Optimal Online Buffer Scheduling for Block Devices *

Anna Adamaszek
Department of Computer Science and
Centre for Discrete Mathematics and its
Applications (DIMAP)
University of Warwick, Coventry, UK
A.M.Adamaszek@warwick.ac.uk

Artur Czumaj
Department of Computer Science and
Centre for Discrete Mathematics and its
Applications (DIMAP)
University of Warwick, Coventry, UK
A.Czumaj@warwick.ac.uk

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

Harald Räcke
Institut für Informatik
Technische Universität München
Munich, Germany
raecke@in.tum.de

ABSTRACT

We introduce a buffer scheduling problem for block operation devices in an online setting. We consider a stream of items of different types to be processed by a block device. The block device can process all items of the same type in a single step. To improve the performance of the system a buffer of size k is used to store items in order to reduce the number of operations required. Whenever the buffer becomes full a buffer scheduling strategy has to select one type and then a block operation on all elements with this type that are currently in the buffer is performed. The goal is to design a scheduling strategy that minimizes the number of block operations required.

In this paper we consider the online version of this problem, where the buffer scheduling strategy must make decisions without knowing the future items that appear in the input stream. Our main result is the design of an $O(\log \log k)$-competitive online randomized buffer scheduling strategy. The bound is asymptotically tight. As a byproduct of our LP-based techniques, we obtain a randomized offline algorithm that approximates the optimal number of block operations to within a constant factor.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Sequencing and scheduling*

*Research supported by the Centre for Discrete Mathematics and its Applications (DIMAP), University of Warwick, EPSRC award EP/D063191/1, by EPSRC grant EP/F043333/1, and by EPSRC grant EP/G069034/1.

STOC' 12, May 19–22, 2012, New York, New York, USA.

General Terms

Theory, Algorithms

Keywords

Online Algorithms, Online Primal-Dual, Buffer Management, Reordering Buffer, Sorting Buffer

1. INTRODUCTION

We consider a buffer scheduling problem for block operation devices. A stream of items of different types (e.g. colors) have to be processed by a block device. For example the items could be data that has to be written to the hard disc and two items have the same type if they are located in the same block on the hard disc. The block device can process all items with the same type in a single step.

To improve the performance of the whole system a buffer of size k can be used to store items in order to reduce the number of operations required. Whenever the buffer becomes full (i.e., contains k elements) a buffer scheduling strategy has to select one color and then a block operation on all elements with this color that are currently stored in memory is performed. The processed elements are then removed from the buffer and free up space for new elements from the input stream. The goal is to design a scheduling strategy that minimizes the number of block operations required to process all elements from the input sequence.

In this paper our focus lies on the online version of this problem, where the buffer scheduling strategy must make decisions without knowing the future items that appear in the input stream. Our main result is the design of an $O(\log \log k)$-competitive online randomized buffer scheduling strategy.

The problem of minimizing the number of block operations for a buffer is closely related to the *reordering buffer problem* ([22]). The reordering buffer problem has the same basic setup; the only difference is that the processing device does not perform block operations but instead it has different states (one for each color), and it can process items of color c if and only if it is in state c. Both problems give a simple yet versatile framework for modelling real world buffer problems (see e.g., [5, 11, 18, 21, 22] for the reordering buffer problem). It is clear that the number of block operations required in

the block device scheduling problem is at least as large as the number of state changes in the corresponding reordering buffer instance. In fact it could be much larger as a sequence of length n consisting of a single color requires only one state change but $\lceil \frac{n}{k+1} \rceil$ block operations.

The reordering buffer problem is quite well studied in the literature. It was introduced by Räcke et al. [22] who gave an $O(\log^2 k)$-competitive online algorithm. This was improved by Englert and Westermann [16] to $O(\log k)$. Aboud [1] showed that with the proof technique from [16] it is not possible to derive online algorithms with a competitive ratio $o(\log k)$. Nevertheless, Avigdor-Elgrabli and Rabani [5] were able to go beyond the logarithmic threshold by presenting an online algorithm with competitive ratio $O(\log k/\log\log k)$ using linear programming based techniques. This result was later improved by Adamaszek et al. [2] who provide a deterministic algorithm with competitive ratio of $O(\sqrt{\log k})$ and show lower bounds of $\Omega(\sqrt{\log k/\log\log k})$ and $\Omega(\log\log k)$ on the competitive ratio for deterministic and randomized strategies, respectively.

In fact, the above lower bounds on the competitive ratio of a reordering buffer strategy also hold in the model where we have block operations, because, with a minor modification, the sequences considered in [2] do not use the fact that it is possible to process items that were not in the buffer when the corresponding state changes occurred.

Hence, the $O(\log\log k)$-competitive algorithm that we develop in this paper obtains an asymptotically optimal competitive ratio.

1.1 Further related work

The reordering buffer problem has been also studied in the offline setting, and it was shown by Chan et al. [13] and independently by Asahiro et al. [4] that the problem is NP-hard. Until recently no offline approximation algorithms that achieve better bounds than the best online algorithm from [2] have been known. However, in a very new development, Avigdor-Elgrabli and Rabani [6] reported a constant factor offline approximation for the reordering buffer problem. Our techniques also give a constant factor approximation algorithm for the offline version of the block device scheduling problem.

Let us also mention that there have been extensive studies of more general versions of the uniform reordering buffer problem mentioned above. Khandekar and Pandit [19], and Gamzu and Segev [17] considered the problem where the colors correspond to points in a line-metric with appropriately defined costs of changing the colors. Englert et al. [14] studied a more general version where colors correspond to arbitrary points in a metric space (C, d) and the cost for switching from color c' to color c is the distance $d(c', c)$ between the corresponding points in the metric space. Research has also been done on the maximization version of the problem, where the cost measure is the number of color changes that the output sequence saved over the unordered input sequence. For this version there exist constant factor approximation algorithms due to Kohrt and Pruhs [20] and Bar-Yehuda and Laserson [10].

1.2 Our results and techniques

In this paper we present asymptotically tight results for the block device scheduling problem. We present a *randomized online* algorithm that obtains a competitive ratio of $O(\log\log k)$. As a byproduct of our techniques, we obtain a randomized *offline* algorithm that approximates the optimal number of block operations to within a constant factor.

While our problem is closely related to the reordering buffer problem mentioned above, our approach differs significantly from the combinatorial approach used in earlier works for the latter problem [5, 2, 16, 22]. Our techniques are similar to the approach used by Bansal et al. in online algorithms for weighted caching [8] and generalized caching [9], and use also ideas developed for improved generalized caching in [3]. In all these papers one first formulates a packing/covering linear program that forms a relaxation of the original problem. Then one solves the linear program in an *online*-manner using the online primal-dual framework for packing and covering problems introduced by Buchbinder and Naor [12]. In the final step, one gives a randomized rounding algorithm to transform the fractional solution into an integral one. This approach has been shown to be very successful in the study of online caching algorithms and in other related online problems (see, e.g., [7]), and in this paper we demonstrate its power in the buffer management scenario.

We begin in Section 2 by formulating a relaxation of our problem that is a packing/covering linear program with an exponential number of constraints. This formulation uses so-called Knapsack cover inequalities that, for example, have also been used for the generalized caching problem [9]. However, applying the online primal-dual framework directly to this LP would lead to a competitive ratio of $O(\log k)$. Therefore in Section 2 we first modify the LP in such a way that applying the primal-dual technique leads to an $O(\log\log k)$ competitive ratio.

In Section 3 we apply the online primal-dual framework. Although this follows more or less the standard primal-dual technique, it is somewhat non-standard because we apply some of the main arguments not to individual variables but to sums of variables. The rounding is done in Section 4. Here, we present an algorithm that transforms a solution to the LP into a distribution over deterministic buffer scheduling strategies with expected cost at most a constant times larger than the cost of the LP solution. These results combined together yield a randomized online algorithm that obtains a competitive ratio of $O(\log\log k)$.

Since we can show that the linear program (despite its exponential size) can be solved in polynomial time, and since the rounding yields a constant factor approximation of the LP, our approach gives us a polynomial time constant factor approximation randomized *offline* algorithm, as well.

2. LP FORMULATION

Our model is the following. At any point in time the buffer can store k elements. At a time-step $t \geq 1$ a new item appears. If the number of items currently stored in the buffer is at most $k-1$, the new item can simply be added to the buffer. Otherwise, an output operation has to be performed. For this, a buffer-management algorithm selects a color c and all elements among the set of $k+1$ visible elements (i.e. the k elements stored in the buffer plus the new element that arrived in time-step t) that have color c are removed from the buffer. The goal is to minimize the total number of output operations.

For this model we set up a linear program as follows. We have variables $x_c(\tau)$ for every time-step τ and every color c. This variable reflects that we switch to color c at time

τ, and remove all elements of this color. We model the buffer constraint at time t, as follows. Suppose, that we fix a time-step $s_c < t$ for every color c and that we consider only elements of color c that appear after time s_c (s_c can be viewed as a time-step at which we performed an output operation for color c; note that we allow $s_c = 0$ so that the constraint $s_c < t$ can be obtained for every time-step $t \geq 1$).

We use $w_c(s_c, t)$ to denote the number of elements of color c that appear between time s_c and t (excluding s_c but including t). For simplicity, we extend this notation to a vector $\vec{s}$ that represents the chosen time-steps for all colors, i.e., $w_c(\vec{s}, t) := w_c(s_c, t)$. We will also allow $s_c \geq t$, in which case we simply set $w_c(s_c, t) := 0$. The total number of elements that appear until time t (but after their respective time-step s_c) is then $W(\vec{s}, t) := \sum_c w_c(\vec{s}, t)$. Of course, if $W(\vec{s}, t) > k$ the buffer constraint at t is violated, unless we remove some elements.

If we perform an output operation for color c at time $\tau > s_c$ we remove at most $w_c(\vec{s}, \tau)$ elements that appear between time s_c and τ (and if there are no other output operations between s_c and τ we remove exactly $w_c(\vec{s}, \tau)$ elements). Summing this over all colors gives that for every vector $\vec{s}$

$$\sum_c \sum_{\tau \leq t} w_c(\vec{s}, \tau) \cdot x_c(\tau) \geq W(\vec{s}, t) - k$$

is a valid constraint (recall that $w_c(\vec{s}, \tau) = 0$ if $\tau \leq s_c$).

In an integral setting, the constraint still remains valid if we *tighten* it by replacing $w_c(\vec{s}, \tau)$ by $\tilde{w}_c(\vec{s}, \tau, t)$ defined as follows:

$$\tilde{w}_c(\vec{s}, \tau, t) := \max\{\min\{w_c(\vec{s}, \tau), W(\vec{s}, t) - k\}, 0\} \ .$$

Our linear program is the following

$$\begin{aligned} \min \quad & \sum_c \sum_\tau x_c(\tau) \\ \text{s.t.} \quad & \forall \vec{s}, t \text{ with } W(\vec{s}, t) \geq k : \\ & \sum_c \sum_{\tau \leq t} \tilde{w}_c(\vec{s}, \tau, t) \cdot x_c(\tau) \geq W(\vec{s}, t) - k \\ & \forall \tau, c : \\ & x_c(\tau) \geq 0 \end{aligned} \qquad (\mathsf{LP}_k)$$

2.1 Modifying the LP

For solving our LP online, we will use the following slightly modified LP in which we replace k by $k' = (1-\varepsilon) \cdot k \geq 1$, where ε will later be chosen as $1/\log k$.

$$\begin{aligned} \min \quad & \sum_c \sum_\tau x_c(\tau) \\ \text{s.t.} \quad & \forall \vec{s}, t \text{ with } W(\vec{s}, t) \geq k : \\ & \sum_c \sum_{\tau \leq t} \tilde{w}_c(\vec{s}, \tau, t) \cdot x_c(\tau) \geq W(\vec{s}, t) - k' \\ & \forall \tau, c : \\ & x_c(\tau) \geq 0 \end{aligned} \qquad (\mathsf{Primal})$$

Here the above linear program uses the definition:

$$\tilde{w}_c(\vec{s}, \tau, t) := \max\{\min\{w_c(\vec{s}, \tau), W(\vec{s}, t) - k'\}, 0\} \ .$$

This means that LP Primal is obtained by using $\mathsf{LP}_{k'}$ and removing constraints for $\vec{s}, t$ with $k' \leq W(\vec{s}, t) < k$.

We will relate the optimal value of Primal LP to the cost of an optimal algorithm. To this end, we compare the cost of an optimal offline solution utilizing a buffer of size k with the cost of an optimal offline solution utilizing a buffer of size $k' = (1-\varepsilon) \cdot k$. The proof is an adapted and generalized version of Theorem 2.1 in [15].

THEOREM 2.1. *For any input sequence, the cost $\mathsf{OPT}_{k'}$ of an optimal offline solution utilizing a buffer of size $k' = (1-\varepsilon) \cdot k$ is at most a factor of $(2 + \varepsilon \ln k')/(1-\varepsilon)$ larger than the cost OPT_k of an optimal offline solution utilizing a buffer of size k.*

PROOF. Fix any input sequence and consider an optimal offline algorithm using a buffer of size k. Without loss of generality, we may assume that each block operation in the optimum solution processes a different color. Indeed, if two operations were to process the same color, we could re-color the elements processed in one of the operations to a completely new color. This does not change OPT_k and can only increase $\mathsf{OPT}_{k'}$.

We number the colors in the order of the respective block operations, i.e., the i-th block operation processes items of color i and the number of different colors equals OPT_k. Now, we design an offline strategy to process the input sequence with a buffer of size k'.

We call a color *finished* at some time t, if all elements of that color have been processed by our strategy by time t. Every other color is called *unfinished*, and in particular, we use ω to denote the unfinished color that has the smallest index. Let $n(c)$ denote the number of elements of color c stored in the buffer. The potential $\phi(c)$ of an unfinished color c is defined as $(c - \omega + 1) \cdot n(c)$. Note that ω, $n(c)$, and $\phi(c)$ change over time but we do not indicate this, since the time will always be clear from the context.

Now consider the following strategy to process the input sequence with a buffer of size k'. The strategy maintains a counter $p(i)$ for each unfinished color i that is initialized with 0 and gives a lower bound on the number of items with color i or larger that have already been processed.

(1) Let ω be the unfinished color with the smallest index.

(2) If the last element for color ω is either in the buffer or is the next element, perform a block operation for ω.

(3) Otherwise, identify a color q with maximum potential and perform a block operation for q.

    If after this, the last element for ω is still neither in the buffer nor the next element, increase $p(i)$ by $n(q)$ for each i, $\omega \leq i \leq q$.

(4) Repeat.

We start our analysis by giving a bound on the values of the counters $p(i)$.

FACT 2.2. *At any point in time and for any i, $p(i) \leq \varepsilon k$.*

PROOF. For an arbitrary fixed time, we give a bound on $p(\omega)$, where ω is the first unfinished color at that time. This bound implies the claim since, due to the way the counters are increased, $p(i) \geq p(i+1)$ for every $i \geq \omega$.

Let ℓ be the number of elements with a color smaller than ω. Preceding the block operation to process color ω, the optimal solution for buffer size k processed exactly these ℓ

elements. Since the block operation for color ω processes all input elements of color ω and the buffer can only hold k elements, all elements of color ω must appear among the first $\ell + k + 1$ elements of the input sequence.

Assume for contradiction that $p(\omega)$ is increased beyond ϵk in Step (3). This means in particular that not all elements of colour ω have yet appeared. Further, we have processed at least $\lceil \epsilon k \rceil$ elements with a color ω or larger (since $p(\omega) > \epsilon k$), and we have processed all ℓ elements with color less than ω. The buffer stores $\lfloor k' \rfloor = k - \lceil \epsilon k \rceil$ elements, and the next element is not the last element of color ω (as otw. we would not increase $p(\omega)$).

Altogether the first $\ell + \lceil \epsilon k \rceil + k - \lceil \epsilon k \rceil + 1 = \ell + k + 1$ elements do not contain all elements of color ω. This is a contradiction. □

Next, we give a lower bound on the potential $\phi(q)$ that color q has in Step (3) of our strategy.

FACT 2.3. $\phi(q) \geq k'/(1 + \ln k')$.

PROOF. Define $s := k'/(1 + \ln k')$ to shorten notation. Suppose there is no color c with $\phi(c) \geq s$. We know that for every color $c < \omega$, $n(c) = 0$ (these are the finished colors) and for $c \geq \omega - 1 + s$, $n(c) = 0$ (since we assume that every color has potential less than s). However, since in total there are k' elements in the buffer, $\sum_{c=\omega}^{\omega-1+\lfloor s \rfloor} n(c) = k'$. On the other hand, for every c, $(c - \omega + 1) \cdot n(c) = \phi(c) < s$. This gives a contradiction since

$$k' = \sum_{c=\omega}^{\omega-1+\lfloor s \rfloor} n(c) < s \cdot \sum_{c=\omega}^{\omega-1+\lfloor s \rfloor} \frac{1}{c - \omega + 1}$$
$$= s \cdot \sum_{i=1}^{\lfloor s \rfloor} \frac{1}{i} = s \cdot H_{\lfloor s \rfloor} \leq s \cdot (\ln(s) + 1) \leq k' \ ,$$

and the claim follows. □

Putting everything together we get:

- For each color, there is at most one block operation due to Step (2).
- Every block operation in Step (3), except OPT_k many (one for each color), results in an increase of $\sum_i p(i)$ by $k'/(1+\ln k')$ due to Fact 2.3. Since $\sum_i p(i)$ is bounded by $\varepsilon k \cdot \mathsf{OPT}_k$ by Fact 2.2, we conclude that there can be at most $\varepsilon k \cdot \mathsf{OPT}_k \cdot (1+\ln k')/k' = (1+\ln k') \cdot \varepsilon/(1-\varepsilon) \cdot \mathsf{OPT}_k$ such block operations.

In total, we have at most $(2 + (1+\ln k') \cdot \varepsilon/(1-\varepsilon)) \cdot \mathsf{OPT}_k < (2 + \varepsilon \ln k')/(1-\varepsilon) \cdot \mathsf{OPT}_k$ block operations as claimed. □

Using Theorem 2.1, we can easily prove the following lemma.

LEMMA 2.4. *For $\varepsilon = 1/\log k$, the value of LP Primal is at most a constant factor larger than the cost OPT_k of an optimal offline algorithm using a buffer of size k.*

PROOF. By the choice of ε and Theorem 2.1, the value of $\mathsf{LP}_{k'}$ is at most $\mathsf{OPT}_{k'} \leq O(1) \cdot \mathsf{OPT}_k$. LP Primal is obtained from $\mathsf{LP}_{k'}$ by deleting constraints $k' \leq W(\vec{s}, t) < k$. Of course, this can only decrease the cost of the optimum solution. □

3. SOLVING THE LP ONLINE

In this section we show how to solve the linear program Primal from Section 2.1 in an online fashion using the primal dual technique introduced by Buchbinder and Naor [12]. The cost of the online solution will turn out to be at most $O(\log \log k) \cdot \text{opt}(\text{Primal})$, where opt(Primal) denotes the cost of an optimum solution to the LP. In Section 4 we then show how to turn such an online fractional solution into an online algorithm while only increasing the cost by another constant factor.

Recall LP Primal:

$$\begin{aligned} \min \quad & \sum_c \sum_\tau x_c(\tau) \\ \text{s.t.} \quad & \forall \vec{s}, t \text{ with } W(\vec{s}, t) \geq k : \\ & \sum_c \sum_{\tau \leq t} \tilde{w}_c(\vec{s}, \tau, t) \cdot x_c(\tau) \geq W(\vec{s}, t) - k' \\ & \forall \tau, c : \\ & x_c(\tau) \geq 0 \end{aligned} \qquad \text{(Primal)}$$

The dual is the following

$$\begin{aligned} \max \quad & \sum_{(\vec{s},t)} \alpha_{\vec{s},t} (W(\vec{s}, t) - k') \\ \text{s.t.} \quad & \forall \tau, c : \\ & \sum_{(\vec{s},t):t \geq \tau} \tilde{w}_c(\vec{s}, \tau, t) \cdot \alpha_{\vec{s},t} \leq 1 \\ & \forall \vec{s}, t \text{ with } W(\vec{s}, t) \geq k \\ & \alpha_{\vec{s},t} \geq 0 \end{aligned} \qquad \text{(Dual)}$$

Note that the LP Dual has variables $\alpha_{\vec{s},t}$ for $W(\vec{s}, t) \geq k$. For simplicity of notation we define $\alpha_{\vec{s},t} := 0$ for $W(\vec{s}, t) < k$ as this allows to write summations in a simpler form.
The following definition uses a constant scaling factor γ defined in Section 4. For this section we only require $\gamma \geq 2$.

DEFINITION 3.1. *A constraint $(\vec{s}, t)$ of LP Primal is called a* proper *constraint for a variable assignment x if the following holds for every color c:*

a) $\sum_{s_c \leq \tau \leq t} x_c(\tau) \geq 1/\gamma$;

b) $\sum_{s_c < \tau \leq t} x_c(\tau) < 2/\gamma$;

c) $x_c(\tau) < 1/\gamma$ *for all* $\tau \in \{s_c + 1, \ldots, t\}$.

The online algorithm in Section 4 successively generates proper violated constraints and feeds them to Algorithm 1. The algorithm then increases $x_c(\tau)$-values until either the constraint $(\vec{s}, t)$ is satisfied, or not proper anymore. In addition the algorithm from Section 4 also guarantees that the constraints are monotone, i.e., for two consecutive constraints $(\vec{s}_1, t_1)$ and $(\vec{s}_2, t_2)$, $\vec{s}_1 \leq \vec{s}_2$ and $t_1 \leq t_2$. In this section we show that the assignment to LP Primal constructed by calls to Algorithm 1 has cost no more than $O(\log \log k) \cdot \text{opt}(\text{Primal})$.

Given a primal constraint $(\vec{s}, t)$, the algorithm increases the dual variable $\alpha_{\vec{s},t}$ for the constraint by an infinitesimal amount $\mathrm{d}\alpha_{\vec{s},t}$ (Line 15). It also computes for every primal variable $x_\tau(c)$, $\tau \in \{s_c+1, \ldots, t\}$ a value $\Delta(\tau, c)$ that specifies an increase to $x_c(t)$ "wished by variable $x_c(\tau)$" (Lines 7 and 11). Then in Line 13 the $x_c(t)$-value is increased by the maximum of the $\Delta(\tau, c)$-values taken over all τ's.

Procedure 1 fix-constraint$((\vec{s},t),x,\alpha)$

Input: A violated, *proper* constraint $(\vec{s},t)$, current variable assignments x and α.

Output: New assignments for x and α. Return if $(\vec{s},t)$ is satisfied or not *proper* anymore.

1: **while** $(\vec{s},t)$ is proper violated constraint **do**
2: **for** each variable $x_c(\tau)$, $s_c < \tau \leq t$ **do**
3: $\Delta(\tau,c) := 0$
4: **if** $(\sum_{(\vec{s},t):t\geq\tau} \tilde{w}_c(\vec{s},\tau,t)\cdot\alpha_{\vec{s},t} == 1)$ **then**
5: // constraint for (τ,c) is tight
6: **if** $(X_c[\tau,t] < \frac{1}{\log^3 k})$ **then**
7: $\Delta(\tau,c) := \frac{1}{\log^3 k}$
8: $\hat{x}_c(\tau) := \frac{1}{\log^3 k}$
9: **if** $(\sum_{(\vec{s},t):t\geq\tau} \tilde{w}_c(\vec{s},\tau,t)\cdot\alpha_{\vec{s},t} > 1)$ **then**
10: // constraint for (τ,c) is violated
11: $\Delta(\tau,c) := X_c[\tau,t]\cdot\tilde{w}_c(\vec{s},\tau,t)\mathrm{d}\alpha_{\vec{s},t}$
12: $\mathrm{d}x_c(t) := \max_{\tau:s_c<\tau\leq t}\Delta(\tau,c)$
13: $\forall c : x_c(t) := x_c(t) + \mathrm{d}x_c(t)$
14: // increase variable $\alpha_{\vec{s},t}$ by infinitesimal amount
15: $\alpha_{\vec{s},t} = \alpha_{\vec{s},t} + \mathrm{d}\alpha_{\vec{s},t}$

The following analysis is close to the standard analysis using the primal dual scheme. The important part is that we can initialize the $x_c(\tau)$-values with $1/\log^3 k$. For most algorithms that use the online primal-dual scheme this initialization is $\frac{1}{n}$, which results in an LP-solution with approximation guarantee $O(\log n)$. Another aspect that differs from the standard analysis is that we do not apply the standard arguments to individual variables but to sums of variables (namely $X_c[\tau,t] := \sum_{\tau\leq i\leq t} x_c(i)$).

Step 1: show that the solution is approximately feasible for the dual.

Let $v(\tau,c) = \sum_{(\vec{s},t):\tau\leq t}\tilde{w}_c(\vec{s},\tau,t)\alpha_{\vec{s},t} - 1$ denote the amount by which the dual constraint for variable $x_c(\tau)$ is violated. Let $X_c[\tau,t] = \sum_{i=\tau}^{t} x_c(i)$. The violation can increase if the $\alpha_{\vec{s},t}$-multiplier for some primal constraint $(\vec{s},t)$ is increased during Procedure 1. However, note that only primal constraints $(\vec{s},t)$ with $\tau\in\{s_c+1,\ldots,t\}$ will result in an increase of $v(\tau,c)$ as for others $\tilde{w}_c(\vec{s},\tau,t) = 0$. Furthermore, $(\vec{s},t)$ is proper which gives $X_c[s_c+1,t] < 2/\gamma$. Therefore, once a dual constraint (τ,c) fulfills $X_c(\tau,t) \geq 2/\gamma$ its violation $v(\tau,c)$ will not be increased anymore. In the following we analyze how large the violation can become until $X_c[\tau,t] \geq 2/\gamma$. For this we need the following claim.

Claim 3.2. *A violated dual constraint fulfills* $X_c[\tau,t] \geq \frac{1}{\log^3 k}\exp(v(\tau,c))$.

Proof. When the dual constraint (τ,c) becomes tight (i.e., $v(\tau,c) = 0$) the algorithm sets $\Delta(\tau,c) := \frac{1}{\log^3 k}$ in Line 7. This means that at this point (after the end of the iteration) we have

$$X_c[\tau,t] \geq \frac{1}{\log^3 k}\exp(v(\tau,c)) \tag{1}$$

and the claim holds.

During the following iterations a dual variable $\alpha_{\vec{s},t}$ is increased by $\mathrm{d}\alpha_{\vec{s},t}$ (Line 15) which increases the violation $v(\tau,c)$ by $\tilde{w}_c(s_c,\tau,t)\mathrm{d}\alpha_{\vec{s},t}$. This means that the right hand side of Equation 1 increases by

$$\frac{1}{\log^3 k}\exp(v(\tau,c))\cdot\tilde{w}_c(s_c,\tau,t)\mathrm{d}\alpha_{\vec{s},t}$$

However, at the same time $X_c[\tau,t]$ increases by at least

$$X_c[\tau,t]\cdot\tilde{w}_c(\vec{s},\tau,t)\mathrm{d}\alpha_{\vec{s},t} \geq \frac{1}{\log^3 k}\exp(v(\tau,c))\cdot\tilde{w}_c(s_c,\tau,t)\mathrm{d}\alpha_{\vec{s},t}$$

according to Line 11 of the algorithm. □

From the above lemma it follows that the violation of a dual constraint can grow to at most $O(\log\log k)$ until $X_c(\tau,t)$ becomes $2/\gamma$ after which the violation does not increase anymore. Hence, scaling down the dual variables by a factor of $O(\log\log k)$ leads to a feasible dual solution.

The primal cost of the solution (the total value of all $x_c(\tau)$-variables) comes from two sources. There may be an increase of $1/\log^3 k$ if a dual constraint becomes tight (Line 7) or there is the *normal* increase in Line 11. We will split the primal cost into two types according to their source. Let $\hat{C}$ denote the cost induced by increases in Line 7 and C denote cost induced by increases in Line 11. Note that $\hat{C} \leq \sum_{\tau,c}\hat{x}_c(\tau)$ since we set $\hat{x}_c(\tau) = 1/\log^3 k$, whenever we make an increase in Line 7 of the algorithm.

Step 2: show that $\hat{x}_c(\tau)$'s fulfill "primal slackness conditions.

We show that $\hat{x}_c(\tau) > 0 \Rightarrow \sum_{(\vec{s},t):t\geq\tau}\tilde{w}_c(\vec{s},\tau,t)\cdot\alpha_{\vec{s},t} \geq 1$. This is immediate as the algorithm only assigns $\hat{x}_c(\tau)$ a non-zero value if the dual constraint corresponding to variable $x_c(\tau)$ is tight. Once a constraint is tight in the online algorithm it means that in the end the constraint will either be tight or violated.

Step 3: show that $\hat{x}_c(\tau)$'s fulfill "dual slackness conditions".

Here we show $\alpha_{\vec{s},t} > 0 \Rightarrow \sum_c\sum_{\tau\leq t}\tilde{w}_c(\vec{s},\tau,t)\cdot\hat{x}_c(\tau) \leq W(\vec{s},t) - k'$. Fix a color c. We partition the time-steps $\tau\in\{s_c+1,\ldots,t\}$ into classes according to the value of $w_c(s_c,\tau)$, The class $T_{j,c}$, $j\geq 0$ contains the time-steps for which $w_c(s_c,\tau)\in[2^j,2^{j+1})$.

Claim 3.3. *A class $T_{j,c}$ includes at most $O(\log\log k)$ time-steps τ that have $\hat{x}_c(\tau) > 0$.*

Proof. Since $\alpha_{\vec{s},t} > 0$ the constraint $(\vec{s},t)$ has been fed to Procedure 1 at some point. We first analyze how many time-steps τ with $\hat{x}_c(\tau) > 0$ can have been created before Procedure 1 was called with constraint $(\vec{s},t)$. Because of the monotonicity of constraints we know that all constraints $(\vec{s}',t')$ among these that have led to an increase in $\hat{x}_c(\tau)$-values fulfill $\vec{s}' \leq \vec{s}$ and $t' \leq t$.

Fix a class $T_{j,c}$ and let $\tau_1,\tau_2,\ldots$ denote its time-steps that have $\hat{x}_c(\tau_i) > 0$ in increasing order. Consider a time-step τ_i. When the algorithm sets $\hat{x}_c(\tau_i)$ to $1/\log^3 k$ it must be at a time $t_i < \tau_{i+1}$. The reason is that setting $\hat{x}_c(\tau_i) = 1/\log^3 k$ also triggers an increase of $1/\log^3 k$ to $x_c(t_i)$. If $\tau_{i+1}\leq t_i$ this high value of $x_c(t_i)$ would prevent $\hat{x}(\tau_{i+1})$ from being increased as this only happens if $X_c[\tau_{i+1},t_{i+1}] < 1/\log^3 k$.

Define $L(\tau_i) := \sum_{(\vec{s},t):\tau_i \le t} \tilde{w}_c(\vec{s},\tau_i,t)\cdot\alpha_{\vec{s},t}$ to be the *load* of τ_i (the left hand side of the constraint (τ_i,c) in the dual). Note that an increase of $L(\tau_i)$, $i>1$ by $\tilde{w}(s'_c,\tau_i,t')\mathrm{d}\alpha_{\vec{s}',t'}$ in Line 15 is connected with an increase of $L(\tau_1)$ by $\tilde{w}(s'_c,\tau_1,t')\mathrm{d}\alpha_{\vec{s}',t'}$ which is approximately the same up to a constant factor as τ_1 and τ_i are in the same class (note that if τ_1 and τ_2 are in the same class w.r.t. $\vec{s}$ they are also in the same class w.r.t. $\vec{s}'$). In order for a τ_i to set its $\hat{x}_c(\tau_i)$-value, it first needs to collect a load of 1 as its constraint needs to be tight. Consequently, for every τ_i with $\hat{x}_c(\tau_i)>0$, the load $L(\tau_1)$ will increase by at least $1/2$. Since the dual solution is almost feasible the load $L(\tau_1)$ is at most $O(\log\log k)$. This gives that before constraint $(\vec{s},t)$ at most $O(\log\log k)$ time-steps $\tau\in\{s_c+1,\dots,t\}$ could achieve $\hat{x}_c(\tau)>0$.

After this at most one time-step within $\{s_c+1,\dots,t\}$ can have its $x_c(\tau)$-value increased. This holds because after the next increase (at time $t''\ge t$) $x_c(t'')$ will be at least $1/\log^3 k$. Again this prevents any $\hat{x}_c(\tau)$-value with $\tau\le t''$ from being increased. □

Let $n_{j,c}$ denote the number of time-steps τ in class $T_{j,c}$ that have $\hat{x}_c(\tau)>0$. Then we have

$$\begin{aligned}\sum_c\sum_{\tau\le t}\tilde{w}_c(\vec{s},\tau,t)\cdot\hat{x}_c(\tau) &\le \sum_c\sum_{j>0} 2^{j+1} n_{j,c}\tfrac{1}{\log^3 k}\\ &\le \sum_c O(\log\log k)\cdot w_c(s_c,t)\tfrac{1}{\log^3 k}\\ &\le W(\vec{s},t)/\log k \le W(\vec{s},t)-k' \ .\end{aligned}$$

Note that the last inequality follows from the transformation of our LP in Section 2, and is the key for obtaining a competitive ratio of $O(\log\log k)$.

Step 4: conclude that cost of $\hat{x}_c(\tau)$'s is not too large.

From the slackness conditions we derive a bound on the cost $\hat{C}$ as follows.

$$\begin{aligned}\textstyle\sum_c\sum_\tau \hat{x}_c(\tau) &\le \sum_c\sum_\tau\Big[\sum_{(\vec{s},t):t\ge\tau}\tilde{w}_c(\vec{s},\tau,t)\cdot\alpha_{\vec{s},t}\Big]\cdot\hat{x}_c(\tau)\\ &\le \sum_{(\vec{s},t)}\alpha_{\vec{s},t}\sum_c\sum_{\tau\le t}\tilde{w}_c(\vec{s},\tau,t)\cdot\hat{x}_c(\tau)\\ &\le \sum_{(\vec{s},t)}\alpha_{\vec{s},t}\cdot\big(W(\vec{s},t)-k'\big) \ .\end{aligned}$$

The last term is the profit of an approximately feasible dual solution. Hence, it is at most an $O(\log\log k)$-factor larger than the cost of an optimal primal solution.

Step 5: show that C is small.

The cost C that is created in Line 11 and Line 12 can be estimated as follows. In one iteration the dual profit increases by $(W(\vec{s},t)-k')\cdot\mathrm{d}\alpha_{\vec{s},t}$. We now estimate the increase of C during the same iteration. Fix a color c. In Line 11 every $\tau\in\{s_c+1,\dots,t\}$ makes a wish for an increase of $x_c(t)$ of $X_c[\tau,t]\cdot\tilde{w}_c(\vec{s},\tau,t)\mathrm{d}\alpha_{\vec{s},t}$. In Line 12 the maximum is taken. This means that

$$\begin{aligned}\mathrm{d}x_c(t) &= \max_\tau X_c[\tau,t]\cdot\tilde{w}_c(\vec{s},\tau,t)\mathrm{d}\alpha_{\vec{s},t}\\ &\le \sum_\tau \tilde{w}_c(\vec{s},\tau,t)\cdot x_c(\tau)\cdot\mathrm{d}\alpha_{\vec{s},t} \ .\end{aligned}$$

Summing this over all colors c gives that the increase in primal cost is at most

$$\sum_\tau\sum_c \tilde{w}_c(\vec{s},\tau,t)x_c(\tau)\cdot\mathrm{d}\alpha_{\vec{s},t} \le (W(\vec{s},t)-k')\cdot\mathrm{d}\alpha_{\vec{s},t} \ .$$

Here the last inequality follows since the constraint $(\vec{s},t)$ is violated. Hence, the increase in the cost C is at most the increase in dual profit of an approximately feasible dual solution, which is at most $O(\log\log k)$ times the cost of an optimal primal solution.

Algorithm 2 RANDOM-STEP(t)

```
// buffer constraints may be violated due to new item
if (s⃗,t) not proper then recompute s⃗
while constraint (s⃗,t) is violated do
    fix-constraint((s⃗,t),x,α) // change current solution
    adjust distribution μ to reflect new x_c(τ)-values
    if (s⃗,t) not proper then recompute s⃗
// buffer constraints are fulfilled
```

4. ROUNDING THE LP-SOLUTION IN AN ONLINE MANNER

The randomized algorithm RANDOM for the buffer management problem uses an "LP-solver" for obtaining assignments to LP Primal and LP Dual with low primal cost and then uses these solutions to generate a distribution μ over deterministic buffer management strategies with low expected cost.

Algorithm 2 shows the general structure of one step for RANDOM. RANDOM maintains a vector $\vec{s}$ and together with the current time-step t the pair $(\vec{s},t)$ forms a primal constraint of the LP that RANDOM tries to satisfy. For this, RANDOM feeds the current primal and dual assignments together with $(\vec{s},t)$ to an "LP-routine". This sub-routine could e.g. be Algorithm 1 from Section 3. However, it could also be a routine that is simply based on knowing the optimal LP-solution (for the design of offline approximation algorithms we can assume that we know the optimum LP-solution as we have a separation oracle (see Section A)).

The LP-subroutine increases some primal variables $x_c(\tau)$ with $s_c<\tau\le t$ until the constraint $(\vec{s},t)$ is either satisfied or the constraint $(\vec{s},t)$ is not proper anymore. Then, RANDOM adjusts the distribution over buffer management strategies to reflect the changed variables $x_c(\tau)$ in the primal LP. In case the LP-routine returned because $(\vec{s},t)$ is not proper, RANDOM computes a new vector $\vec{s}$ and repeats the process.

When finally the constraint $(\vec{s},t)$ is satisfied all deterministic buffer management strategies in the distribution μ fulfill their buffer-constraint (for time t). Observe that the assignment for the primal LP may not be feasible as RANDOM only feeds some constraints to the LP-routine. Further, note that the assignment to the dual LP is not used by RANDOM at all (but may be used by the LP-routine).

A deterministic buffer management strategy is given by a function $D: T\times C\to\mathbb{N}_0$ with the meaning that $D(\tau,c)$ describes the number of output operations for color c performed right after the τ-th item appeared. We allow $D(\tau,c)>1$, i.e., we will allow that the strategy performs several consecutive output operations for the same color. We show that each strategy in the support of μ fulfills the buffer-constraints and that the cost for updating the distribution is only $O(\text{primal-cost})$, where primal-cost is the cost of the primal assignment generated by the calls to the LP-routine.

This means that if the LP-routine is the procedure from Section 3 we obtain an online algorithm with competitive ratio $O(\log\log k)$. If we implement it using the optimal LP-solution we obtain an offline approximation algorithm with a constant approximation factor.

In order to show a bound on the update cost, we show that any change to the LP that increases the LP-cost by ϵ results in a change to the distribution over buffer management strategies with expected cost $O(\epsilon)$.

4.1 Ensuring Buffer Constraints

Suppose that RANDOM just finished its step at time t (Algorithm 2), and that $\vec{s}$ is the vector maintained by RANDOM. RANDOM will always guarantee that $\vec{s}$, and the assignment x fulfill $1/\gamma \leq \sum_{\tau\in\{s_c+1,\dots,t\}} x_c(\tau) < 3/\gamma$ and $x_c(\tau) \leq 1/\gamma$, and that $\vec{s}$ is monotonically increasing (the latter property is only important for the implementation of the LP-routine in Section 3 that assumes that the constraints that are fed to the routine have this property). We will use the fact that the constraint $(\vec{s}, t)$ is fulfilled to show that the strategies in the support of μ fulfill the buffer-constraints.

We first set up some properties that the strategies in μ have to fulfill. The fact that buffer-constraints hold will follow from these properties. In order to define these properties we introduce the following notation. Let γ denote a scaling factor to be determined later. We use $z_c(\tau) := \max\{\gamma x_c(\tau), 1\}$ to denote a scaled version of the current assignment to LP Primal. We partition the pairs (τ, c) with $\tau \in \{s_c+1,\dots,t\}$ into classes according to the value of $w_c(s_c,\tau)$. We say a pair (τ, c) is in class S_i if $w_c(s_c,\tau) \in [2^i, 2^{i+1})$ (we do not care about (τ, c)-pairs that have $w_c(s_c,\tau) = 0$). We further use S^c to denote the set $\{(\tau, c) \mid \tau \in \{s_c+1,\dots,t\}\}$ and $S = \bigcup_c S^c$.

In addition to sets S_i we also define a set L that contains (τ, c)-pairs with a "large" $w_c(\tau, c)$-value. Formally, we first select pairs in decreasing order of $w_c(\tau, c)$-value until L contains pairs whose $z_c(\tau)$-values sum up to at least $\lambda + 1$ (for a parameter $\lambda \ll \gamma$ to be chosen later) or $L = S$; then if the $z_c(\tau)$-values sum up to more than $\lambda + 1$ we remove the last element added. Hence, if $L \neq S$ we have $\lambda \leq \sum_{(\tau,c)\in L} z_c(\tau) \leq \lambda + 1$, as the $z_c(\tau)$'s are at most 1. A buffer management strategy D has to fulfill the following properties.

1. For every color c, $\sum_{s_c\leq\tau\leq t} D(\tau, c) \geq 1$, i.e., D removed the color at least once between time s_c and t.
2. D mirrors the fractional solution of the LP on the sets S_i:
$$\forall S_i: \quad \left\lfloor \sum_{(\tau,c)\in S_i} z_c(\tau) \right\rfloor \leq \sum_{(\tau,c)\in S_i} D(\tau, c) \ .$$
3. D mirrors the fractional solution of the LP on the set L of *large* (τ, c)-pairs:
$$\left\lfloor \sum_{(\tau,c)\in L} z_c(\tau) \right\rfloor \leq \sum_{(\tau,c)\in L} D(\tau, c) \ .$$
4. For every color c and for every class S_i, we have $\sum_{(\tau,c)\in S_i\cap S^c} D(\tau, c) \leq 9$, this means D did not remove the same color very often in the same class.

We first show that a buffer management strategy D that fulfills the above properties also fulfills buffer-constraints provided that the constraint for $(\vec{s}, t)$ in the primal LP is satisfied. Property 1 guarantees that elements/items of color c that appeared at time s_c or before do not influence the buffer-constraint for D at time t, since all these items have already been removed. Hence, the following formula specifies exactly the number of items in D's buffer at time t:

$$\text{buffer}(t) = W(\vec{s}, t) - \sum_c \max_{\tau: D(\tau,c)=1} w_c(s_c, \tau) \ . \qquad (2)$$

This holds because the term $w_c(s_c, \tau)$ (for the maximum τ with $D(\tau, c) = 1$) specifies the items that appeared after time s_c (excluding s_c) and are evicted at time τ or before. Let j denote the index of the largest class that contains a pair (τ, c) with $D(\tau, c) = 1$. Then

$$\begin{aligned}\max_{\tau: D(\tau,c)=1} w_c(s_c, \tau) \geq 2^j &\geq \frac{1}{36}\sum_{i=0}^{j} 9\cdot 2^{i+1} \\ &\geq \frac{1}{36}\sum_{(\tau,c)\in S^c} D(\tau, c)\cdot w_c(s_c, \tau) \ ,\end{aligned}$$

where the last step uses Property 4 (the fact that D does not evict a color too often in the same class). Plugging the above into Equation 2 yields $\text{buffer}(t) \leq W(\vec{s}, t) - \frac{1}{36}\sum_{(\tau,c)\in S} D(\tau, c)\cdot w_c(s_c, \tau)$. We need to show that this is at most k. This means we want to show that $\sum_{(\tau,c)\in S} D(\tau, c)\cdot w_c(s_c, \tau) \geq 36(W(\vec{s}, t) - k)$.

This is encapsulated in the following lemma.

LEMMA 4.1. *Let $(\vec{s}, t)$ be a proper, satisfied constraint. A buffer scheduling strategy that fulfills property 1, 2, 3, or 4 fulfills $\sum_{(\tau,c)\in S} D(\tau, c)\cdot w_c(s_c, \tau) \geq 36(W(\vec{s}, t) - k)$.*

PROOF. For this we need the following claim.

CLAIM 4.2. *Either we have $\lambda \leq \sum_{(\tau,c)\in L} z_c(\tau) \leq (\lambda + 1)$ or $W(\vec{s}, t) \leq k$.*

PROOF. The constraint for $(\vec{s}, t)$ in LP Primal is fulfilled. Therefore,

$$\sum_{s_c<\tau\leq t} \tilde{w}_c(s_c, \tau, t)\cdot x_c(\tau) \geq W(\vec{s}, t) - k \ .$$

If $W(\vec{s}, t) - k \geq 0$ then $\tilde{w}_c(s_c, \tau, t) \leq w(\vec{s}, t)$. Therefore, the $x_c(\tau)$ must sum to at least one. Scaling by γ gives that the $z_c(\tau)$'s sum up to at least γ. Hence, L will contain a $z_c(\tau)$-weight of at least λ and at most $\lambda + 1$, as $\lambda \ll \gamma$. □

In order to show the lemma we can assume that $W(\vec{s}, t) - k > 0$ as otherwise the equation trivially holds as the left hand side is always positive. The above claim then gives $\sum_{(\tau,c)\in L} z_c(\tau) \geq \lambda$.

Now assume that i_ℓ, the class-index of the pair $(\tau, c) \in L$ with smallest $w_c(s_c, \tau)$-value, fulfills $2^{i_\ell} \geq W(\vec{s}, t) - k$. Then

$$\begin{aligned}\sum_{(\tau,c)\in S} w_c(s_c, \tau)\cdot D(\tau, c) &\geq \sum_{(\tau,c)\in L} w_c(s_c, \tau)\cdot D(\tau, c) \\ &\geq 2^{i_\ell}\sum_{(\tau,c)\in L} D(\tau, c) \\ &\geq \lambda(W(\vec{s}, t) - k) \\ &\geq 36(W(\vec{s}, t) - k) \ ,\end{aligned}$$

by choosing $\lambda \geq 36$. In the following we can assume $2^{i_\ell} \leq$

$(W(\vec{s},t)-k)$. We have

$$\begin{aligned}
&\sum_{(\tau,c)\in S} w_c(s_c,\tau)D(\tau,c)\\
&\quad\geq \sum_{i\leq i_\ell}\sum_{(\tau,c)\in S_i} w_c(s_c,\tau)D(\tau,c)\\
&\quad\geq \sum_{i\leq i_\ell} 2^i \sum_{(\tau,c)\in S_i} D(\tau,c)\\
&\quad\geq \sum_{i\leq i_\ell} 2^i\Big(\sum_{(\tau,c)\in S_i} z_c(\tau)-1\Big)\\
&\quad= \frac{1}{2}\sum_{i\leq i_\ell}\sum_{(\tau,c)\in S_i} 2^{i+1}z_c(\tau)-\sum_{i\leq i_\ell}2^i\\
&\quad\geq \frac{1}{2}\sum_{i\leq i_\ell}\sum_{(\tau,c)\in S_i} w_c(s_c,\tau)z_c(\tau)-2^{i_\ell+1}\\
&\quad\geq \frac{\gamma}{2}\sum_{i\leq i_\ell}\sum_{(\tau,c)\in S_i} \tilde{w}_c(s_c,\tau,t)x_c(\tau)-2(W(\vec{s},t)-k)\ .
\end{aligned}\tag{3}$$

Then

$$\begin{aligned}
\frac{\gamma}{2}\sum_{L\setminus S_{i_\ell}} \tilde{w}_c(s_c,\tau,t)x_c(\tau) &\leq \frac{1}{2}\sum_{(\tau,c)\in L}\tilde{w}_c(s_c,\tau,t)z_c(\tau)\\
&\leq (W(\vec{s},t)-k)\sum_{(\tau,c)\in L} z_c(\tau)\\
&\leq (\lambda+1)(W(S)-k)
\end{aligned}$$

where the last inequality follows from $\sum_{(\tau,c)\in L}\leq\lambda+1$. Adding the inequality

$$0\geq \frac{\gamma}{2}\sum_{L\setminus S_{i_\ell}}\tilde{w}_c(s_c,\tau,t)x_c(\tau)-(\lambda+1)(W\vec{(}s),t-k)$$

to Equation 3 gives

$$\begin{aligned}
&\sum_{(\tau,c)\in S} w_c(s_c,\tau)D(\tau,c)\\
&\quad\geq \frac{\gamma}{2}\sum_{(\tau,c)\in S}\tilde{w}_c(s_c,\tau,t)x_c(\tau)-(\lambda+2)(W(\vec{s},t)-k)\\
&\quad\geq 36(W(\vec{s},t)-k)\ ,
\end{aligned}$$

where the last inequality holds for large enough γ, and uses the fact that the constraint for $(\vec{s},t)$ in LP Primal is fulfilled. □

5. UPDATING THE DISTRIBUTION

It remains to show how to update the distribution in an online manner so that the buffer management strategies fulfill Properties 1, 2, 3, and 4.

Instead of constructing μ directly we will first construct distributions μ_1,μ_2,μ_3. A random buffer management strategy according to μ is then chosen by sampling strategies $\mu_1\sim D_1$, $\mu_2\sim D_2$, $\mu_3\sim D_3$ and computing the strategy D by setting $D(\tau,c):=\max\{D_1(\tau,c),D_2(\tau,c),D_3(\tau,c)\}$. Any strategy, whether in the support of μ_1, μ_2, or μ_3 fulfills $\forall c,\forall S_i\sum_{(\tau,c)\in S_i\cap S^c}D(\tau,c)\leq 3$. Hence, the strategy $D=\max\{D_1,D_2,D_3\}$ fulfills $\forall c,\forall S_i\sum_{(\tau,c)\in S_i}D(\tau,c)\leq 9$ (Property 4).

In addition strategies in the support of μ_1 fulfill Property 1, strategies from μ_2 fulfill Property 2, and strategies from μ_3 fulfill Property 3. Hence, D will fulfill all these properties and consequently D will obey the buffer-constraints.

5.1 Maintaining μ_1, μ_2, and μ_3

In the following we describe how to update μ_1, μ_2, and μ_3, and make sure that the strategies in their support fulfill their respective properties. The distributions μ_1 and μ_2 will always fulfill $\sum_D\mu_i(D)\cdot D(\tau,c)=z_c(\tau)$, i.e., the expected number of times the color c is removed at time τ by a random strategy D is equal to the scaled LP-variable $z_c(\tau)$ (recall that we allow a strategy to remove a color several times in the same step). It would be difficult to maintain the above property without allowing changes in the past. Therefore, we will allow a strategy at time t to alter its past behavior and to increase or decrease $D(\tau,c)$ for $\tau<t$ (such a change may incur a cost, of course). In reality, decreasing a $D(\tau,c)$-value only makes fulfilling buffer-constraints more difficult, so allowing this does not give additional power to the algorithm. Similarly, increasing a $D(\tau,c)$-value with $\tau<t$ is never better than increasing $D(t,c)$ instead, since we only care about the buffer-constraint at t as the others have already been met.

There are two types of changes that require updating the distributions. Firstly, the $z_c(\tau)$-values may increase due to executing the LP-subroutine. We will assume that these changes are infinitesimal, i.e., in each step we have to react to an increase of a $z_c(\tau)$-value from $z_c(\tau)$ to $z_c(\tau)+\epsilon$.

The other type of change is a change to the vector $\vec{s}$ triggered in Line 2 or Line 6 of Algorithm 2. The recomputation of $\vec{s}$ is done as follows. Let c denote a color that either has a $\bar{\tau}\in\{s_c+1,\dots,t\}$ with $x_c(\bar{\tau})\geq 1/\lambda$ or has $\sum_{s_c<\tau\leq t}x_c(\tau)\geq 2/\gamma$. In the first case we set $s_c'=\bar{\tau}$. In the second case we compute the largest s such that $\sum_{s\leq\tau\leq t}x_c(\tau)\geq\frac{1}{\gamma}$. Then we set s_c' to this value. We do this for all colors independently. The above procedure guarantees that the following properties *always* hold

- $\sum_{(\tau,c):s_c<\tau\leq t}z_c(\tau)<3$. A constraint $(\vec{s},t)$ is only proper if $\sum_{(\tau,c):s_c<\tau\leq t}z_\tau(c)\leq 2$. When feeding such a constraint to Algorithm 1 from Section 3, the z-values are at most increased by $\gamma/\log^3 k$ within a single step. For large enough k this means that the above sum never exceeds 3.

- $\sum_{s_c\leq\tau\leq t}z_c(\tau)\geq 1$ In the end we want that every strategy D from μ has every color removed at least once within the range $\{s_c,\dots,t\}$. Therefore this property is important.

Whenever $\vec{s}$ changes $\sum_{(\tau,c):s_c<\tau\leq t}z_c(\tau)$ decreases by at least 1. Therefore a fixed pair (τ,c) is only involved in at most 3 different constraints $(\vec{s},t)$.

For increasing $z_c(\tau)$-values we show that an increase in ϵ results in an expected cost of $O(\epsilon)$ for the maintenance operation. We also implement a maintenance operation with $O(\epsilon)$ expected cost when a $z_c(\tau)$-value decreases. With this decrement operation we implement a change of the vector $\vec{s}$ as follows.

Suppose we want to change the c-th coordinate of $\vec{s}$ from s_c to s_c'. This may make all properties that depend on the values $w_c(s_c,\tau)$ invalid. We decrement all $z_c(\tau)$-values with $\tau\in\{s_c'+1,\dots,t\}$ to 0. Then we change s_c to s_c', and after that we increase the $z_c(\tau)$-values again. The cost for this is $O(z_c(\tau))$ for every (τ,c)-pair involved. Since each (τ,c)-pair is only involved in a constant number of these decrement

operations we obtain that the total cost for the change is only $O(\sum_{\tau,c} z_c(\tau)) = O(\text{primal cost})$, provided that we can implement the increase and decrease operations as claimed.

Maintaining μ_1

For maintaining μ_1 we do not require a decrement operation as Property 1 does not depend on $w_c(s_c, \tau)$-values. Suppose that $z_c(\tau)$ increases by ϵ. Then we first identify an ϵ-measure of strategies that have the smallest value of $\arg\max_{\tau'}\{D(\tau', c) > 0\}$, i.e., strategies that did not remove color c for a long time. For these strategies we set $D(\tau, c) = 1$. This means that the strategies evict color c in a round-robin fashion. Since $\sum_{(\tau,c)\in S^c}\sum_D \mu(D)D(\tau, c) = \sum_{(\tau,c)\in S^c} z_\tau(c) \geq 1$, Property 1 follows.

Maintaining μ_2

We maintain a *strengthening* of Property 2, namely for all S_i:

$$\left\lfloor \sum_{(\tau,c)\in S_i} z_c(\tau) \right\rfloor \leq \sum_{(\tau,c)\in S_i} D(\tau, c) \leq \left\lceil \sum_{(\tau,c)\in S_i} z_c(\tau) \right\rceil . \tag{4}$$

Suppose that the $z_{\bar{c}}(\bar{\tau})$ value for some pair $(\bar{\tau}, \bar{c})$ is increased by ϵ and assume that $(\bar{\tau}, \bar{c}) \in S_i$. As we want to satisfy $\sum_D \mu_1(D)D(\bar{\tau}, \bar{c}) = z_{\bar{c}}(\bar{\tau})$ we have to increase $D(\bar{\tau}, \bar{c})$ for some strategies. For this we choose an ϵ-fraction of strategies that have $\sum_{(\tau,c)\in S^c\cap S_i} D(\tau, c) \leq 2$ (these must exist for small enough ϵ as $\sum_D \sum_{(\tau,c)\in S^c\cap S_i} D(\tau, c)\mu_1(D) = \sum_{(\tau,c)\in S^c\cap S_i} z_c(\tau) \leq \sum_{(\tau,c)\in S^c} z_c(\tau) < 3$). We increase the value of $D(\tau, c)$ for the chosen strategies.

Now the constraint in Equation 4 may be violated for class S_i. Let $a = \lfloor\sum_{(\tau,c)\in S_i} z_c(\tau)\rfloor$ (before changing $z_c(\tau)$) and first assume that $\lfloor\sum_{(\tau,c)\in S_i} z_c(\tau)\rfloor$ remains equal to a. Then the strategies that just have been changed may now have $\sum_{(\tau,c)\in S_i} D(\tau, c) = a + 2$, which is not allowed.

In order to fix this we match these strategies to strategies that fulfill $\sum_{(\tau,c)\in S_i} D(\tau, c) = a$. For each strategy D there must exist a (τ, c) such that $D(\tau, c) > D'(\tau, c)$, where D' denotes the strategy that D is matched to. We decrease $D(\tau, c)$ by 1 and increase $D'(\tau, c)$ by 1. This only induces an expected cost of $O(\epsilon)$. The case in which $\lfloor\sum_{(\tau,c)\in S_i} z_c(\tau)\rfloor$ changes is analogous.

Also the decrement operation can be implemented this way. When $z_c(\tau)$ decreases we select an ϵ-measure of strategies that fulfill $D(\tau, c) > 0$ and decrease $D(\tau, c)$ for them. Then we do a re-balancing step.

Maintaining μ_3

Here we maintain a strengthened version of Property 3. Let $(\tau_1, c_1), (\tau_2, c_2), \ldots$ denote the sequence of (τ, c)-pairs from S, in decreasing order of $w_c(s_c, \tau)$. Let (τ_r, c_r) denote the first pair in this sequence that is *not* in L (note that this may not exist; then we define r as $|S| + 1$ since $L = S$). Define a function ℓ on the (τ, c)-pairs that is zero for all (τ_i, c_i), $i > r$; one for all (τ_i, c_i), $i < r$ and $z_{c_r} - (\sum_{i=1}^{r} z_{c_i}(\tau_i) - (\lambda + 1))/z_{c_r}(\tau_r)$ for (τ_r, c_r). For all (τ_i, c_i), $i \neq r$ ℓ simply is the characteristic function of the set L; only for r it measures the fraction by which (τ_r, c_r) needs to be included into L in order that the $z_c(\tau)$-values in L sum up to *exactly* $(1 + \lambda)$ (recall that during the construction of L we were aiming for it to contain a total z-weight of $\lambda + 1$. When we overshoot we remove the last element).

We maintain the constraint that

$$\left\lfloor \sum_{(\tau,c)\in S} z_c(\tau)\ell(\tau, c) \right\rfloor \leq \sum_{(\tau,c)\in L} D(\tau, c) \leq \left\lceil \sum_{(\tau,c)\in S} z_c(\tau)\ell(\tau, c) \right\rceil ,$$

which is a strengthening of Property 3. In addition we maintain $\sum_D D(\tau, c)\mu_3(D) = z_c(\tau)\ell(\tau, c)$.

Suppose that a $z_c(\tau)$-value increases and that $(\tau, c) \in L$ (otherwise we don't need to do anything; observe that if z_{c_r} is increased or decreased by ϵ the value of $z_{c_r}(\tau_r)\ell(\tau_r, c_r)$ stays constant). We increase $D(\tau, c)$ for an ϵ-measure of strategies that have $D(\tau, c) < 3$. In addition we decrease $D(\tau_r, c_r)$ for an ϵ-measure of strategies (only if r is defined, i.e., if $L \neq S$).

Now, there may be an ϵ-fraction of strategies that has a value of $\sum_{(\tau,c)\in L} D(\tau, c)$ that is too large by 1, and there may exist an ϵ-measure of strategies for which this value is by one too low. As before we can perform re-balancing steps in order to fix this at an expected cost of $O(\epsilon)$.

6. REFERENCES

[1] Amjad Aboud. Correlation clustering with penalties and approximating the reordering buffer management problem. *Master's thesis, Computer Science Department, The Technion — Israel Institute of Technology*, 2008.

[2] Anna Adamaszek, Artur Czumaj, Matthias Englert, and Harald Räcke. Almost tight bounds for reordering buffer management. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC)*, pages 607–616, 2011.

[3] Anna Adamaszek, Artur Czumaj, Matthias Englert, and Harald Räcke. An $O(\log k)$-competitive algorithm for generalized caching. In *Proceedings of the 23rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1681–1689, 2012.

[4] Yuichi Asahiro, Kenichi Kawahara, and Eiji Miyano. NP-hardness of the sorting buffer problem on the uniform metric. Unpublished manuscript, 2010.

[5] 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.

[6] Noa Avigdor-Elgrabli and Yuval Rabani. A constant factor approximation algorithm for reordering buffer management. arXiv:1202.4504, 2012.

[7] Nikhil Bansal, Niv Buchbinder, Aleksander Mądry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. In *Proceedings of the 52nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 267–276, 2011.

[8] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. In *Proceedings of the 48th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 507–517, 2007.

[9] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. Randomized competitive algorithms for generalized caching. In *Proceedings of the 30th ACM Symposium on Theory of Computing (STOC)*, pages 235–244, 2008.

[10] Reuven Bar-Yehuda and Jonathan Laserson. Exploiting locality: Approximating sorting buffers. In *Proceedings of the 3rd Workshop on Approximation and Online Algorithms (WAOA)*, pages 69–81, 2005.

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

[12] Niv Buchbinder and Joseph Naor. Online primal-dual algorithms for covering and packing problems. In *Proceedings of the 13th European Symposium on Algorithms (ESA)*, pages 689–701, 2005.

[13] Ho-Leung Chan, Nicole Megow, Rob van Stee, and René Sitters. The sorting buffer problem is NP-hard. *CoRR*, abs/1009.4355, 2010.

[14] Matthias Englert, Harald Räcke, and Matthias Westermann. Reordering buffers for general metric spaces. In *Proceedings of the 39th ACM Symposium on Theory of Computing (STOC)*, pages 556–564, 2007.

[15] Matthias Englert, Heiko Röglin, and Matthias Westermann. Evaluation of online strategies for reordering buffers. *ACM Journal of Experimental Algorithmics*, 14:3:3.3–3:3.14, 2009.

[16] 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.

[17] Iftah Gamzu and Danny Segev. Improved online algorithms for the sorting buffer problem. In *Proceedings of the 24th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 658–669, 2007.

[18] 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.

[19] Rohit Khandekar and Vinayaka Pandit. Online and offline algorithms for the sorting buffers problem on the line metric. *Journal of Discrete Algorithms*, 8(1):24–35, 2010.

[20] Jens S. Kohrt and Kirk Pruhs. A constant factor approximation algorithm for sorting buffers. In *Proceedings of the 6th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 193–202, 2004.

[21] Jens Krokowski, Harald Räcke, Christian Sohler, and Matthias Westermann. Reducing state changes with a pipeline buffer. In *Proceedings of the 9th International Fall Workshop Vision, Modeling, and Visualization (VMV)*, pages 217–224, 2004.

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

APPENDIX

A. FINDING A VIOLATED CONSTRAINT

We want to find a vector $\vec{s}$ minimizing

$$\sum_c \sum_{\tau \le t} \tilde{w}_c(\vec{s}, \tau, t) \cdot x_c(\tau) - (W(\vec{s}, t) - k) \ .$$

If the minimum is smaller than 0, it means that we found a violated constraint, if the minimum is at least 0, all constraints are satisfied.

First, notice that the value $W(\vec{s}, t)$ is an integer between k and n. Therefore it is enough to find for each $\ell \in \{k, \dots, n\}$ a vector $\vec{s}$ such that $W(\vec{s}, t) = \ell$ and $\vec{s}$ minimizes the value of the expression $\sum_c \sum_{\tau \le t} \min\{w_c(\vec{s}, \tau), \ell - k\} \cdot x_c(\tau)$.

For each color c, we define a function $f_c \colon \mathbb{N} \to \mathbb{R}^+$. The value of $f_c(y)$ is defined as follows. Set s_c to the largest time step, such that $w_c(s_c, t) = y$. This gives $f_c(y) := \sum_{\tau \le t} \min\{w_c(s_c, \tau), \ell - k\} \cdot x_c(\tau)$.

With this notation, we have to find values y_c such that $\sum_c f_c(y_c)$ is minimized and $\sum_c y_c = \ell$. The following lemma shows that this can be done in polynomial time.

LEMMA A.1. *Given n functions $f_1, \dots, f_n \colon \mathbb{N} \to \mathbb{R}^+$ and a value ℓ, we can find, in polynomial time, values $x_1, \dots, x_n$ such that $\sum_i f_i(x_i)$ is minimized and $\sum_i x_i = \ell$.*

PROOF. We solve the problem using dynamic programming. For a pair of integers (j, γ) (each between 0 and n), we compute values $x_1, \dots, x_j$ such that $f_1(x_1) + \cdots + f_j(x_j)$ is minimized and $x_1 + \cdots + x_j = \gamma$. The solutions for $(1, \gamma)$ for all values of γ are trivial to compute. Now, in phase $i > 1$, we compute the solution for (i, γ) for any fixed γ as follows.

For each integer k between 0 and γ, lookup $(i - 1, \gamma - k)$, amend that solution by setting $x_i := k$ and compute $f_1(x_1) + \cdots + f_i(x_i)$. Choose the k that leads to the solution minimizing this sum and store it as the solution for (i, γ).

In the end, the solution (n, ℓ) is the solution we were looking for. □