t at least ℓ (possibly different) elements that entered the buffer no later than time step t. Then it is the case that* LP *has at least a total volume of $h\ell$ of elements of color c in its buffer at time t.*

Next we state a lemma that will allow us to compare against an LP with a slightly smaller buffer size. In particular, we will solve the LP with a buffer of size $k' = k - \frac{k}{\log k\gamma}$. This can be done by losing only an $O(1)$ factor in the approximation ratio as the lemma shows. The following lemma was shown in [5] and similar lemmas are known for the unweighted version of the problem. The proof of the lemma is omitted.

Lemma 2. *For any input sequence and $k' < k$, respectively, $\mathrm{OPT}_{k'} \leq O(1) \cdot (\frac{k}{k'} + (k-k')\frac{\log k'\gamma}{k'})\mathrm{OPT}_k$, where OPT_s denotes the cost of the optimal solution using a buffer of size s.*

Finally we introduce a sampling scheme which was originally used in [5] which is independent rounding coupled with a threshold rounding. We refer to a color block as *maximal* in the algorithm's output sequence if when the color block ends there are no more elements of the same color in the buffer at that time. In the sampling we will sample a color block b in the LP solution with probability $\frac{1}{\alpha}x_b$ if $\frac{1}{\alpha}x_b < 1$ and with probability 1 if $\frac{1}{\alpha}x_b \geq 1$. We call this the α-sampling. Let Bag denote the pool of color blocks sampled. Let t_i^α denote the earliest time that a color block in Bag schedules the element e_i and if no such color block exists set $t_i^\alpha = \infty$. We say that element e_i is α-ready at time t_i^α or at any time later. The proof of the following lemma is an extension of a proof found in [5]. The proof is deferred to a full version of this paper. For any set of color blocks A, let $x_{i,A}(t)$ denote the amount by which the element e_i is processed by color blocks in A by time t.

Lemma 3. *For any constant $0 < \alpha < 1$, the α-sampling satisfies the following properties :*

- *For any set of blocks A, the element e_i is α ready by time t with probability at least $(1 - 1/e) \min\{x_{i,A}(t)/\alpha, 1\}$. For any distinct elements e_i and e_j that are not processed by the same blocks in A by time t, the events that they become α ready by sampling color blocks from A are independent.*
- *The previous property implies that that for any element e_i and time step t such that $\beta_{i,t} \leq \alpha$, $\Pr[t_i^\alpha \leq t] \geq (1-1/e)\beta_{i,t}/\alpha$. This probability occurs independently for two elements if they are not processed by the same color blocks ever by time t. In particular, this is always true for elements of different colors.*
- *Consider any collection B' of disjoint maximal color blocks where each block $b' \in B'$ schedules at least one element i at time $t \geq t_i^\alpha$. The expected total cost of the blocks in B' is at most $(1/\alpha)\mathsf{Cost}_{\mathsf{LP}}$.*

The properties of the sampling scheme will be very useful for our analysis. The first property ensures that an element e_i can be scheduled by time t with probability proportional to amount it has been processed by the LP at time t, $\beta_{i,t}$. The second property ensures that the sampling is independent for elements of different colors or for elements where we can identify a set of color blocks that do not process both of them. The third property shows that the cost of outputting elements after their α-ready time can be charged to the LP.

3 Algorithm and Analysis Overview

In this section we give an outline of our algorithm and the analysis. Due to space constraints, the main analysis is deferred to a full version of this paper. The actual analysis is more involved but our goal here is to give the underlying intuition while ignoring lower level details.

Our algorithm begins by solving the linear program for the problem where the buffer size is set to be $k' = k - \frac{k}{\log k\gamma}$. The solution to the linear program is used to guide the algorithm on how elements should be output. The algorithm itself, works like an online algorithm that outputs elements sequentially from time $k+1$ to time $n+k$. At any time t where there is an element in the algorithm's buffer $\mathcal{B}(t)$ that has the same color as the previous element output, the algorithm will output such an element. Otherwise, the algorithm needs to choose a color to switch to. At these points in time, the algorithm will use a set of rules to decide which color to switch to. These rules on the color to switch to are guided by the LP solution.

We now discuss the rules that the algorithm uses to decide the color to switch to. These rules are inspired by the previous work of [16] on the Buffer Reordering Management problem. The first set of rules are simple and similar to previous work. The algorithm is free to switch to any color c where (1) the elements in $\mathcal{B}(t)$ for color c have been processed by color blocks in the LP of total height at least ϵ (2) there is a an element for color c in $\mathcal{B}(t)$ that is α-ready or (3) there are more than $k/10$ elements in $\mathcal{B}(t)$ for color c. The cost of execution rule (1) is easily charged to the LP using Lemma 1 and the same is holds for rule (2) using Lemma 3. The cost of rule (3) can by charged to the LP using observations used in [16]. Intuitively, a color cannot be output many times if it occupies $\Theta(k)$ space in the buffer without the LP also needing to output the color. This is because the LP would need to store all of these elements, contradicting its buffer size and, therefore, we can charge to the LP.

The first three simple rules are used to give structural properties on the algorithm and LP's status when these rules cannot be applied. The interesting rules are the final two rules to be mentioned soon. Recall that the LP solution has buffer of smaller size than the algorithm. This implies that at any time t, the LP must have processed the elements in $\mathcal{B}(t)$ by a $\frac{k}{\log k\gamma}$ aggregate amount. The final two rules are based on whether this aggregate amount of work is focused mostly on elements for colors which occupy a large portion of the algorithm's buffer or a smaller portion of the buffer.

Let $C_s(t)$ be the set of colors where the algorithm has less than $\frac{k}{\log^3 k\gamma}$ elements for each of these colors in its buffer and let $C_b(t)$ be the remaining colors where the algorithm has more than$\frac{k}{\log^3 k\gamma}$ elements for these colors. In [16] it was shown that if a constant fraction of the work the LP has done on elements in $\mathcal{B}(t)$ are for colors in $C_s(t)$ then we should have sampled an element in $\mathcal{B}(t)$ for a colors in $C_s(t)$ with probability at least $1 - \frac{1}{k^2}$. Intuitively, a large volume of work was focused on these colors. Further, knowing that color blocks for colors in $C_s(t)$ can only include $\frac{k}{\log^3 k\gamma}$ elements from $\mathcal{B}(t)$, one can use concentration inequalities to show that we should have sampled such an element. Since we fail to sample an element with low probability, it can be shown that there is some color we can switch to such that the expected cost of switch to this color is small compared to the LP's cost. This will be rule (4).

The final rule and analysis of this rule is where our work differs from [16] and is where the knapsack covering inequalities proves to very useful. The algorithm will only perform this rule so long as the previous rules do not apply. In particular, since we do not execute rule (4), we know that a constant fraction of the work the LP has done by time t on the elements in $\mathcal{B}(t)$ are on elements that have colors in $C_b(t)$. Let $n^A_c(t)$ denote the number of elements for color c in $\mathcal{B}(t)$. Let $n^O_c(t)$ be the number of elements in the LP at time t for color c that have been processed by at most $1/2+2\epsilon$. The first step is showing that there is a color $c \in C_b(t)$ where $n^A_c(t) \geq \frac{3}{5} n^O_c(t)$. This will follow from the fact that if it were not true, then the LP has many elements the algorithm does not have for colors in $C_b(t)$. But then, we also know that no element in $\mathcal{B}(t)$ is processed by ϵ, since we did not use rule (1). Thus, the LP must have all elements in $\mathcal{B}(t)$ and these extra elements in its buffer, but this will cause a contradiction to the LP's buffer size. Rule (5) will allow the algorithm to switch to a color c where $n^A_c(t) \geq \frac{3}{5} n^O_c(t)$. Then we will show that we can execute this rule at most $O(\log\log k\gamma)$ times for a fixed color before the LP must output this color, allowing us to charge to this point in time in the LP. The argument follows by observing that if we output a color with at least $\frac{k}{\log^3 k\gamma}$ elements and $n^A_c(t) \geq \frac{3}{5} n^O_c(t)$ more than

$O(\log\log k\gamma)$ times and the LP does not do this color, then the LP must have $(\frac{k}{\log^3 k\gamma})(1+\frac{3}{5})^{O(\log\log k\gamma)} > \Omega(k)$ elements for this color in its buffer at some time. This will draw a contradiction and therefore we can only output a color $O(\log\log k\gamma)$ times using this rule before we can find a time to charge to in the LP solution.

Naively, rule (5) will show our algorithm is a $O(\log\log k\gamma)$-approximation. However, we can improve this by showing that, in fact, we only perform rule (5) with low probability. Say with probability at most $\frac{1}{\log\log k\gamma}$. This will allow us to show that in expectation we only need to charge $O(1)$ to the LP. Showing this event happens with low probability will follow from the knapsack covering inequalities and by bosting the probability a block is randomly sampled in the LP by a $\Theta(\log\log\log k\gamma)$ factor. We note that these knapsack inequalities were not used in [16] to show a $O(\log\log k\gamma)$-approximation and this is how we circumvent hurdles faced in the analysis of [16]. In particular, it seems perfectly plausible using the standard LP that we could output a color $O(\log\log k\gamma)$ in this step with good probability. To see why this event happens with low probability, consider the knapsack covering inequality for time t.

$$\sum_{b\in B\setminus B'}(|E_{b,\le t}\setminus E'|)x_b \ge (t-k'-|E'|)(1-\sum_{b\in B'}x_b) \quad \forall B'\subseteq B, E'\subseteq E$$

Our goal is to show that there is $\Omega(1)$ height of color blocks the LP has scheduled on elements in $\mathcal{B}(t)$ if we execute rule (5). We will use this coupled with setting $\alpha < \frac{1}{\Theta(1)\log\log\log k\gamma}$ for the sample. If we can find such a height on color blocks in the LP, then the probability no element in $\mathcal{B}(t)$ is α-ready is at most $2^{-\Theta(\log\log\log k\gamma)} = \frac{1}{(\log\log k\gamma)^{\Theta(1)}}$ by Lemma 3. Further, if we did sample such an interval then an element in $\mathcal{B}(t)$ would be α ready at time t. Thus, we will have the desired probability and here one can see why we required that we boosted the probabilities in the sampling by a factor of $\Theta(\log\log\log k\gamma)$. To see why such a such a height exists, consider setting E' to be all elements that arrived by time t except those $\mathcal{B}(t)$ and B' to be the height of color block including that process at least one element in $\mathcal{B}(t)$ before time t. The left hand side must be 0, but $(t-k'-|E'|) = \frac{k}{\log k\gamma}$ since $|E'| = t-k$. Thus, it must be the case that $\sum_{b\in B'} x_b = 1$.

This is the intuition on how we can show that the cost accumulated by the algorithm by rule (5) is at most $O(1)$ multiplied by the cost of the LP in expectation. Unfortunately, the actual proof is much more involved. In particular, there is a dependency at different times on whether or not elements are α-ready. The proof needs to deal with these dependencies delicately. We handle this by showing that, in fact, a very large number of elements will become α-ready with good probability. Then using this we can group time steps together in such a way that if we succeed at a particular time, we will succeed at the later times where there are significant dependencies. This will then allow us to bound the cost of rule (5).

4 Algorithm

We require some notation to define formally the algorithm. Let ϵ be $\Theta(\frac{1}{\log\log\log k\gamma})$ and α at most ϵ. We will later set $\epsilon = \frac{1}{2^{20}\log\log\log k\gamma}$ and $\alpha = \epsilon$. Let $\mathcal{B}(t)$ denote (the set of elements in) the algorithm's buffer at time t. Let $n_c^A(t)$ denote the number of elements for color c in $\mathcal{B}(t)$. Let $n_c^O(t)$ be the number of elements in the LP at time t for color c that have been processed by at most $1/2 + 2\epsilon$. Intuitively, one should think of these elements as the ones not done by the LP. Let $C_s(t)$ contain all colors c where $0 < n_c^A(t) \leq \frac{k}{\log^3 k\gamma}$ and $C_b(t)$ contain all colors c where $n_c^A(t) > \frac{k}{\log^3 k\gamma}$. Let $E^O(t)$ be the set of elements that have been processed by at most $1/2 + 2\epsilon$ in the LP at time t that are not in $\mathcal{B}(t)$, i.e. $E^O(t) := \{e_i \mid e_i \notin \mathcal{B}(t), i \leq t, \beta_{i,t} \leq 1/2 + 2\epsilon\}$. Let $c^*(t)$ be the color such that color blocks in the LP for color $c^*(t)$ that intersect time t have height greater than 1/2, if it exists. Note that there can only be one such color. Let $v_{c,t}^O = \sum_{i,c(e_i)=c} 1 - \beta_{i,t}$ denote the remaining volume of elements for color c in the LP at time t.

Let $t_{c,1}$ be the first time the LP accumulates cost ϵw_c for color c. That is, there exists a set of color blocks for color c of height at least ϵ which *start* at time $t_{c,1}$ or earlier. Assuming $t_{c,i-1}$ is defined, let $t_{c,i}$ be the earliest time that the LP accumulates cost ϵw_c for color c since time $t_{c,i-1}$. That is, during $(t_{c,i-1}, t_{c,i}]$ there exists a set of color blocks for color c of total height at least ϵ that start during $(t_{c,i-1}, t_{c,i}]$. Let $\mathcal{T}_c$ be the set of such times for color c. With these definitions in place, the algorithm can be defined as follows. The algorithm attempts to execute the rules in the order presented.

Algorithm:

Rule (i) If there is a set of color blocks S in the LP of total aggregate height ϵ (i.e. $\sum_{b\in S} x_b \geq \epsilon$) that each processes at least one element in $\mathcal{B}(t)$ for color c by time t then output color c. In particular, in a special case, if there is an element in $e_i \in \mathcal{B}(t)$ processed by ϵ in the LP, then switch to color $c(e_i)$.

Rule (ii) If there is an element $e_i \in \mathcal{B}(t)$ that is α ready at time t then switch to color $c(e_i)$.

Rule (iii) If there is a color c where $n_c^A(t) \geq k/10$, switch to color c.

Rule (iv) If the LP has processed elements in $\mathcal{B}(t)$ corresponding to colors in $C_s(t)$ by a total of at least $(|E^O(t)| + \frac{k}{\log k\gamma})/10$ by time t then switch to a color $c \in C_s(t)$ such that earliest time $t' \in \mathcal{T}_c$ after t is also the earliest time in $\cup_{c'\in C_s(t)}\mathcal{T}_{c'}$ after t.

Rule (v) We perform this rule if none of the others apply. Let L be the set of colors $c \in C_b(t)$ such that $n_c^A(t) \geq \frac{3}{5}n_c^O(t)$. The algorithm switches to a color $c \in L$ such that the earliest time t' in $\mathcal{T}_c$ after t is also the earliest time in $\cup_{c'\in L}\mathcal{T}_{c'}$ after t. We will show that $L \neq \emptyset$ if the earlier rules cannot be used.

5 Analysis

In this section our goal is to prove Theorem 1 by analyzing the algorithm given in the previous section. Recall that we solve the LP with a buffer size $k' := k - \frac{k}{\log k}$ and our algorithm has a buffer of size k. Throughout the proof, we will let LP denote the cost of the LP solution. To prove the approximation ratio of our algorithm, we bound the cost of each of the rules in the algorithm separately. First consider Rule (i). The following lemma is immediately implied by Lemma 1.

Lemma 4. *The total cost accumulated by the algorithm due to executing* Rule (i) *is at most* $O(\frac{1}{\epsilon})\mathsf{LP}$.

Next consider the cost accumulated by Rule (ii). By applying Lemma 3, we have the following lemma.

Lemma 5. *The total expected cost incurred when the algorithm executes* Rule (ii) *is at most* $O(\frac{1}{\alpha})\mathsf{LP}$.

Next we consider the cost accumulated by Rule (iii) and Rule (iv). In this case, we appeal to the proofs shown in [16]. We note that this proof relies on structural properties in the elements in $\mathcal{B}(t)$ have since the algorithm did not use Rule (i) or Rule (ii). These structural properties are sufficient for the proofs shown in [16].

Lemma 6 ([16]). *The total cost incurred when the algorithm executes* Rule (iii) *is at most* $O(\frac{1}{\epsilon})\mathsf{LP}$.

Lemma 7 ([16]). *The total expected cost incurred when the algorithm executes* Rule (iv) *is at most* $O(\frac{1}{\epsilon})\mathsf{LP}$.

We now focus on bounding the expected number of times an element can be output due to Rule (v). The main analysis focuses on proving the following lemma.

Lemma 8. *Consider any time* $t_1 \in \mathcal{T}$ *and let* t_2 *be the next time in* $\mathcal{T}$ *after* t_1*. The expected number of times we execute* Rule (v) *is at most* $O(1)$ *during* $[t_1, t_2)$.

Once we have this lemma, combining it and the previous four lemmas proves Theorem 1. This is because between any two times t_1 and t_2 in $\mathcal{T}_c$ the LP accumulates a cost of at least ϵw_c and we can charge to this cost to bound the expected cost of executing Rule (v) by $O(1)\mathsf{LP}$. Showing this lemma will complete the analysis. Due to space constraints, the proof is deferred.

References

1. Aboud, A.: Correlation clustering with penalties and approximating the reordering buffer management problem. Masters thesis, Computer Science Department, The Technion - Israel Institute of Technology (2008)

2. Adamaszek, A., Czumaj, A., Englert, M., Räcke, H.: Almost tight bounds for reordering buffer management. In: STOC, pp. 607–616 (2011)
3. Alborzi, H., Torng, E., Uthaisombut, P., Wagner, S.: The k-client problem. J. Algorithms **41**(2), 115–173 (2001)
4. Asahiro, Y., Kawahara, K., Miyano, E.: Np-hardness of the sorting buffer problem on the uniform metric. Discrete Applied Mathematics **160**(10–11), 1453–1464 (2012)
5. Avigdor-Elgrabli, N., Im, S., Moseley, B., Rabani, Y.: On the randomized competitive ratio of reordering buffer management with non-uniform costs. Manuscript (2014)
6. Avigdor-Elgrabli, N., Rabani, Y.: An improved competitive algorithm for reordering buffer management. In: SODA, pp. 13–21 (2010)
7. Avigdor-Elgrabli, N., Rabani, Y.: A constant factor approximation algorithm for reordering buffer management. In: SODA (2013)
8. Avigdor-Elgrabli, N., Rabani, Y.: An improved competitive algorithm for reordering buffer management. In: 54th Annual IEEE Symposium on Foundations of Computer Science, FOCS, October 26–29, 2013, Berkeley, CA, USA, pp. 1–10 (2013)
9. Bar-Yehuda, R., Laserson, J.: Exploiting locality: approximating sorting buffers. J. Discrete Algorithms **5**(4), 729–738 (2007)
10. Blandford, D.K., Blelloch, G.E.: Index compression through document reordering. In: DCC, pp. 342–351 (2002)
11. Carr, R.D., Fleischer, L.K., Leung, V.J., Phillips, C.A.: Strengthening integrality gaps for capacitated network design and covering problems. In: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms, SODA 2000, Philadelphia, PA, USA, pp. 106–115. Society for Industrial and Applied Mathematics (2000)
12. Chan, H.-L., Megow, N., Sitters, R., van Stee, R.: A note on sorting buffers offline. Theor. Comput. Sci. **423**, 11–18 (2012)
13. Englert, M., Westermann, M.: Reordering buffer management for non-uniform cost models. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 627–638. Springer, Heidelberg (2005)
14. Esfandiari, H., Hajiaghayi, M.T., Khani, M.R., Liaghat, V., Mahini, H., Räcke, H.: Online stochastic reordering buffer scheduling. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014. LNCS, vol. 8572, pp. 465–476. Springer, Heidelberg (2014)
15. Gamzu, I., Segev, D.: Improved online algorithms for the sorting buffer problem on line metrics. ACM Transactions on Algorithms, **6**(1) (2009)
16. Im, S., Moseley, B.: New approximations for reordering buffer management. In: SODA, pp. 1093–1111 (2014)
17. Khandekar, R., Pandit, V.: Online sorting buffers on line. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 584–595. Springer, Heidelberg (2006)
18. Kohrt, J.S., Pruhs, K.R.: A constant approximation algorithm for sorting buffers. In: Farach-Colton, M. (ed.) LATIN 2004. LNCS, vol. 2976, pp. 193–202. Springer, Heidelberg (2004)
19. Krokowski, J., Räcke, H., Sohler, C., Westermann, M.: Reducing state changes with a pipeline buffer. In: VMV, pp. 217 (2004)
20. Räcke, H., Sohler, C., Westermann, M.: Online Scheduling for Sorting Buffers. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 820–832. Springer, Heidelberg (2002)