New Approximations for Reordering Buffer Management

Sungjin Im * Benjamin Moseley †

Abstract

In this paper we consider the buffer reordering management problem. In this model there are n elements that arrive over time with different colors. There is a buffer that can store up to k elements and when the buffer becomes full an element must be output. If an element is output that has a color different from the previous element, a cost depending on the color must be paid. This cost could be uniform or non-uniform over colors; these are called unweighted and weighted cases, respectively. The goal is to reorder elements within the buffer before outputting them to minimize the total cost incurred.

There has been a search over the last decade to resolve the complexity of this problem online and offline. Very recently, there has been substantial progress for the unweighted case – an $O(1)$-approximation algorithm and an $O(\log \log k)$-competitive randomized algorithm were given [6, 7]. These results resolve the complexity of the unweighted buffer problem, up to constant factors, since the problem is NP-Hard and there is a matching lower bound on the competitive ratio. However, the progress for the weighted case has not been as satisfactory as for the unweighted case.

Our main result is a randomized $O(\log \log k\gamma)$-approximation for the *weighted* case, which gives an exponential improvement over the previously best known result of $O(\sqrt{\log k})$ which assumed $\gamma = \texttt{poly}(k)$. Here γ is the ratio of the maximum to minimum weight. We also revisit the unweighted case and give an improved randomized 66.0823-approximation which improves (modestly) upon the approximation guarantee given in [6]. The algorithm and analysis we use for the unweighted case was done independently of [6]. We believe that our new interpretation of the problem and our analysis of an underlying random process could be of potential use in other settings.

1 Introduction

Buffer management is an area of research with many applications in practice and due to this there is a vast amount of literature on the topic. One well-studied buffer management model is the buffer reordering management problem and its variants. In the buffer reordering problem, n elements arrive over time and the buffer can hold up to k elements. For simplicity, it can be assumed without loss of generality that elements arrive at distinct unit times. Each element e_i has some color $c(e_i)$. When an element arrives and the buffer is full, an element must be output. If the color of the element output is the same as the color of the previous element output then there is no cost for outputting this element. If there is a color change, then the cost is w_c if the new color is c. Thus, if $w_c = 1$ for all c, the problem is to output elements such that the total number of color changes is minimized. If w_c is possibly different for each c then the goal is to minimize the total weighted color changes.

The buffer reordering management problem studies a fundamental question regarding the power of a buffer for minimizing the cost of context switches, and hence it is no surprise that this problem has abundant applications in network routing, paint shops, networking, logistics, and a variety of other ares. See [21, 11, 20, 3] for extensive applications of this elegant model. Besides its applications, the problem has sustained interest in the theoretical computer science community because of the algorithmic challenges the problem poses. Indeed, the problem has been extensively studied both in the offline and online settings [21, 16, 5, 6, 7, 2, 13, 4, 1] . In the online setting, the algorithm only learns of an element once it arrives. It is known that the problem is NP-Hard [13, 4] and therefore previous work has focused on finding an algorithm with the smallest approximation (competitive) ratio. Due to the challenges faced in algorithm design, much of previous work has focused on the unweighted setting [21, 6, 7, 13, 1] and others have resorted to bi-criteria approximations where the algorithm's buffer is assumed to be larger than the optimal solution's buffer [13, 21].

The algorithmic challenge in buffer management problems occurs because the optimal solution can combine many elements together into a single output sequence which the algorithm may not. Indeed, due to this, natural algorithms for the problem fail. Perhaps the most natural algorithm to consider for the problem when the weights are *uniform* (unweighted) is Largest Color First, the algorithm that always outputs the color with the most number of elements in the buffer. Also, the algorithm never changes colors so long as there is an element in the buffer of the same color as the last element output. It is easy to see that there is no reason for

*University of California, Merced, CA, 95343, USA. sim3@ucmerced.edu. Partially supported by NSF grant CCF-1008065.

†Toyota Technological Institute, Chicago IL, 60637, USA. moseley@ttic.edu

an algorithm to change colors so long as it has an element of the same color as the last output. This algorithm is natural to consider because it chooses a color such that many elements can be output for the color, thus combining many elements for the same color into a single output sequence and outputting many elements creates more space in the algorithm's buffer to gather more elements before making its decision on the next color to switch to. However, it was shown that this algorithm has an approximation ratio of $\Omega(\sqrt{n})$ [21]. A simple example shows that the algorithm may repeatedly switch to the same color, which the optimal solution can merge into a single sequence. The optimal solution outputs other colors to accumulate many elements for the same color which the algorithm will also need to accumulate a similar cost for. Similar lower bounds can be shown for other natural algorithms such as Least Recently Used and First-In-First-Out [21].

Since natural algorithms fail to yield strong positive results, previous work has focused on developing more sophisticated algorithms. In [21], Räcke et al. introduced the reordering buffer management problem and gave an $O(\log^2 k)$ competitive algorithm when all weights are uniform. They designed an algorithm known as Bounded Waste which uses an intricate charging scheme to decide which color to output. This was then extended and improved through a sequence of papers for [16, 5, 2, 7]. We note that these works use several quite non-trivial potential function and dual fitting techniques. Online, the best known algorithms for the uniform case are an $O(\log\log k)$-competitive randomized algorithm [7] and an $O(\sqrt{\log k})$-competitive deterministic algorithm [2]. This essentially resolves the complexity of the uniform case online because there is an $\Omega(\sqrt{\frac{\log k}{\log\log k}})$ lower bound on online deterministic algorithms and an $\Omega(\log\log k)$ lower bound on randomized algorithms [2].

This line of work has essentially resolved the worst case online complexity of the unweighted case of the problem. However, the question remained on what is the best approximation ratio that can be obtained in the offline setting. To this end, few results have been shown. This is perhaps due to the fact that it is difficult to use linear programming techniques for the problem – the tight constraint given by the buffer size k makes most rounding techniques unusable. Until recently, the best offline result was a bi-criteria approximation where the algorithm has a buffer $(2+\epsilon)k$ for any constant $\epsilon > 0$ and is compared against an adversary with a buffer of size k [13]. In a breakthrough result, Avigdor-Elgrabli and Rabani gave the first offline algorithm (non-bi-criteria) which is provably better than any online algorithm in the unweighted case [6]. They showed an algorithm that achieves an $O(1)$-approximation. Unfortunately, their result strongly uses the fact that the costs are uniform and it is not clear if their techniques can shed light onto the complexity of the non-uniform weight version of the problem offline. For the non-uniform setting the best result (even offline) is an $O(\sqrt{\log k})$-competitive algorithm. We note that this algorithm requires the ratio of the maximum to minimum weight γ to be polynomially bounded in k. This algorithm was deterministic and essentially resolves the non-uniform case's deterministic complexity online due to the lower bound of $\Omega(\sqrt{\frac{\log k}{\log\log k}})$ on deterministic algorithms for the uniform case. The question remained on whether or not an offline algorithm can achieve an approximation ratio better than $O(\sqrt{\log k})$ in the non-uniform case.

Results: The main result of our paper is a new algorithm for the non-uniform case that achieves an $O(\log\log k\gamma)$ approximation.

THEOREM 1.1. *There is a randomized $O(\log\log k\gamma)$-approximation for the non-uniform (weighted) buffering reordering management problem.*

This gives an exponential improvement over the best previously known result of $O(\sqrt{\log k})$ since they assume γ is polynomially bounded in k. To show our result, we introduce a new algorithm. Our algorithm is guided by a solution to a linear programming relaxation. The algorithm rounds the LP solution in an intricate manner with several phases. The main idea behind the algorithm is to output elements in a way such that either the LP solution is accumulating similar cost, or the LP solution must be accumulating many elements while not processing them. Eventually, the LP will not be able to hold all of the elements in its buffer and must accumulate some cost which the algorithm can charge to. The algorithm itself is rather involved and intricate, but in Section 3.1 we describe the main ideas guiding the development of the algorithm.

We also revisit the uniform (unweighted) case. As mentioned, it was previously shown that an $O(1)$-approximation exists. The algorithm and analysis in [6] giving this result were fairly involved and the approximation ratio given was somewhat large[1]. In this paper, we introduce a new $O(1)$-approximation for this problem. We interpret the problem as a covering problem. Using this new interpretation, we introduce an algorithm that achieves a 66.0823-approximation which modestly improves the previous approximation guarantee given in [6]. However, the main point of this result is not improving the approximation guarantee, but is in the algorithm and solution we use. We note that our work here was done independently of that of [6]. We believe that our new clean view of the problem could prove useful for algorithm design in variants of the buffer management reordering problem. Also our formulation and analysis of the underlying random process that fills 'holes' arising in the covering problem may be of independent interest.

[1]The approximation was not explicitly given in the paper [6], but is on the order of 100.

THEOREM 1.2. *There is a randomized* 66.0823*-approximation for the uniform (unweighted) buffering reordering management problem.*

Other Related Work: Besides the non-uniform case we study, there are several variants of the buffer management problem that have been considered. A line of work has focused on the case where the costs of switching between two colors form a line metric [18, 17]. This also has been further generalized to the case where the costs form a general metric [15, 10]. Also, the buffer recording management problem has been considered with the dual objective of maximizing the total number of elements output that are the same as the previous color. For this problem, $O(1)$-approximation algorithms are known [19, 9].

2 Preliminaries

In this section, we introduce notation, the linear program (LP) which we consider as well as some simple lemmas that will prove useful throughout the paper. We begin by introducing notation. There are n elements that arrive over time. Each element e_i arrives at a distinct integral time $i \in [1, n]$ and is associated with a color. There is a buffer of size k and, once the buffer is full, an element must be output. Outputting an element of color c incurs cost w_c if the last element output has a color different from color c. Let $\mathcal{C}$ denote the set of possible colors and $|\mathcal{C}| = m$ for some m. We let $c(e_i)$ denote the color of element e_i. We will let $\mathsf{Cost}_{\mathsf{LP}}$ denote the cost of the LP solution and $\mathsf{Cost}_{\mathsf{LP}_c}$ be the cost the LP accumulates for color c. For a fixed algorithm, let $\mathcal{B}(t)$ denote the set of elements in the algorithm's buffer at time t. For a given algorithm A, let $n_c^A(t)$ be the total number of elements in the algorithm's buffer at time t for color c. Throughout the analysis we will be concerned with sequences where a set of elements of the same color is output. We will call these *color blocks* or simply block if there is no confusion.

As mentioned before, we can without loss of generality assume that any reasonable algorithm continues a color block until exhausting all elements for the color. Further, we can assume that any algorithm first stores the initial k elements and outputs an element at each time $t \in [k+1, k+n]$. Hence we will think that the buffer at time t contains the element e_t arriving at the time t, and the algorithm must immediately output an element from the buffer.

2.1 Linear Programming Formulation In order to introduce our integer/linear program we need to define some notation. Let B denote the entire set of possible color blocks. We can without loss of generality assume that all elements in each color block b should be ordered in increasing order of their arrival times. Further, we can WLOG assume that each color block b is maximal in the sense that the block, which is specified by its starting element and time, includes the maximum number of elements of the same color. Note that B has a size of a polynomial in n. We let $c(b)$ denote the color of block b. If an element e_i is scheduled at time t in block b, we denote it by $(i, t) \in b$.

We are now ready to present our integer programming formulation IP.

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

$$\text{(2.1)} \quad \text{s.t.} \quad y_{i,t} = \sum_{(i,t) \in b} x_b \qquad \forall i, t$$

$$\text{(2.2)} \quad \sum_{i \in [n]} y_{i,t} = 1 \qquad \forall t \geq k+1$$

$$\text{(2.3)} \quad \sum_{t \in [k+1, k+n]} y_{i,t} = 1 \qquad \forall i \in [n]$$

$$\text{(2.4)} \quad \bar{y}_{i,t} = \sum_{i, t' \leq t} y_{i,t'} \qquad \forall i \in [n], t \in [k+1, k+n]$$

$$\text{(2.5)} \quad \bar{y}_{p(i),t-1} \geq \bar{y}_{i,t} \qquad \forall i \in [n], t \geq k+1$$

$$\text{(2.6)} \quad x_b \in \{0, 1\} \qquad \forall b \in B$$

The 0-1 variable $x_b = 1$ if and only if we schedule color block b. In the objective cost $w_{c(b)}$ is incurred for each color block b of color $c(b)$ where $x_b = 1$. The variable $y_{i,t} = 1$ if and only if element e_i is scheduled at time t by a color block, and this is what constraints (2.1) ensure. Constraints (2.2) say that at any time after the buffer becomes full, exactly one element must be scheduled, and this guarantees that the buffer never overflows. In particular Constraints (2.2) yield

$$\text{(2.7)} \quad \sum_{i \in [n], t' \leq t} y_{i,t'} = t - k \qquad \forall t \in [t+1, t+k]$$

Constraints (2.3) say that every element must be eventually output. Constraints (2.4) define variable $\bar{y}_{i,t}$ that denotes if element e_i is scheduled by time t or not. In constraints (2.5), $p(i)$ denotes the element of color $c(e_i)$ that arrives the latest before e_i. If no such element $p(i)$ exists, we let $\bar{y}_{p(i),t-1} = 1$. Constraints (2.5) ensure that elements of the same color are processed in first-in-first-out order.

We obtain our linear programming relaxation LP by relaxing constraints (2.6) to $x_b \in [0, 1]$. We will refer to the quantity x_b as the *height* of the color block b. We will say that LP has element e_i by an amount of $1 - \bar{y}_{i,t}$ (in its buffer) at time t. We will interchangeably say that we schedule an element and output an element.

2.2 Useful Lemmas and Sampling Before we begin our analysis, we state some simple lemmas that will be useful

throughout the paper. The first lemma below is similar to a lemma shown in [2]. The following lemma applies to all feasible schedules, not only to our algorithm A.

LEMMA 2.1. *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* LP *has processed the first element e_i in b by at least ϵ by time t. 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 $(t', t'']$.*

Proof. Notice that e_i arrives after t' because otherwise the algorithm would have scheduled e_i with the element it scheduled at time $t' - 1$. Now, we know that e_i is not scheduled by A before time t, and e_i is processed by LP by at least ϵ during $(t', t]$. Then the lemma follows from the fact that the color block b is maximal, and color blocks in LP schedule elements in first-in-first-out order.

This lemma will allow us to schedule a color block by starting with an element that has been processed by the LP substantially. The following proposition follows from constraint (2.1) in the LP.

PROPOSITION 2.1. *Suppose that the* LP *has a set $\mathcal{I}$ of color blocks, all starting no later than some time t, for a specific color c and having total height at least h. Further, suppose that each of these color blocks processes at least ℓ (possibly different) elements after time t for color c that all arrive no later than 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.*

We now discuss a sampling scheme which will be very useful for our algorithm and analysis. Our sampling scheme will be useful when our algorithm has no element in the buffer that has been substantially processed by the LP. In that case, we will attempt to schedule an element in the algorithm's buffer with a probability in proportion to the amount of the element that has been processed by the LP. One natural yet naive idea is to sample each element independently, and schedule one of the sampled elements. However, this approach makes it difficult to relate the algorithm's cost to the LP's cost since the cost is determined by color blocks output. Hence we will develop a technique that samples color blocks from the LP, rather than elements.

We will say that two color blocks are disjoint if they share no element. Also we say that a color block b is *maximal* if it starts with some element e_i and, when the block ends, it schedules all elements that arrive after e_i, but before the current time, of color $c(e_i)$.

LEMMA 2.2. *For any constant $0 < \alpha < 1$, there exists a randomized algorithm that associates each element e_i with a time step t_i^α that satisfies the following:*

- *For any element e_i and time step t such that $\bar{y}_{i,t} \leq \alpha$, $\Pr[t_i^\alpha \leq t] \geq (1 - 1/e)\bar{y}_{i,t}/\alpha$.*
- *For any two elements e_i and $e_{i'}$ where $c(e_i) \neq c(e_{i'})$, the probability $\Pr[t_i^\alpha \leq t]$ is independent of $\Pr[t_{i'}^\alpha \leq t]$.*
- *Consider any collection B' of disjoint maximal color blocks where each color block $b' \in B'$ schedules at least one element e_i at time $t \geq t_i^\alpha$. Then the total expected cost of color blocks in B' is at most $\frac{1}{\alpha}\mathsf{Cost}_{\mathsf{LP}}$ in expectation.*

We will refer to t_i^α as element e_i's α-ready time. We say that element e_i is α-ready at time t if $t_i^\alpha \leq t$. The properties claimed in 2.2 prove to be very useful. The first property says that one can get a lot of elements ready for schedule compared to the volume of work that has been done on the elements by LP. Intuitively, this will increase the probability of successfully finding elements to schedule. The third property ensures that a schedule guided by α-ready times incurs a cost comparable to that of LP. This property will allow our algorithm to schedule an α-ready element in the buffer.

At a high-level, our sampling that achieves the properties claimed in Lemma 2.2 has a spirit similar to threshold rounding and configuration style LP rounding. Scheduling elements that have been processed by LP by ϵ can be viewed as a threshold rounding. The α sampling is used to "boost" the probability that elements are α-ready by scaling up the sampling probability by a factor of $1/\alpha$. The actual process of associating each element with a color block that schedules the element can be done by sampling color blocks from the LP solution. Here we have to be careful to cover all elements without sampling too many color blocks. Hence for each color, we define a suitable set of configurations where each configuration corresponds to a set of color blocks that cover each element only once, and sample a configuration.

To prove Lemma 2.2, we first formally describe our sampling method. We will sample a collection of color blocks for each color c such that each element of color c appears exactly once in a color block sampled, and the total expected number of color blocks sampled is at most the optimal LP cost, $\mathsf{Cost}_{\mathsf{LP}}$. We will repeat this $1/\alpha$ times; for simplicity, assume that $1/\alpha$ is an integer, but we can extend this to any $0 < \alpha < 1$. We will call this α-sampling. We can implement α-sampling as follows. Consider any fixed color c. We first need to pack color blocks of c into a rectangle R_c which gives a nice representation of the entire set of color blocks of c scheduled by the LP. We create a rectangle R_c with width n_c and height 1 where n_c is the total number of elements of color c. The rectangle R_c is slotted into units, $1, 2, 3, ..., j, ..., n_c$, horizontally. Each slot is mapped to a unique element of color c based on arrival times of elements of color c. Let $e'_1, e'_2, ..., e'_{n_c}$ denote the elements of color c in increasing order of their arrival times, and the jth slot is

mapped to an element e'_j. See Figure 1 for a visualization of this.

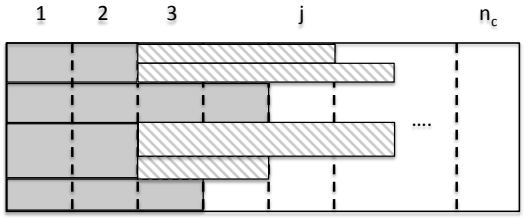

Figure 1: R_c in the α-sampling

We pack rectangle R_c with color blocks b of color c chosen by the LP. We say that block b has height x_b and width z_b which is equal to the number of elements scheduled in block b. At each time we consider the earliest slot j which is not full. Knowing that each element e'_j is scheduled exactly by one unit in LP (constraints (2.3)), we know that there exist color blocks B_j of color c starting with e'_j that were not used yet for packing R_c of total height exactly equal to the total height of empty area of the slot j. We fill this empty slot area with block B_j. A block can be used only once. Here a block can be split into several identical blocks possibly different heights while preserving the total height. By an easy induction, we can show that we can fill R_c fully with the color blocks of color c in LP, in polynomial time.

Let α_c denote a value that is chosen from $[0, 1]$ uniformly at random. Consider a horizontal ray that is off from the bottom of the rectangle R_c by α_c, and we add to Bag_c each color block packed in R_c that intersects the ray. We add all color blocks in $\bigcup_c \mathsf{Bag}_c$ to $\mathcal{B}$. We repeat this entire process $1/\alpha$ times.

PROPOSITION 2.2. *The expected number of color blocks in* Bag *is at most* $(1/\alpha)\mathsf{Cost}_{\mathsf{LP}}$.

Proof. Note that each color block b is sampled with probability x_b in each iteration of α-sampling. Since it is repeated $1/\alpha$ times, the proposition follows.

Note that each element e_i appears in exactly $1/\alpha$ color blocks in Bag. Let t_i^α denote the earliest time when e_i is scheduled by any of blocks in Bag. Then we observe the following property which will be very useful for our analysis.

LEMMA 2.3. *Consider any element e_i and time t. Suppose that $\bar{y}_{i,t} \le \alpha$. Then we have $\Pr[t_i^\alpha \le t] \ge \frac{1-1/e}{\alpha}\bar{y}_{i,t}$.*

Proof. Observe that in each random process, the probability that element e_i is scheduled by time t in a color block sampled is exactly $\bar{y}_{i,t}$. This is because the total height of the color blocks in LP that schedule e_i by time t is exactly $\bar{y}_{i,t}$ due to Constraint (2.1). Furthermore, each of those blocks is sampled with probability of its height since all those blocks intersect the slot in $R_{c(e_i)}$ corresponding to the element, and do not properly intersect with each other (but can touch). Since we repeat α-sampling $1/\alpha$ times independently, we know that $\Pr[t_i^\alpha > t] \le (1-\bar{y}_{i,t})^{1/\alpha} \le \exp(-\bar{y}_{i,t}/\alpha)$. Since $\bar{y}_{i,t}/\alpha \le 1$, we derive that $\Pr[t_i^\alpha \le t] \ge 1-\exp(-\bar{y}_{i,t}/\alpha) \ge (1-1/e)\bar{y}_{i,t}/\alpha$.

Finally, we can extend this sampling to work for arbitrary $0 < \alpha < 1$ using Poisson distribution, removing the assumption that $1/\alpha$ is an integer. That is, we repeat α-sampling $\mathsf{Pois}(1/\alpha)$ times. By a simple calculation, one can show that Proposition 2.2 and Lemma 2.3 are still satisfied.

We are now ready to prove Lemma 2.2.

Proof of [Lemma 2.2] The first property follows from Lemma 2.3. The second property holds since we sample a different α_c for each color c independently. To show the last property, consider each color c and each color block b' of color c in B' in increasing order of the times when they are scheduled – this ordering also aligns with the order of elements arrival. Let e_i be an element that b' schedules at time $t \ge t_i^\alpha$. Then we associate the color block b' with a color block b in Bag that schedules e_i by time t_i^α. Note that b' schedules e_i no earlier than b schedules e_i. Hence b' can schedule as many elements as b after e_i. By repeating this, we can map each block in B' to a unique block in Bag, hence the total cost of color blocks in B' is at most that of color blocks in $\mathcal{B}$, which is in turn upper bounded by $(1/\alpha)\mathsf{Cost}_{\mathsf{LP}}$ by Proposition 2.2. □

3 Weighted Reordering Buffer

In this section we consider the weighted version of the problem and give an $O(\log\log k\gamma)$ approximation algorithm. To do this, we utilize the α-rounding introduced in Section 2 for some α to be fixed later. At any time, once an element in the algorithm's buffer is α-ready, we are going to allow our algorithm to schedule the color corresponding to this element. Then using Lemma 2.2 we will be able to bound the expected cost of these color blocks by the LP's cost. However, there will be points in time where the LP will not have an element to schedule. In these cases, we will have to decide on a color to choose. We will choose these other colors in a intricate manner that will allow us to charge to the LP. The high level idea is to do colors in a way that the LP accumulates elements that we already scheduled. Thus, we call our algorithm `Accumulate`.

3.1 Algorithm and Analysis Overview and Intuition

Our algorithm specifies which color to switch to once it runs out of elements for the current color it is outputting. The algorithm consists of several rules depending on current state of the LP and the algorithm's buffer. Before we formally describe our algorithm, we first give some intuition on how we designed our algorithm that is an $O(\log\log k\gamma)$-approximation. The full description of our algorithm will be

presented in Section 3.2, and the pseudocode will be given in Appendix B. Missing proofs will be given Section 3.3. We begin with discussing three 'simple' rules (i), (ii) and (iii) that are of similar spirit.

Rule i: The first rule (i) is to output a color c at time t when there is an element e_i in the algorithm's buffer for the color c which has been processed by the LP by more than a constant factor. Intuitively, we can charge the cost of this c-color block to $\mathsf{Cost}_{\mathsf{LP}}$ since LP also has accumulated substantial cost for the element e_i. Essentially, between each time we output color c due to this case, the LP must accumulate cost ϵw_c because it completes an element for color c by ϵ. This is similar to an idea used in [2]. For this case we appeal to Lemma 2.1, which we use to show the following lemma.

LEMMA 3.1. *Let $\mathsf{Cost}_{A,1}$ be the total cost incurred when the algorithm switches to color c according to Rule (i). That is there is an element $e_i \in \mathcal{B}(t)$ for color c that is completed by ϵ by LP at time t. Then it follows that $\mathsf{Cost}_{A,1} \leq O(\frac{1}{\epsilon})\mathsf{Cost}_{\mathsf{LP}}$.*

Rule ii: The second rule (ii) is to output a color c at time t when it is α-ready. Here we say that a color c is α-ready when there is an element of the color that is α-ready at the time. In this case the algorithm can safely output color c in this case because Lemma 2.2 shows that the expected cost is $\frac{1}{\alpha}\mathsf{Cost}_{\mathsf{LP}}$.

LEMMA 3.2. *Let $\mathsf{Cost}_{A,2}$ be the total cost incurred when the algorithm switches to a color according to Rule (ii). Then it follows that $\mathbb{E}[\mathsf{Cost}_{A,2}] \leq \frac{1}{\alpha}\mathsf{Cost}_{\mathsf{LP}}$.*

Rule iii: The last rule (iii) of simpler cases it to output a color c when a constant fraction of the algorithm's buffer is occupied by elements for color c. Intuitively, in this case the algorithm can output the same color at most a constant number of times before the LP must output this color. This is because the LP also must output color c before its buffer overflows.

LEMMA 3.3. *Let $\mathsf{Cost}_{A,3}$ be the total cost accumulated by the algorithm when the algorithm switches colors because of Rule (iii). Then $\mathsf{Cost}_{A,3} \leq O(\frac{1}{\epsilon})\mathsf{Cost}_{\mathsf{LP}}$.*

We now discuss more involved rules (iv), (v) and (vi) in the algorithm. Note that these rules can be applied only when all the previous rules cannot be applied.

Rule iv: Here we keep an indicator variable ϕ_c for each color c, and if ϕ_c is 0 for colors, the following operation is performed and some of the colors' indicator variables are set to one. We will only output a color if its indicator variable is 0, so that it is safe to charge to the LP. Now, in this case instead of outputting a single color, we will set many colors which are safe to output. We will 'mark' a set of colors which are now safe to do. Once we output the color, we unmark it. To decide which colors to mark, we geometrically group colors in the buffer based on their weight and the number of elements in the buffer for the color. There will be at most $\log k\gamma$ groups. Then we find the group with the largest number of elements for the colors in the group, which will be most effective for emptying the buffer as much as possible. We mark all colors in this group, but we do not set all indicator variables. Rather, we sort the colors roughly based on the next time the LP will output these colors by a constant amount. We then set the indicator variables for essentially only for the first half of the colors in this group. Since we geometrically grouped colors by the number of elements and weight, we can show the weight of the colors whose corresponding indicator variables were set to one can be used to pay for all colors in the group. At the same time, we can also show that the colors whose indicator variables are set to one, as well as the other colors in the group, contain a constant fraction of the the total number of elements in the group. Finally there are at most $\log k\gamma$ groups, and we marked the group with the most number of elements, so there are $\Omega(k/\log k\gamma)$ elements in the group. Since we only allow these colors to be output if their indicator variable was originally 0 and they are only reset once the LP accumulates some cost for the corresponding colors, we will be able to show the following lemma.

LEMMA 3.4. *Let $\mathsf{Cost}_{A,4}$ be the total cost accumulated by the algorithm such that the algorithm switches colors due to Rule (iv). It is the case that $\mathsf{Cost}_{A,4} \leq O(\frac{1}{\epsilon})\mathsf{Cost}_{\mathsf{LP}}$.*

The reason we only increase the indicator variables for the first half of the colors is the following. Whenever, there is *any* color c such that ϕ_c is non-zero and there are no colors left which are marked, there are at least $\Omega(k/\log k\gamma)$ elements in the LP's buffer that are not in the algorithm's buffer. To see that this is the case fix some time t where this holds and let t' be the time last time ϕ_c was set to be non-zero. At time t' the colors in the group were sorted by when the LP processes the colors next after t'. The indicator variables were only increased for the colors which the LP will output the soonest. Further, by construction there are $\Omega(k/\log k\gamma)$ elements in the group whose indicator variables were not increased, the LP must not have processed these elements by a constant factor by time t' (otherwise, at time t the algorithm would have processed the color which has an element processed a lot by the LP) and since we marked all colors in the group, the algorithm must have already scheduled these elements by time t. Thus, all the elements in the algorithm's buffer at time t' for colors which were marked but whose indicator variables were not increased must be in the LP's buffer at time t.

The LP can partially process some elements, but es-

sentially what the previous step ensures is that there are $\Omega(k/\log k\gamma)$ elements in the LP's buffer, that are not in the algorithm's buffer, processed by at most a small constant. This implies that when we see ϕ_c is non-zero for some c, no element is processed by more than a small constant in the algorithms buffer and there are no colors left which are marked this implies that the LP must have done an $\Omega(k/\log k\gamma)$ volume of processing on the elements in the algorithm's buffer. This is because the LP has all the elements in the algorithm's buffer plus the $\Omega(k/\log k\gamma)$ elements extra elements all at most scheduled by a small constant factor and the LP's buffer can hold a total volume of at most k. The remaining two rules depend on whether these extra elements consist of many colors or not.

Rule v: We now discuss Rule (v). We only perform this case when there is a color whose indicator variable is set. The above argument shows that this implies that the LP must have processed elements in the algorithm's buffer by a substantial amount in total, $\Omega(k/\log k\gamma)$. For this case we assume that most of this processing is done on colors which the algorithm has a small number of elements for. Specifically, less than $k/\log^3 k\gamma$ elements. Now we know the LP has concentrated a lot of work on a small number of elements. Knowing this, we will be able to show that although each element $e(c)$ has been processed in the LP for just a small amount, the α-sampling will make at least one element α-ready with probability at least $1 - 1/k^2$. This crucially relies on the assumption that the LP did a large amount of work on colors that have a small number of elements. If the bad event occurs, the algorithm will choose some element to output. We will show that even if the bad event occurs, the algorithm will only output an element for color c a total $O(k)$ times before the LP accumulates cost $\Omega(w_c)$ for the color. In expectation, we can charge the cost the algorithm accumulates in this case to the LP. We will show the following lemma.

LEMMA 3.5. *Fix any color c. Let $\mathsf{Cost}_{A,5,c}$ be the total cost the algorithm accumulates due to switching to color c because of Rule (v). Then $\mathbb{E}[\mathsf{Cost}_{A,5,c}] \leq O(\frac{1}{\epsilon})\mathsf{Cost}_{\mathsf{LP}_c}$.*

Rule vi: In the final case, the LP puts the $\Omega(k/\log k\gamma)$ total processing into elements which the algorithm has a lot of elements for. Specifically, more than $\Omega(k/\log^3 k\gamma)$. In this case, we can show that for at least one of these colors the total number of elements in the algorithm's buffer for the color is within a constant factor of the total number of elements the LP has for the color. This argument is technical and depends on the specific state of the LP's buffer. In this case, the algorithm will output such a color.

Now we discuss why it is okay for the algorithm to output such a color. Fix any color c If the total number of elements output for color c by the algorithm is within a constant factor of the number of elements the LP has for color c, it can be shown that since the last time the algorithm performed this operation and switched to color c, the total number of elements in the LP's buffer for color c has increased by a constant factor. Say that this factor is $1/4$. This holds so long as the LP does not output this color. If the LP does output this color, then the algorithm simply charges the accumulated cost to the LP. Otherwise, knowing that the algorithm always outputs at least $\Omega(k/\log^3 k\gamma)$ elements every time it performs this operation, the total number of elements in the LP's buffer for color c after performing this operation $\Omega(\log\log k\gamma)$ times for the same color c will be $\Omega(k/\log^3 k\gamma)(1+1/4)^{\Omega(\log\log k\gamma)} > k$. However, all of these elements will be in the LP's buffer, which will contradict the size of the LP's buffer. Thus, the algorithm will switch to the same color c at most $O(\log\log k\gamma)$ times due to Rule (vi) between each time the LP outputs the color by a constant amount. Showing the following lemma will complete our analysis.

LEMMA 3.6. *Let $\mathsf{Cost}_{A,6,c}$ be the total cost generated by the algorithm for color c due to Rule (vi). It is the case that $\mathsf{Cost}_{A,6,c} \leq O(\frac{\log\log k\gamma}{\epsilon})\mathsf{Cost}_{\mathsf{LP}_c}$.*

Lemmas 3.1, 3.2, 3.3, 3.4, 3.5, and 3.6 will imply Theorem 1.1. The actual analysis of each of these cases is more complicated because the LP can partially process elements, start many color blocks which contain the same elements and can merge many elements together which the algorithm does separately. However, this is the high level intuition.

3.2 Algorithm Let ϵ be any constant between 0 and $1/100$ and α be a constant at most ϵ. We will later set $\epsilon = 1/100$ and $\alpha = \epsilon$. For each color c define the following times inductively. 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.

We now define our algorithm. First the algorithm performs the α-rounding in Section 2. For each color c, we have an indicator variable ϕ_c. Initially the indicator variables are set to 0. When the algorithm outputs a color at certain times, it will increment these variables. When any time in $\mathcal{T}_c$ passes, the variables ϕ_c are reset to 0. At any time if the algorithm find elements in the buffer of color that is the same as the current color being output, then the algorithm outputs the earlier arriving element among those. The following algorithm is used whenever the algorithm needs to change colors. 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 $1/2+\epsilon$. Let $G_j(t)$ contain the elements $e_i \in \mathcal{B}(t)$ where $2^{j-1} \leq w_{c(e_i)}/n^A_{c(e_i)}(t) \leq 2^j$. Here we only need consider $j \in [\log(\frac{w_{max}}{\gamma n}) - 1, \log w_{max} + 1]$ where w_{max} denotes the maximum weight that an element can have. Let $M(t)$ denote a set of marked colors at time t. This set is initially empty. A color that is marked, is a color the algorithm is free to schedule.

Now we will also be concerned with elements in the algorithm's buffer which are not being currently scheduled by the LP by more than a constant factor. Let $c^*(t)$ be the color (if it exists) which the LP has a set of color blocks for, that intersect time t, and whose total height is more than $1/2$. Note that there can be at most one such color at any time by constraint (2.2) in the LP. We define $\overline{\mathcal{B}}(t)$ to be all elements in the algorithm's queue excluding those for color $c^*(t)$. Let $\overline{G}_j(t)$ contain the elements $e_i \in \overline{\mathcal{B}}(t)$ where $2^{j-1} \leq w_{c(e_i)}/n^A_{c(e_i)}(t) \leq 2^j$.

The algorithm is formally described as follows. The pseudocode of the algorithm can be found in Appendix B.

Algorithm: Accumulate(t). The algorithm *prioritizes* Rules (i), (ii), (iii), (iv), (v) and (vi) in this ordering. That is, whenever the algorithm outputs a color, it starts from the beginning and performs the first rule that is applicable. The algorithm is used to make a decision at any time t where there is no element in the buffer which is the same color as the last element output.

Rule (i) if there is an element e_i where $\bar{y}_{i,t} \geq \epsilon$ output color $c(e_i)$.

Rule (ii) if there is an element e_i that is α-ready, output color $c(e_i)$.

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

These are the simple cases.

Now we describe the more complicated cases. Each color c has a indicator ϕ_c variable initially set to 0. We also maintain a set $M(t)$ of colors which is initially empty. Colors in this set we called *marked*. If we come to a time step where the previous rules do not apply, $M(t) = \emptyset$ and $\phi_c = 0$ for all colors then we do the following. Let S be the colors for elements in $\overline{G}_{j^*}(t)$ where $j^* = \texttt{argmax}_j |\overline{G}_{j^*}(t)|$. Now we sort the colors in S based on when the LP will do them next. That is, we sort them $c_1, c_2, \ldots c_{|S|}$ so that $t_{c_l} \leq t_{c_{l+1}}$ for all l. Now we find the index of the color which roughly divides the elements corresponding to these colors into two equal sized sets. That is we find the index j' where $\sum_{a=1}^{j'} n^A_{c_a}(t) \geq \sum_{a=1}^{|S|} n^A_{c_a}(t)/2$. We now set the indicator variables for the first half of the colors. That is we set $\phi_{c_a} = 1$ for all $A \leq j'$. We finally added all colors in S to $M(t)$. Recall that we reset ϕ_c to 0 if a time in T_c passes.

Rule (iv) if $M(t)$ is non-empty output a color $c \in M(t)$ and remove c from $M(t)$. Now for the final two cases. These cases are only applicable if $M(t) = \emptyset$, but $\phi_c \neq 0$ for some color c. Let $C_s(t)$ contain the colors which there are not too many elements in the algorithm's buffer for; that is, the colors where $0 < n_c^A(t) \leq k/\log^3 k\gamma$. Let $C_b(t)$ be the other colors in $\mathcal{B}(t)$. Let $E^O(t) := \{e_i \notin \mathcal{B}(t) \mid i \leq t, \bar{y}_{i,t} \leq 1/2 + 2\epsilon\}$ be the elements which still require lots of processing by the LP, but are not in $\mathcal{B}(t)$.

Rule (v) if $\sum_{e_i \in \mathcal{B}(t), c(e_i) \in C_S(t)} \bar{y}_{i,t} \geq |E^O(t)|/8$ then switch to the color $c \in C_s(t)$ such that $t_c = \min_{t' \in \mathcal{T}_c, t' > t} t'$ is minimized. Otherwise,

Rule (vi) switch to the color $c \in C_b(t)$ where $n_c^A(t) \geq \frac{3}{5} n_c^O(t)$. Note that for this case, it is not obvious such a color must exist, but we will show that this is the case any time the algorithm reaches Rule (vi). This completes the algorithm descriptions.

3.3 Analysis In this section we prove Theorem 1.1 by completing the proof of Lemmas 3.3, 3.4, 3.5, and 3.6.

Proof of [Lemma 3.3] Consider any color c and a time $t_{c,i-1}$ in $\mathcal{T}_c$ for any i. Let $t_{c,i}$ be the earliest time in $\mathcal{T}_c$ after $t_{c,i-1}$ and if no time exists set $t_{c,i} = \infty$. We will show that the algorithm cannot output a color 51 times during $[t_{c,i-1}, t_{c,i})$ due to Rule (iii). If this holds for all i this will imply the lemma. This is because between two times in $\mathcal{T}_c$ the LP accumulates cost at least ϵw_c for color c and if $t_{c,i} = \infty$ we can charge to the total cost the LP accumulates for color c. For the sake of contradiction, say that a color c is output 50 times by Rule (iii) between $t_{c,i-1}$ and $t_{c,i}$. Let t_j and t_{j+1} be any two times in $[t_{c,i-1}, t_{c,i})$ where the algorithm switches to color c by Rule (iii). Let t'_j and t'_{j+1} be the time the color blocks started by the algorithm at times t_j and t_{j+1} end, respectively. Notice that there are at least $k/10$ elements for color c that arrive in the time interval $[t'_j, t_{j+1})$ and are not scheduled by the algorithm until time t_{j+1}. Since the algorithm outputs color c 51 by Rule (v), we deduce that at least $5k$ elements for color c arrive during $[t_{c,i-1}, t_{c,i})$.

We claim that there must exist a set of color blocks, $\mathcal{I}$, in the LP of total height greater than $1/2$, which start before time $t_{c,i-1}$, such that each of them contains all but at most $(2 + 10\epsilon)k$ elements for color c that arrive during $[t_{c,i-1}, t_{c,i})$. Otherwise each of the last $(2 + 10\epsilon)k$ elements of color c arriving during $[t_{c,i-1}, t_{c,i})$ is processed by at most half by color blocks starting before time $t_{c,i-1}$. Further, it can be processed by at most ϵ by color blocks starting after the same time $t_{c,i-1}$ since in the LP the total height of all the color blocks for color c beginning during $[t_{c,i-1}, t_{c,i})$ can be at most ϵ by definition of times in $\mathcal{T}_c$. Hence LP will have a volume of at at least $(2 + 10\epsilon)k/2 - \epsilon(2 + 10\epsilon)k = (1-2\epsilon)(1+5\epsilon)k$ left for color c at time $t_{c,i}$. Since $\epsilon \leq 1/100$, this contradicts constraint (2.2) in the LP.

However, from the above we also know that between

two times t_j and t_{j+1} where the algorithm starts a c color block by Rule (iii) in the time interval $[t_{c,i-1}, t_{c,i})$, it is the case that the algorithm outputs $k/10$ elements of colors different from c while accumulating no smaller than $k/10$ elements of color c. The intervals in $\mathcal{I}$ which intersect t_{j+1} must span over these times. This implies that each color block in $\mathcal{I}$ processes at least $5k - (2 + 10\epsilon)k$ elements for color c after time $t_{c,i-1}$ which also arrive before this time. This can be deduced from the fact that LP has "long" while the algorithm has 50 color c blocks of length at least $k/10$ separated by at least $k/10$ element of other colors – such gaps must be bridged by elements arriving before $t_{c,i-1}$. Since each color block in $\mathcal{I}$ starts before time $t_{c,i-1}$ and the total height of the color blocks in $\mathcal{I}$ is at least $1/2$, the LP must have a total volume of at least $(5k - (2 + 10\epsilon)k)/2$ for elements of color c at time $t_{c,i-1}$ by Proposition 2.1. Since $\epsilon \leq 1/100$, this is greater than k and this is a contradiction to constraint (2.2) in the LP. □

Now we bound the cost of Rule (iv). In this case, our goal is to show that the colors whose indicator variables are set to one can be used to pay for all of the elements we mark. We note that this is similar to an idea used in [2].

Proof of [Lemma 3.4] Consider a time t where colors are marked. Notice that a color can only become marked if $\phi_c = 0$ for all colors corresponding to elements in $\mathcal{B}(t)$. Now the indicator variable $\phi_c = 0$ is only increased for some colors corresponding to elements in $\mathcal{B}(t)$. Let S be the set of colors that are marked at time t and $S' \subseteq S$ be the set of colors c whose indicator variables ϕ_c are set to 1 at time t. We first argue that $4\sum_{c\in S'} w_c \geq \sum_{c\in S} w_c$. Indeed, by definition of the algorithm, there is a j^* such that for any color $c \in S$ it is that case that $2^{j^*-1} \leq w_c/n_c^A(t) \leq 2^{j^*}$. Further, $\sum_{c\in S'} n_c^A(t) \geq \sum_{c\in S} n_c^A(t)/2$. Knowing this we have that,

$$\begin{aligned}
& \sum_{c\in S} w_c \\
\leq\ & 2^{j^*} \sum_{c\in S} n_c^A(t) \\
& \quad \text{[Since } 2^{j^*-1} \leq w_c/n_c^A(t) \leq 2^{j^*} \text{ for all } c \in S\text{]} \\
\leq\ & 2^{j^*+1} \sum_{c\in S'} n_c^A(t) \\
& \quad \text{[Since } \textstyle\sum_{c\in S'} n_c^A(t) \geq \sum_{c\in S} n_c^A(t)/2\text{]} \\
\leq\ & 4 \sum_{c\in S'} w_c \\
& \quad \text{[Since } 2^{j^*-1} \leq w_c/n_c^A(t) \leq 2^{j^*} \text{ for all } c \in S'\text{]}
\end{aligned}$$

Thus we have that $4\sum_{c\in S'} w_c \geq \sum_{c\in S} w_c$. Now we set each indicator variable ϕ_c to be 1 for every color c in S'. A indicator variable ϕ_4 is only reset to 0 if a time point in $\mathcal{T}_c$ is passed. We know that between two consecutive times in $\mathcal{T}_c$ the LP accumulates ϵw_c cost due to processing color c. We charge $4\sum_{c\in S'} w_c \geq \sum_{c\in S} w_c$ to these points. If no time in $\mathcal{T}_c$ ever passes again, then we charge to the total cost the LP accumulates for color c, which proves the lemma. □

Now we bound the cost of the more sophisticated operations performed by the algorithm, rules (v) and (vi). Before we bound the cost of these operations, our goal is to show a structural property of the LP's buffer at times these operations are performed. The property we want to show is that the LP has a lot of elements, not processed by at least constant factor, which are not in the algorithm's buffer. This will be useful to bound the amount the LP must have processed the elements which are in the algorithm's buffer when these operations are performed. Recall that $E^O(t)$ is the set of elements not processed by $1/2 - 2\epsilon$ in the LP's buffer at time t which are not in $\mathcal{B}(t)$.

LEMMA 3.7. *Consider any time t where $\phi_{c'}(t) > 0$ for some color c', and $c(e_i)$ is not marked for any $e_i \in \mathcal{B}(t)$. Then $|E^O(t)| \geq \frac{3k}{8\log k\gamma}$. Further, for any color, either in $C_b(t)$ or $C_s(t)$, the elements for the color are processed by at least $|E^O(t)|/4$ in total.*

Proof. Consider any time t and color c' as described in the lemma. Our goal is to show that there are at least $\frac{3k}{8\log k\gamma}$ elements in the LP's buffer at time t, that are not in the algorithm's buffer at time t, that are not processed by more than $2\epsilon + 1/2$. Let t' be the last time no later than time t where $\phi_{c'}$ was set to be non-zero. At time t', we marked all colors in $\overline{G}_{j^*}(t')$ for some j^*. Let S be the set of colors that were marked at time t' and let $S' \subseteq S$ be the set of colors c whose indicator variables ϕ_c where set to be positive at time t'. Let t_c be the earliest time after t' in $\mathcal{T}_c$. Let $c_1, c_2, \ldots, c_{|S|}$ be the ordering of the colors in S such that $t_{c_l} \leq t_{c_{l+1}}$. Say c' has index i^* in this ordering. We know that since the indicator variable $\phi_{c'}$ has not been reset by time t that $t_{c_l} > t$ for all $l \geq i^*$. Further, we know that $\sum_{c\in S} n_c^A(t') \geq \frac{k}{\log k\gamma} - \frac{k}{10} \geq \frac{3k}{4\log k\gamma}$ because there are at most $\log k\gamma$ groups, there are k elements in the buffer, at most $k/10$ elements are for color $c^*(t')$ since we set $\phi_{c'}$ to be non-zero at time t' and we mark the group with the maximum number of elements. By definition of the algorithm, we know that $\sum_{i=i^*}^{|S|} n_{c_i}^A(t') \geq \frac{1}{2}\sum_{c\in S} n_c^A(t') \geq \frac{3k}{8\log k\gamma}$. However, knowing that $t_{c_i} > t$ for all $i \geq i^*$, the LP has not processed any element in $\overline{\mathcal{B}}(t')$ by ϵ at time t' and no element in $\overline{\mathcal{B}}(t')$ has a color block of height greater than $1/2$ intersecting time t' in the LP, it must be the case that every element for a color c_i in $\mathcal{B}(t')$ is completed by at most $2\epsilon + 1/2$ by the LP by time t. Thus, the LP has at least $\frac{3k}{8\log k\gamma}$ elements that are not in the algorithm's buffer at time t, that are not processed by more than $2\epsilon + 1/2$.

Finally, we know that the LP can have a total volume of at most k in its buffer at time t by constraint (2.2) in the LP. Thus, the LP must process the elements in $\mathcal{B}(t)$ by at least $(1/2 - 2\epsilon)|E^O(t)|$. Knowing that $\epsilon \leq 1/100$ this implies

that elements in $\mathcal{B}(t)$ either for colors in $C_b(t)$ or $C_s(t)$ are processed by at least $|E^O(t)|/4$. □

Now our goal is to bound the cost the algorithm accumulates due to Rule (v). In this case we will show that Rule (v) happens with small probability at certain moments in time. To do this, we begin by showing the following lemma which bounds the number of elements that should be available to schedule due to the randomized rounding.

LEMMA 3.8. *Fix a color c, a time t and a set S of elements such that the following conditions hold*

- $|S| \leq k$
- *For all $e_i \in S$, $\bar{y}_{i,t} \leq \epsilon$.*
- *For all colors c', the total number of elements in S for color c' is at most $k/\log^3 k\gamma$, i.e. $|\{e_i \in S \mid c(e_i) = c'\}| \leq k/\log^3 k\gamma$.*
- $\sum_{e_i \in S} \bar{y}_{i,t} \geq \eta/8$ *for some* $\eta \geq \frac{k}{8 \log k\gamma}$.

Then it is the case that at time t there are at least 2η elements in S which are α ready by time t with probability at least $1 - 1/k^2$ for all k greater than a sufficiently large constant.

Proof. We use a concentration inequality stated in Theorem A.1 in Appendix A. Let N_c be the number of elements of color c which are α ready by time t. We would like to show that

$$\Pr\Big[\sum_{c \in S} N_c \leq 2\eta\Big] \leq \frac{1}{k^2}$$

For notational simplicity, let $\mu := \mathbb{E}[\sum_{c \in S} N_c]$. Note that $N_c \leq k/\log^3 k\gamma$. Further we know that $\mu \geq \frac{1}{16\epsilon}\eta$ by Lemma 2.2. This is because our α-rounding makes an element e_i ready for schedule by time t with a probability of at least $(1/(2\epsilon))v_i$ where v_i denotes the volume of work that the LP does on the element e_i by time t. Note that we required $\alpha \leq \epsilon$.

We are now ready to prove the lemma.

$$\begin{aligned}
& \Pr\Big[\sum_{c \in S} N_c \leq 2\eta\Big] \\
\leq\ & \exp\Big(-\frac{(\mu - 2\eta)^2}{2\sum_{c \in S}\mathbb{E}[N_c^2]}\Big) \quad \text{[By Theorem A.1]} \\
\leq\ & \exp\Big(-\frac{(\mu - 2\eta)^2}{2(k/\log^3 k\gamma)\sum_{c \in S}\mathbb{E}[N_c]}\Big) \\
& \qquad \text{[Since } N_c \leq k/\log^3 k\gamma\text{]} \\
\leq\ & \exp\Big(-\frac{(\mu - 32\epsilon\mu)^2}{2(k/\log^3 k\gamma)\mu}\Big) \quad \text{[Since } \mu \geq \tfrac{1}{16\epsilon}\eta\text{]} \\
\leq\ & \exp\Big(-\frac{(\mu/2)^2}{2(k/\log^3 k\gamma)\mu}\Big) \quad \text{[Since } \epsilon < 1/100\text{]} \\
=\ & \exp\Big(-\frac{\mu}{8(k/\log^3 k\gamma)}\Big) \\
\leq\ & \exp\Big(-\frac{\log^3 k\gamma}{8k} \cdot \frac{k}{128\epsilon \log k\gamma}\Big) \\
& \qquad \text{[Since } \mu > \tfrac{1}{16\epsilon}\eta \text{ and } \eta \geq \tfrac{k}{8\log k\gamma}\text{]} \\
\leq\ & \exp\Big(-\frac{\log^2 k\gamma}{12}\Big) \quad \text{[Since } \epsilon < 1/100\text{]} \\
\leq\ & 1/k^2
\end{aligned}$$

The last inequality holds for any k greater than a sufficiently large constant. □

The goal of the next lemma is to show how often the algorithm could possibly switch to a color before the LP accumulates as least some small cost for this color. Later, this will give us an upper bound on how many times the algorithm switches to a color due to Rule (v).

LEMMA 3.9. *Fix any color c. Let $t_1 \in \mathcal{T}_c$ and t_2 be the first time in $\mathcal{T}_c$ after t_1. If no time exists set $t_2 = \infty$. The total number of times the algorithm can switch to color c during $[t_1, t_2)$ is at most $O(k)$.*

Proof. Indeed, for the sake of contradiction say that the algorithm switches to color c at $6k + 2$ distinct times during $[t_1, t_2)$. Let $\mathcal{E}$ be the set of times during $[t_1, t_2)$ where the algorithm switches to color c. Let t' be the second time the algorithm switches to color c at a time during $[t_1, t_2)$ and t'' be the second to last time the algorithm switches to color c at a time during $[t_1, t_2)$. At least $6k$ elements for color c are output by the algorithm during $[t', t'']$ since the algorithm switches to color c at least $6k$ times after t' but before t''. Further these elements must arrive after t_1, since they were not output the first time the algorithm switched to color c after t_1.

We now consider two cases. In the first case assume that the LP does not have a set of color blocks intersecting t' and $3k$ times in $\mathcal{E}$ of total height at least $1/2$ for color c. By definition of t_1 and t_2 it is the case that the LP does not accumulate cost ϵw_c for color c between t_1 and $t_2 - 1$ for color blocks starting between t_1 and $t_2 - 1$. Thus, at any time before t_2 and after time t'' there are $3k$ elements for color c in the LP's buffer processed by at most $1/2 + \epsilon$ which the algorithm has already output. However, the LP has a total volume greater than $3k(1/2 - \epsilon) > k$ in its buffer at time t'' since $\epsilon \leq 1/100$. This contradicts the constraint (2.2) in the LP.

For the second case, assume that the LP has a set of color blocks $\mathcal{I}$ intersecting t' and $3k$ times in $\mathcal{E}$ during $[t', t'']$ of total height at least $1/2$ for color c. We know that the algorithm always outputs an element if its color is the same as the current color and, therefore, there are $3k$ times during $[t', t'']$ that intersect every color block in $\mathcal{I}$ where the

algorithm is outputting an element that is not of color c. This implies that every color block in $\mathcal{I}$ processes $3k$ elements for color c after time t' which arrive before time t'. Knowing that the total height of the color blocks in $\mathcal{I}$ is at least $1/2$, this and Proposition 2.1 implies that the LP has total volume of at least $\frac{3}{2}k$ of elements for color c at time t'. This is a contradiction to constraint (2.2) in the LP. □

Finally, we are ready to bound the cost the algorithm accumulates due to Rule (v). The high level idea is that Rule (v) happens with low probability and therefore there is a color which the algorithm can output such that the expected cost charged to the LP is small.

Proof of [Lemma 3.5] Consider any time $t_1 \in \mathcal{T}_c$ and the time t_2 such that t_2 is the earliest time in $\mathcal{T}_c$ after t_1. If no such time exists, then $t_2 = \infty$. To prove the lemma we will show that the expected number of times the algorithm outputs color c during $[t_1, t_2)$ is at most $O(1)$. Before we give the formal proof, we give a roadmap for the proof. Fix color c. We first define a set X of time steps during $[t_1, t_2)$ where the algorithm can possibly switch to color c due to Rule (v). The definition of X will rely on the LP solution with no dependency of the algorithm's decision, and hence X is a *deterministic* set. Then we further refine X into at most $O(\log k)$ disjoint subsets $\{X_l\}$ such that at *all* time steps in X_l the algorithm has at least one element in $\mathcal{B}(t)$ to schedule following Rule (i), (ii), (iii), or (iv) if a good event Good_l occurs.

We then move to proving $\Pr[\mathsf{Good}_l] \geq 1 - 1/k^2$. This is where we use the precondition of Rule (v). At this point, we can show that the algorithm has processed a certain set of elements of many distinct colors substantially in total. Then with high probability, the α sampling will make some elements α-ready. However, this may not be sufficient since the algorithm might have already scheduled all those elements at some point in X_l. Hence we will try to find "enough" α-ready elements such that the algorithm cannot help but include at least one element in its buffer at all times in X_l. This is done via a careful argument about the volume of several sets of elements we handle. This is also where we appeal to Lemma 3.8. The volume argument requires some carefully chosen conditions in Lemma 3.8 together with definition of X_l.

We now give a formal proof. For a color c', let $t_{c'}(t)$ denote the earliest time in $\mathcal{T}_{c'}$ after time t. If no such time exists then $t_{c'}(t) = \infty$. Define X to be the set of times t during $[t_1, t_2)$ such that

- (1) $|\{e_i \mid \bar{y}_{i,t} \leq \epsilon\}| \geq k$, and
- (2) $\eta_t := |\{e_i \mid \bar{y}_{i,t} \leq 1/2 + 2\epsilon\}| \geq k + \frac{k}{8 \log k\gamma}$, and
- (3) there exists a set of D (can be different for each t) of at most k elements such that
  - (3-a) for all $e_i \in D$, $\bar{y}_{i,t} \leq \epsilon$, and
  - (3-b) for all $e_i \in D$, $t_{c(e_i)}(t) \geq t_2$, and
  - (3-c) for all c', $|\{e_i \in D \mid c(e_i) = c'\}| \leq \frac{k}{\log^3 k\gamma}$, and
  - (3-d) $\sum_{e_i \in D} \bar{y}_{i,t} \geq (\eta_t - k)/8$.

Note that X is a deterministic set depending solely on the LP solution. Before proving X includes all time steps we need to consider, we explain the conditions of X in words to help the reader have a feel. At the current time t when Rule (v) is considered, we would like to find a subset D of elements some of which will likely to become α-ready. To apply α sampling, recall that we require that the element has been processed by at most ϵ (Condition (3-a)). Also to apply α sampling effectively, we need that LP has processed elements in D a lot (Condition (3-d)). Since each color has not many elements in D (Condition (3-c)), this will imply that D consists of many colors. Since α-sampling uses an independent random value α_c for each color, this will help argue that there must exist some α-ready elements by Lemma 3.8. The Condition (3-b) says that all elements in D are processed by LP by ϵ before time t_2. This condition will be useful to show Rule (v) is applied at no time in X if a "good" event occurs. The Conditions (1) and (2) will be useful in the volume argument that will lead to the conclusion that some of the α-ready elements we found must exist in the algorithm's buffer at many time steps. More details will be given as we proceed with the analysis.

CLAIM 3.10. *If the algorithm switches to color c at time $t \in [t_1, t_2)$, then it must be the case that $t \in X$.*

Proof. Suppose the condition stated in the lemma is satisfied. The first condition in the definition of X is satisfied since for all elements $e_i \in \mathcal{B}(t)$, $\bar{y}_{i,t} \leq \epsilon$ – this is because otherwise Rule (i) would be applied. By considering element e_i with $\bar{y}_{i,t} \leq 1/2 + 2\epsilon$ depending on whether $e_i \in \mathcal{B}(t)$ or not, it follows that $\eta_t = |E^O(t)| + |\mathcal{B}(t)| = |E^O(t)| + k \geq k + \frac{k}{8 \log k\gamma}$; the last inequality is due to Lemma 3.7. Hence the second condition holds true. To see the remaining conditions hold, set $D := C_s(t) \cap \mathcal{B}(t)$. By definition of $C_s(t)$, it follows that for all $e_i \in C_s(t)$, $t_{c(e_i)}(t) \geq t_2$. This is because the algorithm chose to schedule c over all other colors, in particular $c(e_i)$. Also we know that for all c', $|\{e_i \in C_s(t) \mid c(e_i) = c'\}| \leq \frac{k}{\log^3 k\gamma}$ from definition of $C_s(t)$. Finally, the last condition follows from the condition of Rule (v), and the fact that $|E^O(t)| = \eta - k$. Finally $|D| \leq k$ since $|\mathcal{B}(t)| \leq k$.

Now we define X_l by partitioning X into disjoint subsets X_l of X such that $(1 + 1/2)^{l-1} \leq \eta_t - k < (1 + 1/2)^l$. Observe that $l = O(\log k)$ since there are η_t elements e_i with $\bar{y}_{i,t} \leq 1/2 + 2\epsilon$, and therefore $\eta_t(1/2 - 2\epsilon) \leq k$ due to constraint (2.2). This in turn implies that $(1 + 1/2)^{l-1} \leq \eta_t - k \leq \frac{1/2+2\epsilon}{1/2-2\epsilon} \cdot k \leq 2k$ since $\epsilon \leq 1/100$. Also we know that $l \geq 1$ from Condition (2). Now consider a fixed set X_i

and let t be the first time in X_i. Define Good_l be the event where there is a subset $D' \subseteq D$ of $2(\eta - k)$ elements e_i that are α-ready by time t. Note that since D' is a subset of D, it inherits all conditions (3-a), (3-b), and (3-c).

CLAIM 3.11. $\Pr[\mathsf{Good}_l] \geq 1 - 1/k^2$.

Proof. The claim immediately follows by applying Lemma 3.8 with set $S := D$ stated in Condition (3).

We now argue the usefulness of partitioning time steps in X into X_l with respect to the event Good_l.

CLAIM 3.12. *Suppose the event Good_l occurs. Then at all time steps t' in X_l, there must exist an element e_i that is ready for schedule by Rules (i) or (ii), i.e. $\bar{y}_{i,t'} \geq \epsilon$ or e_i is α-ready by time t'.*

Proof. Consider any time $t' \in X_l$. Since Good_l occurs, there exists a set D' of elements as described above. From definition of D' and X_l, note that $|D'| \geq 2(\eta_t - k)$, and $(\eta_{t'} - k) \leq (1 + 1/2)(\eta_t - k)$. Let D'_{c^*} be the elements in D' for a fixed color $c^*(t)$ which have a set of color blocks intersecting time t with height greater than $1/2$. We know that there are at most $k/\log^3 k\gamma$ elements in D'_{c^*} and there can at most one color $c^*(t)$ with a set of of color blocks in the LP intersecting time t with total height greater than $1/2$ by constraint (2.2). Due to Condition (3-b), we know that for all $e_i \in D' \setminus D'_{c^*}$, $\bar{y}_{i,t'} \leq 2\epsilon + 1/2$, therefore the elements in $D' \setminus D'_{c^*}$ contribute to $\eta_{t'}$. Also observe that $|D' \setminus D'_{c^*}| \geq 2(\eta_t - k) - 8(\eta_t - k)/\log^2 k\gamma > \frac{3}{2}(\eta_t - k) \geq \eta_{t'} - k$ where the first and second inequalities follow from Condition (3-c) and (2), respectively, and the second to last inequality holds when k is greater than a sufficiently big constant.

To recap, we have shown that $D' \setminus D'_{c^*} \subseteq \{e_i \mid \bar{y}_{i,t'} \leq 1/2 + 2\epsilon\}$ and that $|D' \setminus D'_{c^*}| > \eta_{t'} - k$. Also we know that $\eta_{t'} := \{e_i \mid \bar{y}_{i,t'} \leq 1/2 + 2\epsilon\}$ and that $\mathcal{B}(t') \subseteq \{e_i \mid \bar{y}_{i,t'} \leq 1/2 + 2\epsilon\}$. Hence it must be the case that $|\mathcal{B}(t') \cap (D' \setminus D'_{c^*}) > 0|$. This completes the proof.

Now we are ready to complete our proof. We have shown that if $\bigwedge_l \mathsf{Good}_l$ occurs, then Rule (v) is not performed. Observe that $\Pr[\neg \bigwedge_l \mathsf{Good}_l] \leq O(\log k/k^2) = O(1/k)$ by a simple union bound. By Lemma 3.9 the most number of times the algorithm can switch to color c during $[t_1, t_2)$ is $O(k)$. Hence the expected number of times the algorithm switches to color c for times in X due to Rule (v) is at most $O(k) \cdot O(1/k) = O(1)$, which implies the lemma. □

Finally we focus on the last case, bounding the the algorithm's cost incurred due to Rule (vi). The following lemma shows that the algorithm always has a color to schedule at each time. That is the conditions of Rule (vi) are satisfied if the algorithm cannot use rules (i), (ii), (iii), (iv) or (v).

LEMMA 3.13. *At any time t, if none of the Rules (i), (ii), (iii), (iv) or (v) cannot be performed, then there exists a color $c \in C_b(t)$ where $n_c^A(t) \geq \frac{3}{5} n_c^O(t)$.*

Proof. Let $T_b^O(t)$ denote the set of elements in the LP's buffer, processed by at most $1/2 + \epsilon$ at time t, which are for colors in $C_b(t)$. Let $T_b^A(t)$ be the set of elements in $\mathcal{B}(t)$ for colors in $C_b(t)$. Note that $|T_b^O(t)| = \sum_{c \in C_b(t)} n_c^O(t)$ and $|T_b^A(t)| = \sum_{c \in C_b(t)} n_c^A(t)$.

To show the lemma, it suffices to show $|T_b^A(t)| \geq (3/5)|T_b^O(t)|$. Let $S = T_b^O(t) \setminus \mathcal{B}(t)$. Note that by definition of $E^O(t)$ that $S \subseteq E^O(t)$. By Lemma 3.7 and due to prerequisite condition of Rule 6, at time t the LP must have processed elements in $\mathcal{B}(t)$ for colors in $C_b(t)$ by at least $|E^O(t)|/4 - |E^O(t)|/8 = |E^O(t)|/8$. This implies that $|T_b^A(t)| \geq |E^O(t)|/(8\epsilon) \geq 10|E^O(t)|$ since $\epsilon \leq 1/100$ and no element in $\mathcal{B}(t)$ has been processed by LP by more than ϵ. Thus $|T_b^A(t)| \geq 10|S| = 10 \cdot |T_b^O(t) \setminus \mathcal{B}(t)|$. Clearly, $|T_b^A(t)| = |T_b^O(t) \cap \mathcal{B}(t)|$ by definition. Thus $|T_b^O(t)| = |T_b^O(t) \cap \mathcal{B}(t)| + |T_b^O(t) \setminus \mathcal{B}(t)| \leq \frac{11}{10}|T_b^A(t)|$. This yields $|T_b^A(t)| \geq (3/5)|T_b^O(t)|$, which completes the proof. □

Finally we bound the cost due to Rule (vi) which will complete the proof for the weighted case.

Proof of [Lemma 3.6] Let $\mathcal{E}$ be the set of times the algorithm switches to color c according to Rule (vi) as described in the lemma. At any time $t \in \mathcal{E}$, it is the case that $n_c^A(t) \geq \frac{k}{\log^3 k\gamma}$ and $n_c^A(t) \geq \frac{3}{5} \cdot n_c^O(t)$ by Lemma 3.13 and the definition of elements in $C_b(t)$. Let t_1 be a time in $\mathcal{T}_c$ and t_2 be the earliest time in $\mathcal{T}_c$ after t_1. If t_2 does not exists, set $t_2 = \infty$. We will show that the algorithm can only switch to color c $O(\log\log k\gamma)$ times during $[t_1, t_2]$. Knowing that the LP accumulates cost ϵw_c for color c during $(t_1, t_2]$ if t_1 and t_2 are in $\mathcal{T}_c$ we can charge this cost to the cost LP accumulates for color c during $(t_1, t_2]$. If $t_2 = \infty$ we charge to the total cost the LP accumulates for color blocks corresponding to color c.

For the sake of contradiction, suppose that there are at least $40\log\log(k\gamma) + 4$ times in $\mathcal{E}$ during $[t_1, t_2]$. We break the analysis into two cases. For the first case assume that there are at most $20\log\log(k\gamma) + 2$ time steps in $\mathcal{E}$ during $[t_1, t_2]$ where the LP has a set of color blocks for color c of total height greater than $1/2$ that all intersect time t_1. Let t'_1 be the first time in $\mathcal{E}$ during $[t_1, t_2]$ when there is not a set of color blocks for color c of total height greater than $1/2$ that intersect t_1. There are at least $20\log\log(k\gamma) + 2$ time steps in $\mathcal{E}$ during $[t'_1, t_2]$ where there is not a set of color blocks for color c of total height greater than $1/2$ in the LP since the LP can start color blocks for color c of total height at most height ϵ during $[t_1, t_2]$.

Let t''_1 be the second time the algorithm starts a color block for c at a time in $\mathcal{E}$ during $[t'_1, t_2]$. Any element for

color c output by the algorithm during $[t_1'', t_2]$ arrives after time t_1'. Further, by definition of t_1', for any of these elements e_i it is the case that $\bar{y}_{i,t_2} \leq 1/2 + \epsilon$. Let $\overline{n}_c^O(t)$ be the elements e_i with $\bar{y}_{i,t_2} \leq 1/2 + \epsilon$ which arrive after time t_1' for color c. Note that this implies that $n_c^O(t) \geq \overline{n}_c^O(t)$ at all times $t \in [t_1'', t_2]$.

The first time the algorithm processes color c at a time $t \in \mathcal{E}$ during $[t_1'', t_2]$, it processes at least $\frac{k}{\log^3 k\gamma}$ elements which arrive after t_1' and, therefore, $\overline{n}_c^O(t) \geq \frac{k}{\log^3 k\gamma}$ at this time. Consider any time $t' \in \mathcal{E}$ and the time t'' that is the earliest time in $\mathcal{E}$ after t'. We know that all of the elements contributing to $\overline{n}_c^O(t')$ also contribute to $\overline{n}_c^O(t'')$. All of the elements contributing to $n_c^A(t'')$ arrive during $(t', t'']$, these elements contribute to $\overline{n}_c^O(t'')$ and do not contribute to $n_c^O(t')$. Hence, $\overline{n}_c^O(t'') \geq \overline{n}_c^O(t') + n_c^A(t'')$. Further, we know that $n_c^A(t'') \geq \frac{3}{5} \cdot n_c^O(t'') \geq \frac{3}{5} \cdot \overline{n}_c^O(t'')$. Thus, $n_c^A(t'') \geq \frac{3}{5}(\overline{n}_c^O(t') + n_c^A(t''))$ which implies $n_c^A(t'') \geq \frac{3}{2} \cdot \overline{n}_c^O(t')$, hence $\overline{n}_c^O(t'') \geq \frac{3}{2} \cdot \overline{n}_c^O(t')$. This implies that at time t_2 the LP has $\frac{k}{\log^3 k\gamma}(3/2)^{20 \log\log(k\gamma)} \geq 3k$ elements for color c processed by at most $1/2 + \epsilon$. However, then the LP has a total volume of at least $3k(1/2 - \epsilon) > k$ at time t_2 unprocessed, a contradiction to constraint (2.2) since $\epsilon < 1/100$.

For the second case, assume that there are at least $20 \log\log(k\gamma) + 2$ time steps in $\mathcal{E}$ where the the LP has a set of color blocks for color c of total height greater than $1/2$ that each intersect time t_1. Let S be a set of such color blocks of non-zero height in the LP solution. Let t_2' be the third to last time in $\mathcal{E}$ during $[t_1, t_2]$ where all of these color blocks intersect. Let b_i be a color block from this set, which processes element e_i at time t_1 and i is as large as possible. Notice that since b_i contains two times in $\mathcal{E}$ after t_2' it is the case that at any time t in $\mathcal{E}$ during $[t_1, t_2']$ b_i has not processed any element that the algorithm has in its buffer for color c. Otherwise, b_i could not extend until the second in $\mathcal{E}$ after t_2'. We know that all the color blocks in S will eventually process all elements for color c which are in $\mathcal{B}(t)$ for $t \in [t_1, t_2']$ because the color blocks the LP is allowed to use are assumed to be maximal. Further, every element that b_i will process, which it has not by time t, will be processed by every color block in S after t by definition of b_i. We also know that since each element can be processed by at most 1 in the LP by constraint (2.2) and the total height of all color blocks in S is at least $1/2$, this implies that all elements b_i will process after time t during $[t_1, t_2']$ have not been processed by $1/2$ in the LP solution at time t. Thus, every element which b_i will process after time t during $[t_1, t_2']$ contributes to $n_c^O(t)$.

Let t' be a time in $\mathcal{E}$ during $[t_1, t_2']$ and t'' be the earliest time in $\mathcal{E}$ after t'. We know that the color block b_i must not have processed any element in $\mathcal{B}(t')$ for color c at time t'. Further we know for the algorithm to have $n_c^A(t'')$ elements for color c at time t'', it must be the case that during $(t', t'']$ the algorithm is outputting at least $n_c^A(t'')$ which are not of color c, which implies that there are $n_c^A(t'')$ elements for color c that arrive before t' and b_i have not processed by time t'. As observed before, this implies that there exist $n^A(t'') + n^A(t')$ elements for color c such that LP has processed by at most half by time t'. Hence we derive $n_c^O(t') \geq n^A(t'') + n^A(t')$. We also know that $\frac{5}{3} \cdot n^A(t') \geq n^O(t')$. Thus, $\frac{5}{3} \cdot n^A(t') \geq n^A(t'') + n^A(t')$ and we have that $n_c^A(t') \geq \frac{3}{2} n^A(t'')$. Knowing that at time t_2' the algorithm outputs $k/\log^3 k\gamma$ elements for color c, this implies that at time t_1 the algorithm has at least $\frac{k}{\log^3 k\gamma}(3/2)^{20 \log\log(k\gamma)} \geq 3k$ elements for color c in $\mathcal{B}(t_1)$. This contradicts the size of the algorithm's buffer. □

4 Unweighted Reordering Buffer

In this section we study the unweighted version of the Reordering Buffer Management problem, i.e. $w_c = 1$ for all $c \in \mathcal{C}$. We use the same linear programing we used for the weighted case (see Section 2), hence the objective will be simply to minimize $\sum_{b \in B} x_b$. However, we strengthen the LP via knapsack covering inequalities. It will allow us to view the Buffer Management Problem as a covering problem and enable our improved analysis. We first explain the extra constraints we add, and give a high-level overview of our algorithm and analysis. Our algorithm and analysis build on an understanding of a procedure that fills some "holes" random sampling leaves, which may be of independent interest; see Section 4.2. We first show how we strengthen LP and give a high-level overview of our algorithm. We will then presents our algorithm and analysis.

4.1 Linear Programming

The linear programming LP is strengthened by the following constraints.

$$\sum_{b \in B \setminus B'} (|E_{b,\leq t} \setminus E'|) x_b \geq (t -' k - |E'|)(1 - \sum_{b \in B'} x_b) \quad (4.8)$$
$$\forall t \in [k+1, k+n], B' \subseteq B, E' \subseteq E$$

The notation $E_{b,\leq t}$ denotes the elements that are scheduled by block b by time t. What the constraints imply is the following: Consider any integer solution $\{x_b\}$, $b \in B$. For a while consider the case where $B' = \emptyset$. We know that $t - k$ elements are scheduled by time t. Hence without using any set E' of elements, we should be able to find at least $t - k - |E'|$ elements that are scheduled by time t. Now suppose $B' \neq \emptyset$. The left-hand-side is non-negative while the right-hand-side is non-positive, hence the constraints are satisfied. As before, we relax $x_b \in \{0, 1\}$ into $0 \leq x_b \leq 1$. These constraints are essentially knapsack covering inequalities which have been shown to be useful for a variety of covering problems. For example, see [12, 8]. However, the number of these constraints can be exponential in the size of

the number of elements, hence in order to solve LP in polynomial time we will give a separation oracle.

Separation Oracle: The total number of constraints, besides those in (4.8), is at most a polynomial in n. Hence to solve the LP in polynomial time, we need a separation oracle only for constraints (4.8). Unfortunately, we do not have a separation oracle for constraints (4.8). However, we do not need an "exact" separation oracle since we will need to satisfy constraints (4.8) only for a specific set E' (which can vary depending on x_b for a fixed time).

LEMMA 4.1. *Suppose that we are given $0 \leq x_b \leq 1$ that satisfy all constraints in* LP *possibly except some in (4.8). Then consider any $t \in [k+1, k+n]$, and let $E' = \{e_i \mid \bar{y}_{i,t} \geq \rho, i \leq t'\}$ for any time t' and some constant $0 < \rho \leq 1$. Then if there exists a violated constraint for some B' in (4.8) we can find it in polynomial time.*

Proof. Fix time t and t'. Since E' is fixed we only need to consider arbitrary $B' \subseteq B$. Note that $|E_{b,\leq t'} \setminus E'|$ and $t - k - |E'|$ are now fixed. By rearranging terms, we have

$$\begin{aligned} & \Big(\sum_{b \in B} |E_{b,\leq t} \setminus E'|)x_b\Big) - (t - k - |E'|) \\ \geq & \sum_{b \in B'} \Big((|E_{b,\leq t} \setminus E'|) - (t - k - |E'|)\Big)x_b, \end{aligned}$$

which is equivalent to (4.8). Note that the left-hand-side is fixed. Hence if there exists any violated constraint in (4.8), the constraint for B' that maximizes the right hand side must be a violated one. Also it is easy to see that the right-hand-side is maximized when $b \in B'$ if and only if $(|E_{b,\leq t} \setminus E'|) - (t - k - |E'|) \geq 0$. If the right-hand-side is greater than the left-hand-side for such B', we have found a violated constraint. Otherwise, there is no violated constraint in (4.8) for time t.

Using this separation oracle, we can find a solution to LP that is good enough for our purpose.

COROLLARY 4.1. *One can find a* LP *solution x_b^* in polynomial time that satisfies the following:*

- *$\sum x_b^*$ is no greater than $\mathsf{Cost}_{\mathsf{LP}}$, the cost of the optimal solution to* LP.
- *For any t, t' and $E' := \{e_i \mid \bar{y}_{i,t} \geq \rho, i \leq t'\}$, all constraints (4.8) are satisfied.*
- *All other constraints in* LP *are satisfied.*

4.2 Overview of Algorithm and Analysis As mentioned before, our algorithm crucially relies on viewing the Buffer Management Problem as a covering problem. To have a feel of what we mean by a covering problem, suppose we have a pool Pool of color blocks where each element appears in at least one color block in Pool. Then we can associate each element e_i with the earliest time step when e_i is scheduled in a color block in Pool. Say that element e_i is ready (to be scheduled) at time t if it is associated with the time step t or earlier. We observe that if there is always an element ready for schedule in our buffer (or equivalently that we haven't scheduled) then the cost of the schedule is at most the number of color blocks in Pool. This is because when we schedule an element e_i ready for schedule, we know that there is a block in Pool that schedules e_i, so it must be case that we schedule all elements (not scheduled yet) in b with element e_i.

Hence to have a low cost schedule, it is essential to construct a set of color blocks Pool that has a small number (cost) of color blocks, and that covers all time steps by 'ready-for-schedule' elements. More precisely, for any time $t \geq k + 1$, if there are at least $t - k$ ready-for-schedule elements by time t, and the Pool size is small, we will be in good shape. Then how can we construct such a pool? As observed in previous work, one can always schedule an element that has been processed by LP substantially. Intuitively, the resulting blocks can be compared to $\mathsf{Cost}_{\mathsf{LP}}$ since LP must have processed some elements in such blocks substantially (See Lemma 2.1). However, if LP schedules many colors fractionally, such 'above-threshold' covering will leave some times where there is no element available. We can think of such times as being 'holes'.

Using the α-sampling we already used for the weighted case, the algorithm will likely find elements which are ready for schedule. However, there can be still some uncovered time steps. In this case, roughly speaking, we will try to schedule a color with the most elements in the current buffer. Ideally we would like to bound the number of such color blocks since they did not come from sampling and hence are hard to compare to the LP cost. Our observation is that after the threshold rounding and α-sampling, the number of additional color blocks needed to fill the remaining holes is not many. Here the tricky part is that the elements of the same color behave dependently. Nevertheless, we show the following "Filling the Gap" lemma which may be of potential use for other covering problems. The proof will be presented in Section 4.4.1.

LEMMA 4.2. *Consider a collection of independent random variables Z_l, $1 \leq l \leq \ell$ where Z_l can take an integer value in $[0, L]$ for a positive integer L. Let $\max Z_l$ denote the maximum value that Z_l can take. Suppose that $\sum_{l \in \ell} \mathbb{E}[Z_l] \geq \eta L$ for some constant $\eta > 1$. Let S denote the minimum subset of indices with highest $\max Z_l$ such that $\sum_{l \in S} \max Z_l + \sum_{l \notin S} Z_l \geq L$. Then it follows that $\mathbb{E}[|S|] \leq g(\eta)$, where $g(\eta) := 1/\Big(1 - 1/\exp(2(1 - 1/\sqrt{\eta})^2)\Big)$.*

Roughly speaking the above lemma says that if the expected number of elements that are ready to be scheduled

is large (by more than η factor), then by adding elements of only a constant number of colors in expectation, one can meet the demand.

Our algorithm and analysis look involved, however, the underlying covering view and sampling ideas are fairly intuitive. The large volume of algorithm description and analysis is mostly on combining the above three main ideas, threshold rounding, α-sampling and "Filling-the-Gap." We feel that our algorithm and analysis are fairly different from the previous work on the Buffer Management problem, and could give a more direct view that interprets the problem as a covering problem, thereby making other analysis tools more accessible.

4.3 Algorithm For each element we will define time steps when it becomes available for schedule. Intuitively, the time is when it is scheduled by a color block we sample. More specifically, we will associate each element e_i with possibly four different time steps, t_i^ρ, t_i^α, t_i^β, and t_i^σ, which we call element e_i's ρ, α, β, and σ-times, respectively. All elements have t_i^ρ and t_i^α, but some elements may not have t_i^β or t_i^σ. Also t_i^ρ and t_i^α may induce times $t_i^{\rho'}$, $t_i^{\alpha'}$, respectively. We say that an element e_i is ready for schedule at time t if its ρ, ρ', α, α', β, or σ-time is no greater than t. Once we fix these time steps our algorithm becomes straightforward to describe. As before, once we start a new color, we continue to schedule elements of the same color until we run out of such elements. Also for the same color, we always schedule elements in first-in-first-out order. Scheduling a color means scheduling an element of the color. The ρ-ready elements have the highest priority and the σ-ready elements have the lowest priority.

Algorithm: Algorithm–Unweighted (A_u)

At each time t, schedule a color of an element in the buffer that is ready at the time t while prioritizing elements e_i by ρ, α, ρ', α', β and σ-times as in this order.

We will ensure there is at least one element ready for schedule at any time so that the algorithm never gets stuck. We say that element e_i is ρ-ready if $t_i^\rho \le t$. Similarly we define ρ', α, α', β, σ-ready times. We now describe how to associate each element with these times.

Setting ρ-times, t_i^ρ: This is the simplest part. For each element e_i, we let t_i^ρ denote the first time t when LP schedules element e_i by time t by at least ρ, i.e. $\bar{y}_{i,t} \ge \rho$. The constant $0 < \rho < 1$ will be fixed later.

Setting α-times, t_i^α: We use the α-sampling we introduced in Section 2. This defines t_i^α time for each element e_i. Later, α will be set to be equal to ρ.

We now explain how to set β and σ-times, and also ρ' and α'-times. We will define phases by partitioning the time horizon. We note that an element may have a different β, σ, ρ', α' for each phase. However, for notational simplicity, we will mostly drop the notation denoting the phase. Also our analysis will focus on each phase separately.

Setting β, σ-times, t_i^β, t_i^σ: Partition the time horizon $[k+1, n+k]$ by time steps, $k = \tau_0 < \tau_1 < \tau_2 < \tau_3 ...$, where τ_j is defined as the earliest time after τ_{j-1} when the LP cost increases by at least Δ where $0 < \Delta < 1$ is a constant which will be fixed later. For each color block b in LP, we increase the cost of LP by its height x_b at the time when it *ends*. Note that this is the opposite of the weighted case where we associate the cost of color blocks with the time it starts in the LP. We refer to the interval $I_j := (\tau_{j-1}, \tau_j]$ as phase j. When we say that we charge to phase j, we mean that we charge to the increase of at least Δ in LP cost during I_j. Now consider any j, and the latest time step $t_1^j \in (\tau_{j-1}, \tau_j]$ such that there are at least $t-k$ ρ-ready elements at each time t during $(\tau_{j-1}, t_1^j]$. If t_1^j does not exist, set $t_1^j = \tau_{j-1}$. Note that the total height of color blocks $\mathcal{I}'_j$ that end during (τ_{j-1}, τ_j) is at most Δ. Let $t_2^j \ge t_1^j$ be the earliest time step when one of the color blocks $\mathcal{I}_j$ that intersect τ_j, schedules only the elements arriving after time t_1^j, during $(t_2^j, \tau_j]$. If t_2^j does not exists, set $t_2^j = \tau_j$. Note that if $t_2^j < \tau_j$ then there must exist a color block in $\mathcal{I}_j$ that since time t_2^j schedules only the elements arriving after time t_1^j. Fix one such color block and let σ_j denote this block. Now if σ_j schedules an element e_i at time t', then we set $t_i^\sigma = t'$.

We now set β-times. The goal of this step is to find enough elements to schedule by time t_2^j without using elements arriving after t_1^j. Later we will show that we only need to use a constant number of colors C_j^β in expectation, and this will be the most interesting part of our algorithm and analysis. Let $C_j^\beta = \emptyset$. We add a color c to C_j^β with the maximum number of elements arriving by time t_1^j until we have that,

$$\begin{aligned} & \sum_{c \in C_j^\beta} |E_{c, \le t_1^j}| \\ & + |\{e_i \in E_{\le t_1^j} \mid c(e_i) \notin C_j^\beta \text{ and } \min\{t_i^\rho, t_i^\alpha\} \le t_2^j\}| \\ \ge \;& t_2^j - k. \end{aligned}$$

Here $E_{c, \le t_1^j}$ denotes the set of elements arriving by time t_1^j for color c, and $E_{\le t_1^j}$ denotes the set of elements arriving by time t_1^j (for any color). Then for each element e_i with color $c(e_i)$ in C_j^β, that arrives by time t_1, we set $t_i^\beta = t_1^j$. As mentioned before, each element may have a different β, σ-time in each phase.

Setting ρ', α'-times, $t_i^{\rho'}$, $t_i^{\alpha'}$: It now remains to set ρ', α'-times. If there exists any element $e_i \in E_{\leq t_1^j}$ that becomes ρ-ready during $(t_1^j, t_2^j]$ for the first time, we let all elements of color $c(e_i)$ arriving by time t_1^j to have ρ'-ready time t_1^j, and we add $c(e_i)$ to $C_j^{\rho'}$. We will refer to the elements with ρ'-ready time t_1^j as ρ'-ready elements in phase j, and add them to $E_j^{\rho'}$. Also if there exists any element $e_i \in E_{\leq t_1^j}$ that becomes α-ready during $(t_1^j, t_2^j]$ for the first time, we let all elements of color $c(e_i)$ arriving by time t_1^j to have α'-ready time t_1^j, and add $c(e_i)$ to $C_j^{\alpha'}$, and add all those elements to $E_j^{\alpha'}$. We refer to the elements with α'-ready time t_1^j as α'-ready elements in phase j. We note that some colors my appear both in $C_j^{\rho'}$ and $C_j^{\alpha'}$. As mentioned before an element may have a different ρ', α'-time in each phase.

4.4 Analysis We will say that an element e_i is ready for schedule at time t if its ρ, ρ', α, α', β or σ time is not greater than time t. We first show that the algorithm never gets stuck. Our analysis will be done for each phase $I_j = (\tau_{j-1}, \tau_j]$. For notational convenience, we will refer to t_1^j, t_2^j simply as t_1, t_2.

LEMMA 4.3. *Suppose $\rho \leq 1 - \Delta$, then at any time t, there is at least one element ready to be scheduled.*

Consider any j. By definition of t_1, we know that there is at least one ρ-ready element in the buffer during $(\tau_{j-1}, t_1]$. Also we know that using the block σ_j, we can find one σ-ready element arriving after t_1 in the buffer at each time during $(t_2, \tau_j]$. Hence if we can find $t_2 - k$ elements that become ρ, ρ', α, α' or β-ready by time t_1 (and that arrive by time t_1), then we will be done.

We note that Lemma 4.3 is not completely obvious since LP may schedule some elements arriving after t_1 (by a small amount though) during $(t_1, t_2]$, and our goal to find enough elements arriving by t_1 to cover all time steps by time t_2. Here constraints (4.8) play a crucial role. We use the constraint by setting E' to be the elements that are ρ-ready by time t_2 and that arrive by time t_1, and by setting $B' = \mathcal{I}'_j$; see Corollary 4.1. Knowing that the total height of color blocks $\mathcal{I}'_j$ (that end during (τ_{j-1}, τ_j)) is at most Δ, we have

$$\sum_{b \in B \setminus \mathcal{I}'_j} |E_{b,\leq t_2} \setminus E'| \cdot x_b \geq (1-\Delta)\Big(t_2 - k - |E'|\Big) \tag{4.9}$$

Note that the volume of work in the left-hand-side is only on the elements that arrive before time t_1, and do not become ρ-ready by time t_2. This is because by definition of t_2^j, the only possible color blocks that can work on elements arriving after time t_1 during $(t_1, t_2]$ are all in $\mathcal{I}'_j$, and we discarded them by setting $B' = \mathcal{I}'_j$. Hence we know that each element contributing to the left-hand-side has been processed by the LP by at most ρ. Thus, we know that there are at least $(1-\Delta)(t_2 - k - |E'|)/\rho$ elements that are scheduled by LP that arrive before time t_1 that do not become ρ or ρ'-ready by time t_1 when $(1-\Delta)/\rho \geq 1$. By the way we defined β-time steps, we conclude that Lemma 4.3 holds.

We now upper bound the (expected) cost of our solution. We say that a color block b is ρ, α, β, σ, ρ', or α'-block if it is initiated by ρ, α, β, σ, ρ', α' times respectively. Bounding the cost of ρ, σ blocks is easy. We will abuse the notation ρ' and let ρ' also denote a constant such that $0 < \rho' < \rho$, which will be fixed soon.

Note that the following lemma bounds the total cost of ρ-blocks.

LEMMA 4.4. *The total cost (number) of color blocks (including ρ-blocks) that schedule an element that* LP *has processed by at least ρ' is at most $\frac{1}{\rho'}\mathsf{Cost}_{\mathsf{LP}}$.*

Proof. The proof immediately follows from Lemma 2.1.

LEMMA 4.5. *The total expected cost (number) of α-color blocks is at most $\frac{1}{\alpha}\mathsf{Cost}_{\mathsf{LP}}$.*

Proof. The proof immediately follows from Lemma 2.2.

LEMMA 4.6. *The total cost (number) of σ-blocks is at most $\frac{1}{\Delta}\mathsf{Cost}_{\mathsf{LP}}$.*

Proof. Note that there is at most one color block σ_j for each phase j. We charge this cost to the LP cost increase in phase j. To see the details, say that element e_i is scheduled by σ_j. Then if there is any σ-block that is started by one of these elements σ_j, it must continue to schedule all elements in σ_j. Hence σ_j can result in at most one σ-block.

LEMMA 4.7. *The total expected cost (number) of ρ' and α'-blocks that do not schedule an element processed by more than ρ' is at most*

$$\frac{\Delta+1}{\Delta} \cdot \frac{1}{\rho - \rho'} \mathsf{Cost}_{\mathsf{LP}}$$

Proof. Consider any ρ'-block that is scheduled by a ρ'-ready element e_i in phase j. Since it has not been processed by LP by more than ρ' when it is scheduled, there exists an element of the color that is processed during $(t_1, t_2]$ by at least $\rho - \rho'$. In other words, color $c(e_i) \in C_j^{\rho'}$ used at least $\rho - \rho'$ height during $(t_1, t_2]$. Let $h^{\rho'}$ be the total height of color blocks of a color in $C_j^{\rho'}$. Then we know that the total number of ρ'-blocks that meet the description of the lemma is at most $\frac{h^{\rho'}}{\rho - \rho'}$. This is because there can be at most one ρ'-color block of the same color in each phase (all ρ'-ready elements in phase j have the same ρ'-ready time t_1).

Also consider any α'-block that is scheduled by a α'-ready element e_i in phase j. Since it is not ρ-ready by time t_2, for it to be a α'-ready element in phase j, a color block that intersects a time in $(t_1, t_2]$ and schedules the element must have been sampled by α-sampling. And we know that the expected number of color blocks sampled out of $\mathcal{I}_j \cup \mathcal{I}'_j$, the set of color blocks that intersect a time in $(t_1, t_2]$, is at most $h^{\alpha'}/\alpha$ where $h^{\alpha'}$ denote the total height of color blocks of a color not in $C_j^{\rho'}$.

Note from definition of τ_j that the total height of $\mathcal{I}_j \cup \mathcal{I}'_j$ is at most $(1+\Delta)$, hence we have $h^{\rho'} + h^{\alpha'} \leq 1 + \Delta$. We charge the total expected cost of

$$\frac{h^{\rho'}}{\rho - \rho'} + \frac{h^{\alpha'}}{\alpha} \leq \max\{\frac{1}{\rho - \rho'}, \frac{1}{\alpha}\} \cdot (1 + \Delta)$$

to the LP cost increase in phase j. Since $\alpha \geq \rho$, the lemma follows.

To bound the cost of β-blocks we need to understand the underlying random process behind the construction of C_j^{β}. Lemma 4.2 will be crucial for our analysis. The lemma upper bounds the expected number of "corrections" (additional colors) needed to meet the given demand. Now let us take a close look at (4.9) again. The set E' includes all elements that arrive by time t_1 and ρ-ready by time t_2, or equivalently all elements that are ρ or ρ'-ready time at t_1. As discussed already, (4.9) implies that there is enough volume of work that has been done on the elements that arrive by time t_1 and that do not become ρ-ready by time t_2. Hence for each color c, we create a random variable Z_c that denotes the number of elements of color c that arrive by time t_1 and that do not become ρ-ready by time t_2, but become α-ready by time t_2. Let $L := t_2 - k - |E'|$. By (4.9) and Lemma 2.2, then we have

$$\sum_c \mathbb{E}[Z_c] \geq \eta \cdot L \qquad \text{where } \eta = \tfrac{(1-1/e)(1-\Delta)}{\rho}.$$

Then by applying Lemma 4.2 we have that $\mathbb{E}|C_j| \leq g(\frac{(1-1/e)(1-\Delta)}{\rho})$. Since one can schedule all elements of each color $c \in C_j$ that arrive by time t_1 in one color block since t_1, by charging the cost $\mathbb{E}[|C_j|]$ to phase j, we have

LEMMA 4.8. *The total expected cost (number) of β-blocks is at most $\frac{1}{\Delta} g(\frac{(1-1/e)(1-\Delta)}{\alpha})$.*

By combining Lemma 4.5, 4.6, 4.7, 4.8, we conclude that the approximation ratio of our algorithm is at most

$$(\frac{1}{\rho'} + \frac{1}{\alpha} + \frac{1}{\Delta} + \frac{\Delta+1}{\Delta} \cdot \frac{1}{\rho - \rho'}) + \frac{1}{\Delta} g(\frac{(1-1/e)(1-\Delta)}{\alpha})$$

We set $\alpha = 0.19, \Delta = 0.45, \rho = 0.19, \rho' = 0.0678$, and the quantity has value 66.0823. This proves Theorem 1.2.

4.4.1 Proof of Lemma 4.2 This section is devoted to proving Lemma 4.2. We partition random variables into two groups depending on the ratio of their expected value to their maximum value. All random variables Z_l such that $\max Z_l \geq \zeta \mathbb{E} Z_l$ are put into $\mathcal{Z}^{big}$, where $\zeta > 1$ is a constant to be fixed later. All other random variables are put into $\mathcal{Z}^{small}$. (For notational convenience, we let $\mathcal{Z}^{big}$ and $\mathcal{Z}^{small}$ refer to the indices of the random variables they have.) Now we scale down the non-zero probability of each random variable $Z_l \in \mathcal{Z}^{small}$ by a factor of ζ–this decreases $\mathbb{E} Z_l$ by a factor of ζ. Obviously this can only increase the expected number of $|S|$. Note that now we have $\sum_l \mathbb{E} Z_l \geq (\eta/\zeta) L$, and for all Z_l, $\max Z_l \geq \zeta \mathbb{E} Z_l$; η/ζ will be set to be greater than 1.

We use the concentration inequality which can be found in Appendix A . We reindex Z_l such that $\max Z_l$ non-increases in l. To prove Lemma 4.2, we consider the probability that we are forced to add X_l to S. This can happen only if $\sum_{l'<l} \max Z_{l'} + \sum_{l' \geq l} Z_l < L$. Hence we conclude the following:

$$\mathbb{E}|S| \leq \sum_{l \geq 1} \Pr\Big[\sum_{l'<l} \max Z_{l'} + \sum_{l' \geq l} Z_l < L\Big] \tag{4.10}$$

Let $L' = \max Z_l$; note that $L' \geq Z_l, Z_{l+1}, ..., Z_\ell$. Hence

$$\begin{aligned}
&\Pr\Big[\sum_{l' \geq l} Z_{l'} \leq L - \sum_{l'<l} \max Z_{l'}\Big] \\
=\ &\Pr\Big[\sum_{l' \geq l} Z_{l'} \leq \mathbb{E}(\sum_{l' \geq l} Z_{l'}) - \Big(\sum_{l' \geq l} \mathbb{E} Z_{l'} + \sum_{l'<l} \max Z_{l'} - L\Big)\Big] \\
\leq\ &\exp\Big(-\frac{(\sum_{l' \geq l} \mathbb{E} Z_{l'} - L + \sum_{l'<l} \max Z_{l'})^2}{2\sum_{l' \geq l} \mathbb{E} Z_{l'}^2}\Big) \\
&\qquad \text{[By Theorem A.1]} \\
=\ &\exp\Big(-\frac{(\sum_{l'} \mathbb{E} Z_{l'} - L + \sum_{l'<l} (\max Z_{l'} - \mathbb{E} Z_{l'}))^2}{2\sum_{l' \geq l} \mathbb{E} Z_{l'}^2}\Big) \\
\leq\ &\exp\Big(-\frac{(\sum_{l'} \mathbb{E} Z_{l'} - L + (1-1/\zeta)\sum_{l'<l} \max Z_{l'}))^2}{2\sum_{l' \geq l} \mathbb{E} Z_{l'}^2}\Big) \\
&\qquad \text{[Since } \max Z_{l'} \geq \zeta \mathbb{E} Z_{l'}\text{]} \\
\leq\ &\exp\Big(-\frac{((1-\frac{\zeta}{\eta})\sum_{l'} \mathbb{E} Z_{l'} + (1-1/\zeta)\sum_{l'<l} \max Z_{l'}))^2}{2\sum_{l' \geq l} \mathbb{E} Z_{l'}^2}\Big) \\
&\qquad \text{[Since } \textstyle\sum_{l'} \mathbb{E} Z_{l'} \geq \frac{\eta}{\zeta} L\text{]} \\
\leq\ &\exp\Big(-\frac{((1-\frac{\zeta}{\eta})\sum_{l'} \mathbb{E} Z_{l'} + (1-1/\zeta)(l-1)L')^2}{2\sum_{l' \geq l} \mathbb{E} Z_{l'}^2}\Big) \\
&\qquad \text{[Since } \max Z_1 \geq ... \geq \max Z_l = L'\text{]} \\
\leq\ &\exp\Big(-\frac{((1-\frac{\zeta}{\eta})\sum_{l'} \mathbb{E} Z_{l'} + (1-1/\zeta)(l-1)L')^2}{2L'\sum_{l' \geq l} \mathbb{E} Z_{l'}}\Big) \\
&\qquad \text{[Since } L' = \max Z_l \geq Z_{l+1} \geq ... \geq Z_\ell\text{]} \\
\leq\ &\exp\Big(-\frac{4(1-\frac{\zeta}{\eta})(1-1/\zeta)(l-1)L'\sum_{l'} \mathbb{E} Z_{l'}}{2L'\sum_{l' \geq l} \mathbb{E} Z_{l'}}\Big) \\
&\qquad \text{[Since } (a+b)^2 \geq 4ab \text{ for } a, b \geq 0\text{]}
\end{aligned}$$

$$\leq \exp\Big(2(1-\frac{\zeta}{\eta})(1-1/\zeta)(l-1)\Big)$$

Hence from (4.10) we have

$$\begin{aligned}\mathbb{E}|C| &\leq \sum_{l\geq 1}\exp\Big(2(1-\frac{\zeta}{\eta})(1-\frac{1}{\zeta})(l-1)\Big)\\ &= \frac{1}{1-1/\exp(2(1-\frac{\zeta}{\eta})(1-\frac{1}{\zeta}))}\\ &= \frac{1}{1-1/\exp(2(1-\frac{1}{\sqrt{\eta}})^2)}\end{aligned}$$

The last equality follows by setting $\zeta = \sqrt{\eta}$.

References

[1] A. Aboud. Correlation clustering with penalties and approximating the reordering buffer management problem. Masters 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 *STOC*, pages 607–616, 2011.

[3] Houman Alborzi, Eric Torng, Patchrawat Uthaisombut, and Stephen Wagner. The k-client problem. *J. Algorithms*, 41(2):115–173, 2001.

[4] Yuichi Asahiro, Kenichi Kawahara, and Eiji Miyano. Np-hardness of the sorting buffer problem on the uniform metric. *Discrete Applied Mathematics*, 160(10-11):1453–1464, 2012.

[5] Noa Avigdor-Elgrabli and Yuval Rabani. An improved competitive algorithm for reordering buffer management. In *SODA*, pages 13–21, 2010.

[6] Noa Avigdor-Elgrabli and Yuval Rabani. A constant factor approximation algorithm for reordering buffer management. In *SODA*, 2013.

[7] Noa Avigdor-Elgrabli and Yuval Rabani. An optimal randomized online algorithm for reordering buffer management. *CoRR*, 1303.3386, 2013.

[8] Nikhil Bansal, Anupam Gupta, and Ravishankar Krishnaswamy. A constant factor approximation algorithm for generalized min-sum set cover. In *SODA*, pages 1539–1545, 2010.

[9] Reuven Bar-Yehuda and Jonathan Laserson. Exploiting locality: approximating sorting buffers. *J. Discrete Algorithms*, 5(4):729–738, 2007.

[10] Siddharth Barman, Shuchi Chawla, and Seeun Umboh. A bicriteria approximation for the reordering buffer problem. In *ESA*, pages 157–168, 2012.

[11] Daniel K. Blandford and Guy E. Blelloch. Index compression through document reordering. In *DCC*, pages 342–351, 2002.

[12] Robert D. Carr, Lisa K. Fleischer, Vitus J. Leung, and Cynthia A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *SODA*, pages 106–115, 2000.

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

[14] Fan Chung and Linyuan Lu. *Complex Graphs and Networks (Cbms Regional Conference Series in Mathematics)*. American Mathematical Society, Boston, MA, USA, 2006.

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

[16] Matthias Englert and Matthias Westermann. Reordering buffer management for non-uniform cost models. In *ICALP*, pages 627–638, 2005.

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

[18] Rohit Khandekar and Vinayaka Pandit. Online sorting buffers on line. In *STACS*, pages 584–595, 2006.

[19] Jens S. Kohrt and Kirk Pruhs. A constant approximation algorithm for sorting buffers. In *LATIN*, pages 193–202, 2004.

[20] Jens Krokowski, Harald Räcke, Christian Sohler, and Matthias Westermann. Reducing state changes with a pipeline buffer. In *VMV*, page 217, 2004.

[21] Harald Räcke, Christian Sohler, and Matthias Westermann. Online scheduling for sorting buffers. In *ESA*, pages 820–832, 2002.

A Concentration Inequality

The following theorem can be found in Chapter 2 in [14].

THEOREM A.1. *Let X_1, X_2, ..., X_n be non-negative independent random variables, then we have the following bound for the sum $X = \sum_{i=1}^{n} X_i$.*

$$\Pr[X \leq \mathbb{E}X - \lambda] \leq \exp\Big(-\frac{\lambda^2}{2\sum_{i=1}^{n}\mathbb{E}(X_i^2)}\Big)$$

B Pseudocode

Algorithm: `Accumulate`(t)

`Rule (i)`..
if There is an element $e_i \in \mathcal{B}(t)$ that is completed by ϵ by the LP at time t **then**
  Switch to color $c(e_i)$ for any such element e_i
end if
`Rule (ii)`..
if There is an element $e_i \in \mathcal{B}(t)$ that is α ready at time t **then**
  Switch to color $c(e_i)$ for any such element e_i
end if
`Rule (iii)`..
if There is a color c where $n_c^A(t) \geq k/10$ **then**
  Switch to color c
end if
`Rule (iv)`..
if There is an element $e_i \in \mathcal{B}(t)$ such that $c(e_i) \in M(t)$ **then**
  Switch to color $c(e_i)$ for any such element e_i
  Set $M(t) = M(t) \setminus \{c(e_i)\}$ // unmark color $c(e_i)$
end if
if For every element $e_i \in \mathcal{B}(t)$, $\phi_{c(e_i)} = 0$ **then**
  Set $j^* = \texttt{argmax}_j |\overline{G}_j(t)|$
  Let S be the set of colors corresponding to elements in $\overline{G}_{j^*}(t)$
  For color c let t_c be the earliest time after t such that $t_c \in \mathcal{T}_c$
  Sort the colors in S, $c_1, c_2, \ldots c_{|S|}$ such that $t_{c_l} \leq t_{c_{l+1}}$
  Let j' be the smallest index such that $\sum_{a=1}^{j'} n_{c_a}^A(t) \geq \sum_{a=1}^{|S|} n_{c_a}^A(t)/2$
  Set $\phi_{c_a} = 1$ for all colors c_a where $a \leq j'$
  Add all colors in S to $M(t)$
  Go to line (13)
else
  Let $C_s(t)$ contain all colors c where $0 < n_c^A(t) \leq \frac{k}{\log^3 k\gamma}$
  Let $C_b(t)$ 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 which are not in $\mathcal{B}(t)$, i.e. $E^O(t) := \{e_i \notin \mathcal{B}(t) \mid i \leq t, \bar{y}_{i,t} \leq 1/2 + 2\epsilon\}$.
  if The LP has processed elements in $\mathcal{B}(t)$ corresponding to colors in $C_s(t)$ by a total of at least $|E^O(t)|/8$ by time t **then**
    `Rule (v)`..
    For color c let t_c be the earliest time after t such that $t_c \in \mathcal{T}_c$
    Switch to the color $c \in C_s(t)$ at time t such that t_c is minimized
  else
    `Rule (vi)`..
    Switch to a color $c \in C_b(t)$ such that $n_c^A(t) \geq \frac{3}{5} n_c^O(t)$ // We will show that such a color exists
  end if
end if