SIAM J. COMPUT.
Vol. 51, No. 3, pp. 701–722

ALMOST TIGHT BOUNDS FOR REORDERING BUFFER MANAGEMENT*

ANNA ADAMASZEK[†], ARTUR CZUMAJ[‡], MATTHIAS ENGLERT[‡], AND HARALD RÄCKE[§]

Abstract. We give almost tight bounds for the online reordering buffer management problem on the uniform metric. Specifically, we present the first nontrivial lower bounds for this problem by showing that deterministic online algorithms have a competitive ratio of at least $\Omega(\sqrt{\log k/\log\log k})$ and randomized online algorithms have a competitive ratio of at least $\Omega(\log\log k)$, where k denotes the size of the buffer. We complement this by presenting a deterministic online algorithm for the reordering buffer management problem that obtains a competitive ratio of $O(\sqrt{\log k})$, almost matching the lower bound. This improves upon an algorithm by Avigdor-Elgrabli and Rabani that achieves a competitive ratio of $O(\log k/\log\log k)$.

Key words. online algorithms, reordering buffers, scheduling

AMS subject classifications. 68Q25, 68W27

DOI. 10.1137/20M1326167

1. Introduction.

In the *reordering buffer management problem* a stream of colored items arrives at a service station and has to be processed. The cost for servicing the items depends heavily on the processing order: servicing an item with color c, when the most recently serviced item had color $c' \neq c$, incurs a context switching cost w_c.

In order to reduce the total processing cost, the servicing station is equipped with a reordering buffer able to store k items. This buffer can be used to reorder the input sequence in a restricted fashion to construct an output sequence with a lower processing cost. At each point in time, the buffer contains the first k items of the input sequence that have not yet been processed. A scheduling strategy has to decide which item to service next. Upon its decision, the corresponding item is removed from the buffer and serviced, while the next item from the input sequence takes its place in the buffer.

This simple and versatile framework has many important applications in areas like production engineering, computer graphics, storage systems, and information retrieval, among others [8, 11, 20, 26, 27]. We give two examples.

In the paint shop of a car manufacturing plant, switching colors between two consecutive cars induces nonnegligible cleaning and setup costs. Therefore, paint shops are preceded by a reordering buffer (see [20]) to reorder the stream of incoming

*Received by the editors March 19, 2020; accepted for publication (in revised form) January 10, 2022; published electronically May 24, 2022. A preliminary version of this paper appeared at the 43rd Annual ACM Symposium on Theory of Computing (STOC 2011) and was part of the first author's PhD thesis.
https://doi.org/10.1137/20M1326167
Funding: The work is supported by the Engineering and Physical Sciences Research Council (EPSRC) under grants EP/D063191/1 and EP/F043333/1. The first author is supported by the Danish Council for Independent Research under a DFF-MOBILEX mobility grant.
[†]SimCorp, Copenhagen, Denmark (a.m.adamaszek@gmail.com).
[‡]Department of Computer Science and Centre for Discrete Mathematics and its Applications (DIMAP), University of Warwick, Coventry CV4 7AL, UK (a.czumaj@warwick.ac.uk, m.englert@warwick.ac.uk).
[§]Department of Computer Science, Technische Universität München, Munich, Germany (raecke@in.tum.de).

cars into a stream with a lower number of color changes. This setting is modeled by the reordering buffer framework with *uniform costs*, i.e., $w_c = 1 \; \forall$ colors c.

In a 3D graphic rendering engine [26], a change in attributes between two consecutively rendered polygons slows down the graphics processing unit (GPU), as, for instance, the shader program needs to be replaced. A reordering buffer can be included between application and graphics hardware in order to reduce such state changes. This setting may be modeled by the reordering buffer framework with *nonuniform cost*. Nonuniform costs are required as the cost for a state change depends on the size of the program that has to be loaded.

In this paper we focus on the *online version* of the reordering buffer management problem, in which when the buffer becomes full, one has to decide which item to service next, without knowing the rest of the input sequence. The cost of an online algorithm is compared to the cost of an optimal offline strategy that knows all items in the input sequence in advance and may use the buffer of size k to reorder these items. The worst case ratio between the cost of the online algorithm and the cost of an optimal offline algorithm is called the *competitive ratio*. While we focus mainly on the uniform case our deterministic algorithm and its analysis generalize to the nonuniform case. The competitive ratio then also depends on $\gamma = \max w_c / \min w_c$, the maximum ratio between the context switching cost of any two colors.

1.1. Related work. Table 1 presents an overview of the results for the reordering buffer management problem.

The reordering buffer management problem was introduced by Räcke, Sohler, and Westermann [27], who developed an $O(\log^2 k)$-competitive online algorithm for the version with uniform costs. Englert and Westermann [16] improved the competitive ratio to $O(\log k)$, and their algorithm is also able to handle nonuniform costs with the same bound. Their proof works in two steps. First, it is shown that an online algorithm with a buffer of size k is constant competitive w.r.t. an optimal offline algorithm with a buffer of size $k/4$. Then, it is shown that an optimal algorithm with a buffer of size $k/4$ only loses a logarithmic factor compared to an optimal algorithm with a buffer of size k.

It was shown in [1] that with this proof technique it is not possible to derive online algorithms with a competitive ratio $o(\log k)$ by presenting an input sequence where the gap between an optimal algorithm with a buffer of size $k/4$ and an optimal algorithm with buffer size k is $\Omega(\log k)$. Nevertheless, Avigdor-Elgrabli and Rabani [8]

Table 1

An overview of the results for the reordering buffer management problem. Results presented in this work are highlighted using bold font. γ is defined as $\max w_c / \min w_c$.

	Online competitive ratio	Offline
Algorithm	$O(\log^2 k)$, uniform cost [27] $O(\log k)$ [16] $O(\log k/\log\log k)$ [8] **$O(\sqrt{\log \gamma k})$, deterministic** $O(\log\log k)$, randomized, uniform [7] $O((\log\log k\gamma)^2)$, randomized [5]	$O(1)$-approx., uniform [6] $O(\log\log k\gamma)$-approx. [21]
Lower bound	**$\Omega(\sqrt{\log k/\log\log k})$, deterministic, uniform** **$\Omega(\log\log k)$, randomized, uniform**	NP-hard, even for uniform cost [13, 4]

were able to go beyond the logarithmic threshold by presenting an online algorithm with a competitive ratio $O(\log k/\log\log k)$ using LP-based techniques.

After a preliminary version of this paper has been published, randomized approximation algorithms with much smaller competitive ratios have been presented. Adamaszek et al. [2] gave a randomized algorithm with a competitive ratio of $O(\log\log k)$ for a slightly different model, called a block operation model. Then, Avigdor-Elgrabli and Rabani [7] presented a randomized algorithm with the same competitive ratio for the reordering buffer management problem with uniform costs. These algorithms are based on online primal-dual LP schemes [12]. For the nonuniform cost model, the competitive ratio of the randomized algorithm depends on the ratio between the maximum and the minimum weight [5].

For the offline problem it was shown by Chan et al. [13] and independently by Asahiro, Kawahara, and Miyano [4] that the problem is NP-hard even for uniform costs. Avigdor-Elgrabli and Rabani [6] gave a constant factor approximation algorithm for the offline problem with uniform costs. For nonuniform costs, the best-known approximation factor is $O(\log\log\log k\gamma)$, where γ is the ratio between the maximum and the minimum weight [21, 22].

There also exists a more general version of the problem, where colors correspond to arbitrary points in a metric space (C, d) and the cost for switching from color c' to color c is the distance $d(c', c)$ between the corresponding points in the metric space. Englert, Räcke, and Westermann [15] considered this more general setting, and they obtained a competitive ratio of $O(\log^2 k \log |C|)$, which has been subsequently improved to $O(\log k \log |C|)$ by Englert and Räcke [14]. Kohler and Räcke [24] present an online algorithm with competitive ratio $O(\log k\gamma)$ in this model. This is not directly comparable to the previous results because the parameters $\log |C|$ and $\log \gamma$ are not comparable.

Khandekar and Pandit [23] and Gamzu and Segev [19] study the problem where the colors correspond to points in a line metric. Colors c' and c are integer points on the line, and the cost for switching from c' to c is $|c' - c|$. This version of the problem is motivated by disk scheduling. Khandekar and Pandit [23] give a randomized $O(\log^2 n)$-competitive online algorithm for n uniformly spaced points on a line. Gamzu and Segev improve this to competitive ratio $O(\log n)$ and show a lower bound of about 2.1547 on the competitive ratio of deterministic online algorithms on the line.

Research has also been done on the maximization version of the problem, where the costmeasure is the number of color changes that the output sequence saved over the unordered input sequence. For this version there exist constant factor approximation algorithms due to Kohrt and Pruhs [25] and Bar-Yehuda and Laserson [9].

The reordering buffer management problem has also been studied in the stochastic setting [18], in the setting of bicriteria approximation [10], and in the setting of online algorithms with advice [3]. A more detailed overview of the results can be found in a recent survey [17].

1.2. Our results. We start by presenting the first nontrivial lower bound on the competitive ratio of online algorithms for the problem. We show in Theorem 2.6 that any deterministic online algorithm for the reordering buffer management problem has a competitive ratio of at least $\Omega(\sqrt{\log k/\log\log k})$ even in the uniform case. For randomized algorithms we are able to construct a lower bound of $\Omega(\log\log k)$ (Theorem 2.10). Before this work, no lower bounds were known, and it was quite conceivable that the existing "algorithms might actually have a much better competitive ratio than what was proven about them, possibly even a constant competitive ratio" [8].

Both our lower bounds can be viewed as constructing a specialized caching instance, in which a page request increases the size of the requested page by some factor. Then we exploit existing results for caching to obtain lower bounds for such caching instances and relate these caching instances to our buffering problem.

We complement the lower bound for deterministic algorithms with a deterministic online algorithm whose competitive ratio nearly matches it in the uniform case. We present a deterministic online algorithm that obtains a competitive ratio of $O(\sqrt{\log(k\gamma)})$ for the nonuniform case in which the ratio between the smallest and the largest weight of a color is polynomially bounded in k. This improves upon the result of Avigdor-Elgrabli and Rabani [8] who obtained a competitive ratio of $O(\log k/\log\log k)$.

All results for the reordering buffer management problem preceding this work used very similar algorithms with only subtle differences between them [8, 16, 27]. The differences between the results were mostly based on the analysis. In contrast, our new result relies on an important modification in the algorithm. In addition to techniques similar to those used in [8, 16, 27], our algorithm also relies on classifying colors according to the number of items of the color in the buffer. Then, the algorithm tries to evict items of a color class that currently occupy a large fraction of the buffer. We use this algorithmic ingredient to reduce the competitive ratio to $O(\sqrt{\log k})$ in our analysis.

This adaption of the algorithm is actually motivated by our lower bound example for deterministic algorithms. In this example, the natural strategy for the online algorithm is to free space by removing colors that are similar in the sense that they roughly occupy the same space inside the buffer. Hence, the insights gained by the lower bound example directly gives rise to an improved algorithm.

2. Lower bounds.

In this section we give lower bounds on the competitive ratio of online algorithms for the reordering buffer management problem with uniform costs. We do this by carefully constructing an input sequence σ for which any online algorithm ONL exhibits a large cost, while the optimum algorithm OPT can process σ with a significantly lower cost.

2.1. Preliminaries.

We first describe the general scheme for constructing σ. For this we introduce parameters α, d, and N_i, whose precise values will be fixed later. For simplicity of notation we assume that k is sufficiently large and chosen in such a way that no rounding issues occur.

2.1.1. The general scheme for constructing $\boldsymbol{\sigma}$.

The input sequence σ that we construct has the property that an optimal algorithm OPT' with a buffer of size $(1+\alpha)k$ can process σ in such a way that the cost of the output sequence equals the total number of different colors contained in σ (i.e., all elements of a single color are output consecutively). This means OPT' is truly optimal for the sequence, and even increasing the size of the buffer further cannot reduce the cost for processing σ.

Later, we will show that an online algorithm with buffer size k will incur a lot more color changes than OPT', while an optimal offline algorithm with buffer size k will only have a slightly larger cost than OPT'. This implies the lower bounds.

To specify the input sequence σ further, we will view the buffer of OPT' as partitioned into d *classes* $C_1, \ldots, C_d$ (see Figure 1). Each class C_i can store $(1+\alpha)k/d$ elements (i.e., each class consists of a $1/d$ fraction of the buffer space of OPT'). We further partition the buffer of each class C_i into N_i *slots*, where each slot can store $s_i = \frac{(1+\alpha)k}{dN_i}$ elements (i.e., each slot consists of a $\frac{1}{N_i}$ fraction of the buffer space of the

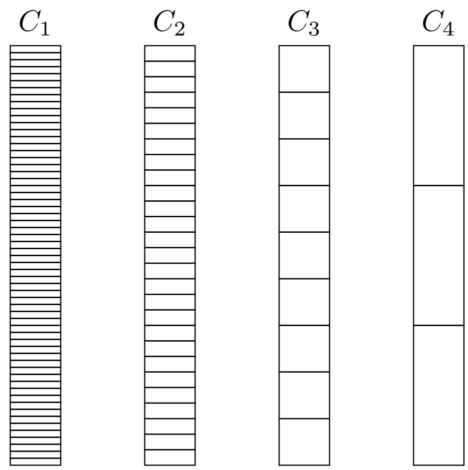

FIG. 1. *Partitioning a buffer space of size $(1+\alpha)k$ into d classes $C_1, \ldots, C_d$ (here $d = 4$). Each class C_i is further partitioned into N_i slots.*

class C_i). The number of slots in a class will be decreasing with the number of the class, i.e., for $i < j$ we have $N_i > N_j$.

The main property of the input sequence σ will be as follows.

> Whenever OPT' has to make a color change while processing σ, the buffer of OPT' looks as follows. Each slot is completely filled with elements of the same color, and different slots contain elements of different colors.

This means, e.g., that if n_c denotes the number of elements with color c at a time right before a color change of OPT', then $n_c = s_i$ for some $i \in \{1, \ldots, d\}$. Because of this property it makes sense to refer to the *color of a slot* as the color that every element in the slot has. Note that the color of a slot may change over time, but in the following proof the time will always be clear from the context.

We obtain the above property by constructing the sequence σ as follows. The initial $(1 + \alpha)k$ elements of σ fill up the buffer of OPT'. They are chosen in such a way that the invariant is satisfied right before OPT' outputs its first element, i.e., among the first $(1 + \alpha)k$ elements of σ there are exactly N_i colors with s_i elements for $i = 1, \ldots, d$. The exact order in which these first $(1 + \alpha)k$ elements appear in σ is not important, but we assume that elements among them that share the same color appear consecutively.

Further elements in σ are chosen in *rounds* in the following way. For each round we choose a slot z_i, $i \in \{1, \ldots, d\}$, from every class (see Figure 2(a)). We add $k + 1$ elements of the color of z_d to σ, which will force any algorithm to switch to this color at this point. Then, for every i, starting from $d - 1$ down to 1, we add $s_{i+1} - s_i$ elements of the color of slot z_i to σ. Finally, we add s_1 elements of a completely new color to σ. This finishes the round.

The algorithm OPT' works as follows. In the beginning of a round it switches to the color of slot z_d. This frees up space s_d in the buffer, and hence, OPT' can hold all further elements appearing in the round without requiring any further color changes.

In order to maintain the main property, we do the following. For $i \leq d - 1$, all s_i elements in the slot z_i plus the $s_{i+1} - s_i$ elements with the same color arriving in

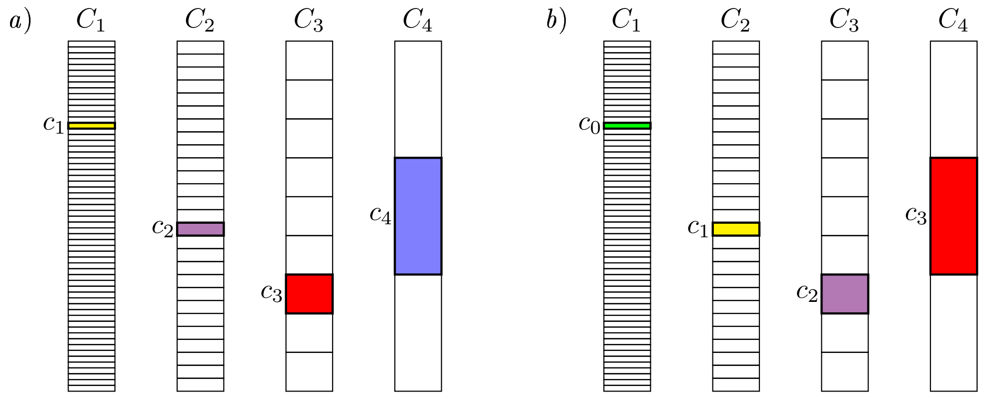

Fig. 2. *One round of creating the sequence σ. (a) The chosen slots and the colors of the elements that are currently occupying the slots. (b) How the contents of the buffer changes after the round. The sequence has been extended as follows: $c_4^{k+1}\ c_3^{s_4-s_3}\ c_2^{s_3-s_2}\ c_1^{s_2-s_1}\ c_0^{s_1}$, where c_0 is the new color introduced in this round.*

the round are moved to the slot z_{i+1} (and completely fill this slot).[1] We say that we *promote slot z_i in class C_i*. Finally, the slot z_1 holds the s_1 elements of the new color (see Figure 2(b)).

This finishes the description of the construction of σ up to the selection of the d slots for each round. Note that regardless of how we choose these slots, it follows from the above discussion that the cost of OPT' for processing σ is equal to the number of different colors in the sequence.

2.1.2. A sketch of the analysis. From the above description it is clear that the cost of OPT' for processing the sequence σ is equal to the number of different colors in the sequence.

However, the online algorithm ONL and the optimal algorithm OPT only have a buffer of size k. Hence, at any time, these algorithms have already removed at least αk elements that are still held by OPT'. Suppose for the time being that these algorithms only remove whole slots (remember that we can simply view a slot as a set of elements that share the same color).

Clearly, if, for example, ONL removed all elements of a slot z_i and this slot is promoted, then ONL will have an additional color change that OPT' does not have. Now, if our aim is to maximize the ratio between the cost of ONL and the cost of OPT', it turns into a caching problem, where in each round the adversary has to pick a slot from every class in such a way that the adversary *hits* many slots that ONL has removed.

For making this idea work we need to show that
- it is more or less optimal for ONL to remove whole slots,
- an adversary can always promote a large number of slots that have been removed by ONL, and
- OPT can handle the resulting input sequence fairly well (as we are not interested in the ratio between the cost of OPT' and ONL but in the ratio between the cost of OPT and ONL).

[1] Observe that the notion of a slot has only been introduced for illustration. Since it is irrelevant where in the buffer something is stored, it is also possible to simply view a slot as the set of all elements of a particular color that are currently stored by OPT'.

2.1.3. The caching framework. In this section we make a formal connection of our problem to the caching-related problem hinted at in the above sketch.

We say that an algorithm ALG *cleared slot* z *before round* r, if right before the time ALG reads the first element of round r, there are no elements from the slot z in the buffer of ALG. This implies that ALG does not hold any elements of the color of slot z. Define the *cost* $\mathrm{cost}^r_{\mathsf{ALG}}$ of an algorithm ALG in round r as the number of slots promoted in round r that are cleared by ALG (i.e., $\mathrm{cost}^r_{\mathsf{ALG}}$ corresponds to the number of *cache misses* in round r).

A lower bound for ONL*.* The following lemma gives a lower bound on the cost of the online algorithm in terms of $\mathrm{cost}^r_{\mathsf{ONL}}$.

LEMMA 2.1. *The total cost of* ONL *on the generated input sequence* σ *is at least*

$$\sum\nolimits_{i=1}^{d} N_i + \sum\nolimits_{r} \mathrm{cost}^r_{\mathsf{ONL}}.$$

Proof. First observe that we can compute the cost of an algorithm by increasing its cost by 1 whenever an element arrives whose color is different from the last color that was appended to the output sequence (called the *active output color*) and also different from all colors present in the buffer.

Initially, the first $(1+\alpha)k$ elements of the input sequence have $\sum_{i=1}^{d} N_i$ different colors, each of them contributing 1 to the cost of the algorithm.

Then, consider a slot z in class C_i for $i \leq d-1$ that is cleared by ONL and promoted in round r. Since ONL cleared z before round r, ONL does not store any elements of the color of slot z in its buffer at the beginning of the round. On the other hand, z is promoted, which means that elements of the color of z appear in σ in round r.

The first $k+1$ elements of round r belong to some color c of a slot in class C_d. After they arrived, the active output color of ONL is c. As the slot z was cleared by ONL before round r, at the time the first element of the color of z appears in σ in round r, the active output color of ONL has to be different from the color of z. Therefore, each such slot z contributes 1 to the total cost of ONL, and there are at least $\mathrm{cost}^r_{\mathsf{ONL}} - 1$ such slots, where the -1 accounts for the slot in class C_d. The cost of ONL is further increased by one in every round r, because of the single element of a completely new color appearing in σ.

By summing over all rounds and taking the first $(1+\alpha)k$ elements into account, it follows that the total cost of ONL is at least $\sum_{i=1}^{d} N_i + \sum_r \mathrm{cost}^r_{\mathsf{ONL}}$. $\square$

An upper bound for OPT*.* In order to give an analogous upper bound on the cost of OPT, we will present specific offline algorithms and analyze their cost. In order to do this analysis in a round-based manner, we further restrict these offline algorithms in the following way. We require that right before a new round, the algorithm has less than $k - s_d$ elements stored in the buffer. To be more specific, we consider algorithms that process a round as follows.

1. Right before the first element of the round appears in σ, the number of elements stored in the buffer is reduced to less than $k - s_d$. This is done by selecting $\alpha N_i + 1$ slots from each class C_i and removing all of the elements in these slots from the buffer.
2. Let $z_1, \ldots, z_d$ denote the slots that are promoted during the round. The algorithm outputs every element of the color of slot z_d. This includes the first $k+1$ elements arriving in the round and also elements of the same color that may be stored in the buffer.

3. Finally, the algorithm stores all other elements arriving during the round in the buffer. This is possible since the number of these elements is s_d.

In order to satisfy the buffer constraint for the first $(1+\alpha)k$ elements of the input sequence, we assume that the algorithm immediately outputs elements from slots that get cleared from the buffer before the first round.

LEMMA 2.2. *Given any offline algorithm* OFF *with the property above, the total cost of* OPT *on the generated input sequence* σ *is at most*

$$\sum\nolimits_{i=1}^{d} N_i + \sum\nolimits_{r} (\mathrm{cost}^r_{\mathsf{OFF}} + 1) \enspace .$$

Proof. The proof is similar to the proof of Lemma 2.1. Again, we observe that we can compute the cost of an algorithm by increasing its cost by 1 whenever an element arrives whose color is different from the last color that was appended to the output sequence and also different from all colors present in the buffer.

Initially, the first $(1+\alpha)k$ elements of the input sequence have $\sum_{i=1}^{d} N_i$ different colors, each of them contributing 1 to the cost of the algorithm OFF. The cost of OFF also increases by 1 in each round due to the single element of a completely new color that appears in every round. The sum of all these costs is $\sum_{i=1}^{d} N_i + \sum_r 1$.

All further increases in the cost of OFF are caused by a sequence of elements arriving in some round r that have a color which is not currently present in the buffer of OFF but which is present in the buffer of OPT'. Such a sequence of elements corresponds to the promotion of a slot z_i, where at the time of the promotion, and therefore also at the time the first element of the round is read from the input, OFF does not store any elements of the slot in the buffer.

In other words, each such increase in the cost of OFF is caused by a promotion of a cleared slot in some round r and therefore also contributes 1 to $\mathrm{cost}^r_{\mathsf{OFF}}$. Hence, $\sum_r \mathrm{cost}^r_{\mathsf{OFF}}$ is an upper bound on such increases to the cost of OFF.

Observing that the cost of OPT is upper bounded by the cost of OFF completes the proof. ∎

2.1.4. Choosing parameters. For the remainder we fix the number of classes as $d :- \log k/(2\log\log k)$ and the size of a slot in class C_i as $s_i := \log^{i-1} k$. The parameter α will be chosen differently depending on whether we want to derive lower bounds for deterministic or for randomized online algorithms. If we deal with deterministic algorithms we choose $\alpha :- \sqrt{\log\log k/\log k}$, otherwise we set $\alpha :- (\log\log k)^2/\log k$.

2.1.5. An important lemma. We now prove a lemma that shows that in the beginning of a round the online algorithm has many *cleared slots* and that these lie in different classes (so that the adversary can choose many of them). This lemma forms the basis of our analysis.

We first require a technical claim that essentially states that for our specific choice of the values of s_i the online algorithm either has a slot cleared or has stored nearly all elements of the slot.

CLAIM 2.3. *For a round* r *and a slot* z *in a class* C_i *at least one of the following is true:*

(a) ONL *cleared slot* z *before round* r.
(b) *The color of slot* z *is equal to the color of the element that* ONL *appended to the output sequence right before reading the first element of round* r.
(c) ONL *holds at least* $\log^{i-1} k - \log^{i-2} k$ *of the* $\log^{i-1} k$ *elements* OPT' *stores in slot* z*, right before* ONL *reads the first element of round* r.

Proof. Consider a slot z in a class C_i. There are only two possible reasons that z is not cleared (i.e., we are not in case (a)), but ONL does not store all $\log^{i-1} k$ elements of z. Either z is the active output color, which means that ONL is in the process of removing elements from z (case (b)), or some elements of z have been previously removed by ONL.

In the latter case some elements have arrived after the removal, as otherwise z would be cleared. However, the last sequence of elements with the color of slot z was the sequence of length $\log^{i-1} k - \log^{i-2} k$. All these elements must still be in the buffer of ONL. This means we are in case (c). This proves the claim. ∎

Using the claim, we can now prove the desired bounds on the number of slots cleared by ONL.

LEMMA 2.4. *Let ℓ_i be the number of slots from class C_i cleared by* ONL *before round r. The following holds:*

(a) $\sum_{i=1}^{d} \ell_i \log^{i-1} k \geq \frac{\alpha k}{2}$.

(b) *At least $\alpha d/4$ of the values ℓ_i are not 0. In other words, at least $\alpha d/4$ different classes contain at least one cleared slot.*

Proof. At the beginning of a round there must exist at least αk elements that ONL has already removed from its buffer while they are still held by OPT'.

Due to Claim 2.3, every slot of class C_i that is not cleared by ONL before round r and whose color is not the active output color of ONL contains at least $\log^{i-1} k - \log^{i-2} k$ elements. Hence, the number of elements that are held by OPT' but not by ONL is at most

$$\log^{d-1} k + \sum_{i=1}^{d} (N_i - \ell_i) \log^{i-2} k + \sum_{i=1}^{d} \ell_i \log^{i-1} k,$$

where the first term accounts for the active output color of ONL and the second term accounts for the possible excess elements of OPT' in slots that are not cleared. This, however, has to be at least αk. We get

$$\begin{aligned}
\alpha k &\leq \log^{d-1} k + \sum_{i=1}^{d} (N_i - \ell_i) \log^{i-2} k + \sum_{i=1}^{d} \ell_i \log^{i-1} k \\
&\leq \frac{k}{\log k} + \frac{(1+\alpha)k}{\log k} + \sum_{i=1}^{d} \ell_i \log^{i-1} k \\
&\leq \frac{\alpha k}{2} + \sum_{i=1}^{d} \ell_i \log^{i-1} k,
\end{aligned}$$

where the last step follows since $(2+\alpha)k/\log k \leq \alpha k/2$ for sufficiently large k. This implies the first claim.

For the second claim assume that less than $\alpha d/4$ of the values ℓ_i are greater than 0. Then we obtain

$$\sum_{i=1}^{d} \ell_i \log^{i-1} k < \frac{(1+\alpha)k}{d} \cdot \frac{\alpha d}{4} = \frac{\alpha(1+\alpha)k}{4} \leq \frac{\alpha k}{2},$$

which is a contradiction to the first claim. ∎

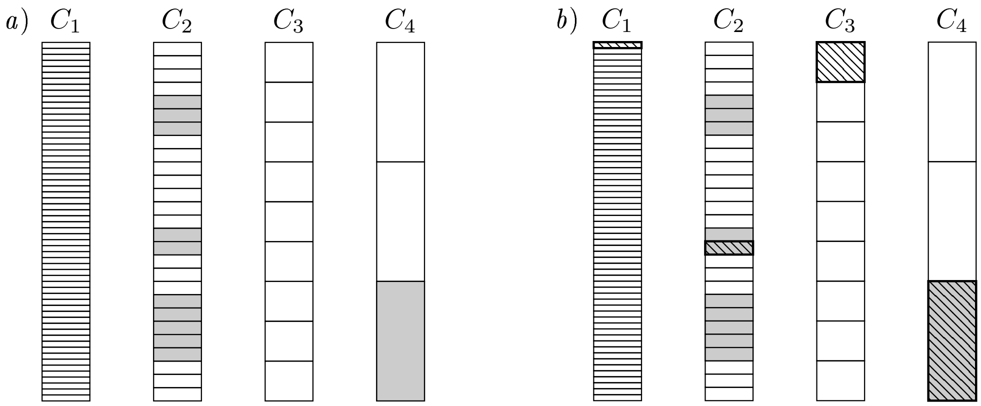

FIG. 3. *Constructing a bad input sequence for a deterministic algorithm. (a) The slots cleared by* ONL *(marked in gray). (b) The slots chosen to be promoted. As there are no cleared slots in classes C_1 and C_3, the first slots of these classes have been chosen.*

In the following sections we describe how to choose the slots to be promoted in a round in such a way that for ONL many cleared slots are promoted while for OPT this happens very rarely. This choice depends on whether we want to derive lower bounds for deterministic or for randomized algorithms.

2.2. Lower bound for deterministic algorithms. In this section we present a lower bound of $\Omega(\sqrt{\log k/\log\log k})$ on the competitive ratio of any deterministic online algorithm for the reordering buffer problem. For this section, we define α to be $\sqrt{\log\log k/\log k}$.

As we consider deterministic algorithms, while constructing the input sequence σ we know what will be the exact contents of the buffer when the algorithm processes the input sequence up to the current round.

For every class C_i, we choose a slot for promotion as follows (see Figure 3).

If in class C_i there exists a slot cleared by ONL, we choose an arbitrary such slot to be promoted. Otherwise, we promote the first slot of class C_i.

We present a randomized algorithm RND that processes σ with a buffer of size k and has small expected cost compared to ONL. As in the general outline for our offline algorithms in the previous section, the algorithm ensures that at the beginning of a round at least $\alpha N_i + 1$ slots from class C_i are cleared. More precisely, for each class C_i, RND chooses $\alpha N_i + 1$ slots uniformly at random from all but the first slot in the class. At the beginning of each round, RND removes all elements belonging to the selected slots from the buffer.

LEMMA 2.5. *The expected cost of* RND *in round r is*

$$O\left(\sqrt{\frac{\log\log k}{\log k}}\right)\cdot cost^r_{\mathsf{ONL}} - 1.$$

Proof. Since RND never chooses to evict the first slot of a class, this slot is never cleared by RND. The probability that a specific other slot of a class is cleared is $(\alpha N_i+1)/(N_i-1) < 2\alpha$. Therefore, the expected cost of RND in round r is at most

$2\alpha \operatorname{cost}^r_{\mathsf{ONL}}$. This is because, due to the way the slots are chosen for promotion, at most $\operatorname{cost}^r_{\mathsf{ONL}}$ slots are promoted that are not the first slots of a class.

From Lemma 2.4(b) we know that $\operatorname{cost}^r_{\mathsf{ONL}} \geq \alpha d/4$. Now since $\alpha = 1/\sqrt{2d} = \sqrt{\log\log k/\log k}$, we get that the expected cost of RND is at most $2\alpha \cdot \operatorname{cost}^r_{\mathsf{ONL}} \leq 10\alpha \cdot \operatorname{cost}^r_{\mathsf{ONL}} - 1$, where the inequality holds since $8\alpha \operatorname{cost}^r_{\mathsf{ONL}} \geq 2\alpha^2 d = 1$. ∎

The lemma implies the following theorem.

THEOREM 2.6. *Any deterministic algorithm for the reordering buffer management problem has a competitive ratio of* $\Omega(\sqrt{\frac{\log k}{\log\log k}})$.

Proof. Clearly, the cost of OPT is at most the expected cost of RND which is, due to Lemma 2.2 and Lemma 2.5, at most $\sum_{i=1}^d N_i + O(\sqrt{\log\log k/\log k})\sum_r \operatorname{cost}^r_{\mathsf{ONL}}$. Due to Lemma 2.1, the cost of ONL is at least $\sum_{i=1}^d N_i + \sum_r \operatorname{cost}^r_{\mathsf{ONL}}$. Therefore, the competitive ratio tends to $\Omega(\sqrt{\log k/\log\log k})$ as the number of rounds tends to infinity. ∎

2.3. Lower bound for randomized algorithms. In this section we provide a lower bound of $\Omega(\log\log k)$ on the competitive ratio of any randomized online algorithm for the reordering buffer management problem. For this section, we define α to be $(\log\log k)^2/\log k$.

For the analysis in this section, for every class C_i, we choose a slot for promotion in the following way.

For class C_i choose a slot z in the class uniformly at random. Promote z.

We start by giving a bound on the expected cost of any online algorithm on the resulting input sequence.

LEMMA 2.7. *For a randomized online algorithm* ONL*, for an input sequence consisting of R rounds, and for sufficiently large k it holds that*

$$\mathbf{E}\left[\textstyle\sum_r \operatorname{cost}^r_{\mathsf{ONL}}\right] \geq R \cdot \frac{\log\log k}{8}.$$

Proof. Let ONL be an arbitrary online algorithm using a buffer of size k. We fix a round r and analyze $\mathbf{E}[\operatorname{cost}^r_{\mathsf{ONL}}]$. For a class C_i, let ℓ_i denote the number of slots from C_i cleared by ONL before round r. Note that the ℓ_i's are (dependent) random variables, but the following holds for any valid fixed choice of values.

According to Lemma 2.4(a) we have $\sum_{i=1}^d \ell_i \cdot \log^{i-1} k \geq \alpha k/2$. If during the round we promote one of the ℓ_i cleared slots in C_i, the value of $\operatorname{cost}^r_{\mathsf{ONL}}$ increases by one. This happens with probability

$$\frac{\ell_i}{N_i} = \ell_i \cdot \frac{\log^{i-1} k \cdot d}{k(1+\alpha)} \geq \ell_i \cdot \log^{i-1} k \cdot \frac{d}{2k}.$$

Summing this over all classes we obtain $\mathbf{E}[\operatorname{cost}^r_{\mathsf{ONL}}] \geq \alpha d/4 = \log\log k/8$. Taking the sum over all R rounds completes the proof. ∎

Next we need to show that the expected optimal cost on the input sequence is significantly smaller.

LEMMA 2.8. *There is an offline algorithm* OFF *using a buffer of size k such that for an input sequence consisting of R rounds and for sufficiently large k,*

$$\mathbf{E}\left[\sum_r \mathrm{cost}^r_{\mathsf{OFF}}\right] \leq 2\alpha \sum_{i=1}^{d} N_i + O(R) \ .$$

Proof. We present an offline algorithm OFF using a buffer of size k that has the desired upper bound on the expected cost. The algorithm ensures that in the beginning of a round at least $\alpha N_i + 1$ slots are cleared from every class C_i. This means that the algorithm has more than $\alpha k + \log^{d-1} k$ elements removed that are held by OPT'.

For a class C_i, one slot is promoted in each round. Consider the sequence of slots from C_i that are promoted over all rounds. We partition this sequence into *phases* such that each phase contains $N_i - \alpha N_i - 1$ pairwise different slots. Then, in the beginning of each phase, i.e., in the beginning of the round in which the first promotion of the new phase takes place, OFF clears all slots that are not contained in the phase. Note that due to the definition of a phase, exactly $\alpha N_i + 1$ slots in class C_i are cleared. Also note that these slots remain cleared during the whole phase, since none of them are promoted.

Let $\mathrm{cost}^r_{\mathsf{OFF}}(i)$ denote the contribution of class C_i to $\mathrm{cost}^r_{\mathsf{OFF}}$; i.e., $\mathrm{cost}^r_{\mathsf{OFF}}(i)$ is 1 if a cleared slot in class C_i is promoted in round r and 0 otherwise. Clearly, $\sum_{i=1}^{d} \sum_r \mathrm{cost}^r_{\mathsf{OFF}}(i) = \sum_r \mathrm{cost}^r_{\mathsf{OFF}}$. In the following we analyze $\mathbf{E}[\sum_r \mathrm{cost}^r_{\mathsf{OFF}}(i)]$ for $i \in \{1, \ldots, d\}$.

Observe that the contribution of one phase to the value of $\sum_r \mathrm{cost}^r_{\mathsf{OFF}}(i)$ is at most $\alpha N_i + 1$. Let p_i be the total number of phases of class C_i; then we get $\mathbf{E}[\sum_r \mathrm{cost}^r_{\mathsf{OFF}}(i)] \leq \mathbf{E}[p_i](\alpha N_i + 1)$. Let X be the length of a single phase (except the last phase which may be incomplete and therefore shorter). Clearly,

$$\mathbf{E}[p_i] \leq 1 + \frac{1}{\mathbf{Pr}[X \geq N_i \ln(1/\alpha)/4]} \cdot \frac{4R}{N_i \ln(1/\alpha)} \ .$$

If we can show that $\mathbf{Pr}[X \geq N_i \ln(1/\alpha)/4] \geq 1/2$, the lemma follows since

$$\begin{aligned}
\mathbf{E}[\textstyle\sum_r \mathrm{cost}^r_{\mathsf{OFF}}] &= \sum_{i=1}^{d} \mathbf{E}[\textstyle\sum_r \mathrm{cost}^r_{\mathsf{OFF}}(i)] \leq \sum_{i=1}^{d} \mathbf{E}[p_i](\alpha N_i + 1) \\
&\leq \sum_{i=1}^{d} 2\,\mathbf{E}[p_i]\alpha N_i \leq \sum_{i=1}^{d} \left(2\alpha N_i + \frac{16R\alpha}{\ln(1/\alpha)}\right) \\
&\leq 2\alpha \sum_{i=1}^{d} N_i + \frac{16dR\alpha}{\ln(1/\alpha)} = 2\alpha \sum_{i=1}^{d} N_i + O(R).
\end{aligned}$$

Here, the second inequality uses the fact that $\alpha N_i \geq 1$, which follows from the integrality of αN_i. The remaining fact required for the proof is encapsulated in the following claim, which is proven in the appendix.

CLAIM 2.9. $\mathbf{Pr}[X \geq N_i \ln(1/\alpha)/4] \geq 1/2$.

THEOREM 2.10. *Any online algorithm for the reordering buffer management problem has competitive ratio at least $\Omega(\log\log k)$.*

Proof. Combining Lemma 2.7 with Lemma 2.1 shows that the expected cost of an online algorithm ONL on the input sequence consisting of R rounds is at least

$\sum_{i=1}^{d} N_i + R \cdot \log\log k/8$. Combining Lemma 2.8 with Lemma 2.2 shows that the expected cost of OPT on the input sequence consisting of R rounds is at most $(2\alpha + 1)\sum_{i=1}^{d} N_i + O(R)$. Therefore, the competitive ratio is at least

$$\frac{\sum_{i=1}^{d} N_i + R \cdot \log\log k/8}{(2\alpha+1)\sum_{i=1}^{d} N_i + O(R)} \ .$$

Letting R tend to infinity gives the theorem. ∎

3. The deterministic upper bound.

In this section we present a deterministic, $O(\sqrt{\log k\gamma})$-competitive online algorithm for the reordering buffer management problem. The cost for switching to a color c can be described by a weight w_c for this color. We assume without loss of generality that $w_c \geq 1$ $\forall$ colors c.

3.1. The algorithm.

Without loss of generality we can assume that an algorithm for the reordering buffer management problem works according to the following general scheme. In each step the algorithm has an *active output color*, which is equal to the color of the last element that was appended to the output sequence. If there is at least one element with this active color in the buffer, the earliest among these elements is removed and appended to the output sequence, and the next element from the input sequence takes its place in the buffer. Otherwise, if there are no more elements of the active output color in the buffer, the algorithm performs a color change and chooses a new color (among the colors present in the buffer) to output next.

Note that the algorithm only has to make a decision if a color change is performed. Therefore, we describe our algorithm LCC (largest color class) by specifying how the new output color is chosen when a color change is required. But first, we introduce some further notation. The *ith step* of an algorithm is the step in which the algorithm appends the ith element to the output sequence. The *buffer content at step i* for an algorithm ALG is the set of elements in ALG's buffer right before the ith element is moved to the output. For the analysis we will assume that the buffer always contains k elements. This may not be true at the end of the input sequence as the algorithm runs out of elements to fill the buffer. However, this part of the sequence does not influence our asymptotic bounds.

Let, for a given color c at a given time t, $\phi_c^t = w_c/n_c^t$ denote the *cost-effectiveness* of color c at time t, where n_c^t denotes the number of elements of color c that are in the buffer of LCC at time t. In the following we drop the superscript, as the time step t will be clear from the context.

For each time step, we partition the colors into classes according to their cost-effectiveness. For $i \in \{-\lceil \log k \rceil, \ldots, \lceil \log \gamma \rceil\}$, the class C_i consists of colors with cost-effectiveness between 2^i and 2^{i+1}. Let $d = O(\log k\gamma)$ denote the number of different classes.

The general idea behind the algorithm is that it aims to remove colors from classes that occupy a large fraction of the space in the buffer. To this end the algorithm selects the class that currently occupies the largest space in the buffer (i.e., it contains at least $\frac{k}{d}$ elements) and *marks* all colors in this class for eviction (line 6 in Algorithm 3.1). Whenever a color change is required, one of these marked colors is chosen as the new output color and unmarked. If there are no marked colors left, the new class that occupies the largest space is selected, and the process is repeated.

This algorithmic idea is combined with a mechanism that penalizes colors for using up space in the buffer at the time a color change occurs. This is similar to techniques

Algorithm 3.1 Largest Color Class (LCC)

Output: a new output color

// *let* n_c *denote the number of elements with color* c *in the buffer*
1: $\forall$ colors $c : t_c \leftarrow \frac{w_c - P_c}{n_c/k}$; $\quad t \leftarrow \min(\{t_c \mid \text{color } c\} \cup \{P\})$;
2: $P \leftarrow P - t$; $\quad \forall$ colors $c : P_c \leftarrow P_c + \frac{n_c}{k} \cdot t$
// *the above ensures that* t *is small enough s.t.* $P \geq 0$ *and* $P_c \leq w_c \ \forall \ c$
3: **if** $P = 0$ **then**
4:     **if** no marked color exists **then**
5:         // *let* $C_{\max}$ *be the class that occupies the largest space in the buffer*
6:         mark all colors in $C_{\max}$
7:     **end if**
    // *let* c_m *denote an arbitrary marked color*
8:     $P \leftarrow w_{c_m}$
9:     $P_{c_m} \leftarrow 0$
10:     unmark color c_m
11:     **return** color c_m as the new output color
12: **else**
13:     $c_a \leftarrow \arg\min_c t_c$ // *pick color* c_a *such that* $P_{c_a} = w_{c_a}$
14:     $P_{c_a} \leftarrow 0$
15:     unmark color c_a if it was marked
16:     **return** color c_a as the new output color
17: **end if**

used, e.g., in [27, 16, 8], and ensures that colors (in particular colors with a low weight) do not stay in the buffer for too long, thereby blocking valuable resources.

To realize all this, our algorithm LCC maintains a counter P and additional penalty counters P_c for every color c. LCC also maintains a flag for every color that indicates if the color is marked. Whenever a color is not in the buffer, its penalty counter is zero. In particular, in the beginning of the algorithm all penalty counters, including the counter P, are zero. The formal description of our algorithm for selecting a new output color is given as Algorithm 3.1.

Before the algorithm selects a marked color c_m as the new output color, it assigns a value of w_{c_m} to a penalty counter P (line 8). In a *postprocessing phase* (after outputting all elements of color c_m) this penalty is distributed to penalty counters of individual colors, as follows. The penalty counter P is continuously decreased at rate 1, while the penalty counters of colors in the buffer are increased at rate n_c/k, where n_c denotes the number of elements of color c that are in the buffer (lines 1 and 2). Note that we assume that the buffer is full; hence the rate of decrease of the P-counter equals the total rate of increase of P_c-counters.

When a counter P_c reaches w_c the penalty distribution is interrupted; the P_c-counter is reset to 0; and the corresponding color c returned as the new output color (lines 13–16). The penalty distribution resumes when all elements with color c have been removed and the next color change takes place. The penalty distribution and the postprocessing phase ends once the P-counter reaches 0.

We note that the algorithm can be significantly simplified for the uniform cost model, i.e., when all colors c have weight $w_c = 1$. In particular, in this case there are just classes C_i, $i \in \{-\lceil \log k \rceil, \ldots, 0\}$, where the class $-i$ contains colors that have roughly 2^i elements in the buffer. Further, instead of the more complicated color distribution process described above one can simply increase the penalty of a counter P_c by n_c/k. After this, the postprocessing phase removes all colors c with $P_c > 1$ via a forced colorchange (instead of interrupting the penalty distribution as described in the algorithm). However, the analysis for this simplified algorithm would have to be slightly adapted as now it just fulfills $P_C \leq 2w_c$ for every color, whereas the above algorithm gurantees that $P_c \leq w_c$ always holds (see Proof of Lemma 3.3).

3.2. The analysis. Let, for a reordering algorithm ALG and an input sequence σ, $\mathsf{ALG}(\sigma)$ denote the output sequence generated by ALG on input σ. A *color-block* of an output sequence is a maximal subsequence of consecutive elements with the same color. The *cost* of a color-block of color c is equal to the weight w_c of c. The cost $|\mathsf{ALG}(\sigma)|$ of algorithm ALG on input σ is defined as the sum of the costs taken over all color-blocks in the output sequence $\mathsf{ALG}(\sigma)$.

For a color-block b we use $s_{\text{start}}(b)$ and $s_{\text{end}}(b)$ to denote the start index of b and end index of b, respectively, in the output sequence. This is the same as the time step when the first and the last element of b is appended to the output sequence.

3.2.1. A few simple cases. In this section we first identify different types of color-blocks for which we can fairly easily derive a bound on their respective contribution to the cost $|\mathsf{LCC}(\sigma)|$ of our online algorithm. In section 3.2.2 we will then introduce a technique that enables us to handle the remaining color-blocks, as well.

We call a color-block of LCC that is not generated in a postprocessing phase a *normal color-block* (these are the color-blocks produced when the algorithm switches to the respective color in line 11). Other color-blocks are called *forced color-blocks* (the ones caused by line 16). The following lemma shows that we can focus our analysis on normal color-blocks.

LEMMA 3.1. *The sum of the costs of forced color-blocks is at most the sum of the costs of normal color-blocks.*

Proof. The total cost for forced color-blocks does not exceed the total penalty that is distributed to colors during the postprocessing phase of normal color-blocks. The penalty that is distributed during the postprocessing phase of a (normal) color-block b with color c is equal to w_c, i.e., the cost for b. Summing over all normal color-blocks gives the lemma. ∎

In the following we use OPT to denote an optimal offline algorithm. We say that an element is *online-exclusive in step i* if in this step the element is in LCC's buffer but has already been removed from OPT's buffer. Similarly we call an element *opt-exclusive in step i* if it is in OPT's buffer but not in LCC's buffer at this time. Note that by this definition in every step the number of online-exclusive elements equals the number of opt-exclusive elements, since the size of LCC's and OPT's buffer is the same.

We extend the above definition to colors. We say that a *color c is online-exclusive* in step i if there exists an element of color c that is online-exclusive. Finally, we say that a *color-block b is online-exclusive* if its color is online-exclusive in step $s_{\text{start}}(b)$. The following lemma derives a bound on the cost of online-exclusive color-blocks.

LEMMA 3.2. *The cost of* LCC *for online-exclusive color-blocks is not larger than* $|\mathsf{OPT}(\sigma)|$.

Proof. Let b denote an online-exclusive color-block in $\mathsf{LCC}(\sigma)$, let e denote its first element, and let c be the color of b. Let b' denote the color-block of color c that precedes b. In case b is the first color-block of color c we define $s_{\text{end}}(b') = -1$ for the following argument. Note that element e is not yet in the buffer at step $s_{\text{end}}(b')+1$ as in this case it would be appended to the output sequence in step $s_{\text{end}}(b')+1$.

Let b_{opt} denote the color-block of OPT that contains the element e. Clearly, this block ends after step $s_{\text{end}}(b')+1$ as e only arrives after this step. Since b is online-exclusive, its first element (i.e., e) is removed from the buffer of OPT before step $s_{\text{start}}(b)$.

Altogether, we have shown that there exists a color-block b_{opt} in $\mathsf{OPT}(\sigma)$ which has color c and ends in the interval $\big(s_{\text{end}}(b')+1, s_{\text{end}}(b)\big)$. We match the online-exclusive block b to b_{opt}. In this way we can match every online-exclusive block to a unique block in $\mathsf{OPT}(\sigma)$ with the same color. This gives the lemma. ∎

Another class of color-blocks for which we can easily derive a bound on the contribution to the cost $|\mathsf{LCC}(\sigma)|$ is given by so-called *opt-far color-blocks* defined as follows. A normal color-block b from the sequence $\mathsf{LCC}(\sigma)$ is called *opt-far* if during its post-processing phase the number of online-exclusive elements never drops below $k/\sqrt{\log k\gamma}$. This means that throughout the whole postprocessing phase for b the buffers of LCC and OPT are fairly different. The following lemma derives an upper bound on the cost of opt-far blocks in an output sequence generated by LCC.

LEMMA 3.3. *The cost of* LCC *for opt-far color-blocks is* $O(\sqrt{\log k\gamma}\cdot|\mathsf{OPT}(\sigma)|)$.

Proof. Fix an opt-far color-block b, and let c denote the color of b. During the postprocessing phase for b the number of online-exclusive elements is always at least $k/\sqrt{\log k\gamma}$. Therefore, at least a $1/\sqrt{\log k\gamma}$ fraction of the penalty distributed during the postprocessing phase goes to online-exclusive colors. The total cost for online-exclusive color-blocks is at least as large as the penalty that these colors receive, since the penalty of a color c cannot increase beyond its cost w_c.

Hence, the total penalty distributed during the postprocessing phases of opt-far color-blocks is at most $\sqrt{\log k\gamma}$ times the cost for online-exclusive color-blocks. This in turn is at most as large as $|\mathsf{OPT}(\sigma)|$ due to Lemma 3.2. The lemma follows by observing that the total penalty distributed during post-processing phases of opt-far color-blocks is equal to the cost of these blocks. ∎

3.2.2. The potential. A crucial ingredient for the proof of the upper bound in section 3.2.3 is the way we how handle *normal* color-blocks that are neither *online-exclusive* nor *opt-far*. For this we introduce the notion of *potential*. The idea is that, on the one hand, the total potential is bounded by some function in terms of the optimal cost $|\mathsf{OPT}(\sigma)|$ (see Claim 3.4(a)). On the other hand, we will show that normal color-blocks that are neither opt-exclusive nor opt-far generate a large potential. This allows us to derive a bound on the contribution of these color-blocks to the cost $|\mathsf{LCC}(\sigma)|$.

The definition of potential is based on the differences in the buffer between LCC and OPT. In the following we use τ_j to denote the start index of the jth color-block of OPT.

For an element e_τ that is appended to the output sequence $\mathsf{LCC}(\sigma)$ at time τ we define for $\tau_j > \tau$

$$\varphi(\tau,\tau_j) = \begin{cases} 0 & \text{if } \mathsf{OPT} \text{ processed } e_\tau \text{ before step } \tau_j,\\ 1, & \text{otherwise.}\end{cases}$$

$\varphi(\tau,\tau_j)$ simply measures whether the element e_τ occupies a slot in OPT's buffer at time τ_j. We say that element e_τ generates *potential* $w_{c_j}\cdot\varphi(\tau,\tau_j)$ for time step τ_j, where c_j denotes the color of the jth color-block in $\mathsf{OPT}(\sigma)$.

For technical reasons we also introduce a *capped potential* as follows. We define

$$\hat{\varphi}(\tau,\tau_j) = \begin{cases} 0 & \text{if } \mathsf{OPT} \text{ processed } e_\tau \text{ before step } \tau_j \textit{ or } \text{at least } {}^{k}/\!\sqrt{\log k\gamma} \\ & \text{elements } e_{\tau'} \text{ with } \tau' < \tau \text{ have } \varphi(\tau',\tau_j) = 1, \\ 1, & \text{otherwise.} \end{cases}$$

$\hat{\varphi}(\tau,\tau_j)$ measures whether the element e_τ is one of the first ${}^{k}/\!\sqrt{\log k\gamma}$ elements to occupy a slot in OPT's buffer at time τ_j, where elements are ordered according to their appearance in $\mathsf{LCC}(\sigma)$. We say that element e_τ generates *capped potential* $w_{c_j} \cdot \hat{\varphi}(\tau,\tau_j)$ for time step τ_j, where c_j denotes the color that is processed by OPT at time τ_j.

We use $\hat{\varphi}(\tau) :- \sum_{j:\tau_j>\tau} w_{c_j}\hat{\varphi}(\tau,\tau_j)$ to denote the total capped potential generated by the element at position τ in $\mathsf{LCC}(\sigma)$. We define the *total capped potential* $\hat{\varphi}$ by $\hat{\varphi} :- \sum_\tau \hat{\varphi}(\tau)$.

CLAIM 3.4. *The capped potential fulfills the following properties:*
(a) $\hat{\varphi} \le k/\sqrt{\log k\gamma} \cdot |\mathsf{OPT}(\sigma)|$.
(b) *Let* $\tau < t < \tau_j$, *and assume that the number of online-exclusive elements in step* t *is at most* $k/\sqrt{\log k\gamma}$. *Then* $\hat{\varphi}(\tau,\tau_j) = \varphi(\tau,\tau_j)$, *and hence the* capped potential $w_{c_j} \cdot \hat{\varphi}(\tau,\tau_j)$ *generated by* e_τ *for* τ_j *is equal to the potential. In other words the contribution of* e_τ *is not capped.*

Proof. The first statement follows from the fact that the capped potential generated for a time step τ_j cannot exceed $k/\sqrt{\log k\gamma} \cdot w_{c_j}$, where w_{c_j} is the cost of OPT in the step. This holds because of the cap. Since the potential is generated for time steps τ_j that correspond to color changes by OPT, the statement follows.

Now, assume for contradiction that the second statement does not hold. Since obviously $\hat{\varphi}(\tau,\tau_j) \le \varphi(\tau,\tau_j)$, it must hold that $\hat{\varphi}(\tau,\tau_j) = 0$ and $\varphi(\tau,\tau_j) = 1$. This means that element e_τ occupies a place in OPT's buffer at time τ_j, but there are at least $k/\sqrt{\log k\gamma}$ elements $e_{\tau'}$, $\tau' < \tau < t$, that also occupy a place in OPT's buffer at time τ_j, and therefore e_τ's contribution is *capped*. But all these elements are opt-exclusive at time t. Since at any time step the number of opt-exclusive elements must be equal to the number of online-exclusive elements, we can conclude that in step t there are more than $k/\sqrt{\log k\gamma}$ online-exclusive elements. This is a contradiction. ◻

3.2.3. The main theorem.

THEOREM 3.5. LCC *is a deterministic online algorithm with competitive ratio* $O(\sqrt{\log k\gamma})$.

Proof. The algorithm LCC marks all colors in a class and then selects an arbitrary marked color whenever it has to do a normal color change. When no marked colors are left, it again marks all colors in some class and continues.

We call the time between two marking operations, or after the last marking operation, a *phase*. Fix some phase P, and let C denote the set of colors that get marked in the beginning of the phase. Let, for $c \in C$, s_c denote the number of elements of color c in LCC's buffer at the time of the marking operation that starts P. Further, let ϕ denote the lower bound on the cost-effectiveness of colors in C, i.e., $\phi \le w_c/s_c \le 2\phi$ holds $\forall$ colors $c \in C$. We call a color change *normal* (*forced*) if it starts a normal (forced) color-block in the output sequence $\mathsf{LCC}(\sigma)$. In $\mathsf{LCC}(\sigma)$ the phase consists of a consecutive subsequence of elements, starting with an element of a color in C and ending with the last element of a color-block from the postprocessing phase of the last normal color change of the phase.

Let $\operatorname{cost}(P)$ denote the cost incurred by LCC during the phase. This cost consists of color changes to colors in C (either normal or forced) and of color changes to colors not in C (these are forced). Let $\operatorname{ncost}(P)$ and $\operatorname{fcost}(P)$ denote the cost incurred by LCC during the phase for normal and forced color changes, respectively. Further, let $\operatorname{ncost} := \sum_{\text{phase } P} \operatorname{ncost}(P)$ denote the total *normal cost* summed over all phases. In the light of Lemma 3.1 it is sufficient to relate ncost to the optimal cost $|\mathsf{OPT}(\sigma)|$. In order to do this we distinguish the following cases:

Case 1. The normal cost $\operatorname{ncost}(P)$ is at most $9/10 \cdot \operatorname{fcost}(P)$. Let $\operatorname{ncost}_{\text{small}}$ denote the normal cost summed over all phases P that fulfill this condition, and let $\operatorname{ncost}_{\text{large}}$ denote the normal cost summed over other phases (i.e., $\operatorname{ncost}_{\text{large}} = \operatorname{ncost} - \operatorname{ncost}_{\text{small}}$). Then

$$\operatorname{ncost}_{\text{small}} \le \frac{9}{10} \sum_P \operatorname{fcost}(P) \le \frac{9}{10} \operatorname{ncost} = \frac{9}{10} (\operatorname{ncost}_{\text{small}} + \operatorname{ncost}_{\text{large}}),$$

where the second inequality follows from Lemma 3.1. This gives $\operatorname{ncost}_{\text{small}} \le 10 \operatorname{ncost}_{\text{large}}$. In the following cases we show that $\operatorname{ncost}_{\text{large}} = O(\sqrt{\log k\gamma} \cdot |\mathsf{OPT}(\sigma)|)$. Then we have that the normal cost $\operatorname{ncost}_{\text{small}}$ generated by phases that fulfill the condition for Case 1 is $O(\sqrt{\log k\gamma} \cdot |\mathsf{OPT}(\sigma)|)$.

Case 2. The cost of OPT during the phase is at least $\operatorname{ncost}(P)/4$. The total normal cost generated by phases that fulfill this condition is $O(|\mathsf{OPT}(\sigma)|)$.

Case 3. The cost of online-exclusive color-blocks generated during the phase is at least $\operatorname{ncost}(P)/4$. Then we can amortize the normal cost of the phase against the cost of online-exclusive color-blocks, which in turn can be amortized against the cost of OPT by Lemma 3.2. This gives that the total normal cost generated by phases that fulfill the conditions for this case is $O(|\mathsf{OPT}(\sigma)|)$.

Case 4. The cost of opt-far color-blocks generated during the phase is at least $\operatorname{ncost}(P)/4$. Then we can amortize the cost of the phase against the cost of opt-far color-blocks, which in turn can be amortized against the cost of OPT by Lemma 3.3.

Hence, the total cost for phases that fulfill the conditions for this case is $O(\sqrt{\log k\gamma} \cdot |\mathsf{OPT}(\sigma)|)$.

Case 5. In the following we assume that none of the above cases occurs. This means there must exist a subset $C' \subset C$ of colors marked in the phase P such that for each color $c \in C'$ its first color-block in the phase is

(a) not online-exclusive,
(b) not opt-far, and
(c) not forced.

Further, we have that

(d) elements of colors in C' are not appended to the output sequence by OPT during the phase
(e) and $\operatorname{cost}(C') \ge \frac{1}{10} \operatorname{cost}(C)$,

where we use $\operatorname{cost}(S) := \sum_{c \in S} w_c$ for a set S of colors.

To see this we generate C' as follows. First take all colors from C (colors initially marked in the phase), and remove colors among them for which the first color change is forced (this ensures property (c)). The cost of the remaining set of colors is exactly $\operatorname{ncost}(P)$. Then remove colors for which the first block of the phase is online-exclusive or opt-far and colors that are requested by OPT during the phase. Since we are not in Case 2, Case 3, or Case 4, this step can only remove colors with a total cost of $3/4 \cdot \operatorname{ncost}(P)$. After this properties (a), (b), and (d) hold. This gives the set C'.

Property (e) can be seen as follows. By the construction $\mathrm{cost}(C') \geq \mathrm{ncost}(P)/4$. From the fact that Case 1 does not hold we get that $\mathrm{ncost}(P) \geq 9/10 \cdot \mathrm{fcost}(P)$, and hence, $2 \cdot \mathrm{ncost}(P) \geq 9/10 \cdot \mathrm{cost}(P) \geq 9/10 \cdot \mathrm{cost}(C)$. This gives $\mathrm{cost}(C') \geq 1/10 \cdot \mathrm{cost}(C)$.

Let S denote the set of elements with colors in C' that are in LCC's buffer in the beginning of the phase. We will show that these elements generate a large potential after the end of the phase. From this it follows that we can amortize the cost of the phase against $|\mathsf{OPT}(\sigma)|$ because of the following argument.

Assume that for some value Z the elements in S generate a potential of at least $Z \cdot \mathrm{cost}(C')$ after time t, where t is the index of the last time step of the phase. Observe that, according to property (b) above, the (first) color-blocks of colors in C' that are generated during the phase are not opt-far. This means that during the postprocessing phase of each of these blocks, the number of online-exclusive items falls below $k/\sqrt{\log k\gamma}$ at some point. This means that we can apply Claim 3.4(b) to all elements in S, which gives that elements in S also generate at least $Z \cdot \mathrm{cost}(C') \geq \frac{Z}{10} \cdot \mathrm{cost}(C)$ *capped potential* after time t, as their contribution to the potential is not capped.

Claim 3.4(a) tells us that the total capped potential is at most $k/\sqrt{\log k\gamma} \cdot |\mathsf{OPT}(\sigma)|$. Therefore, the total normal cost generated by phases that fulfill the conditions for Case 5 is $O(\frac{k}{Z\sqrt{\log k\gamma}}) \cdot |\mathsf{OPT}(\sigma)|$. By showing that elements in S generate at least $Z \cdot \mathrm{cost}(C') = \Omega(\frac{k}{\log k\gamma}) \cdot \mathrm{cost}(C')$ potential we get that the cost of the phases satisfying Case 5 is $O(\sqrt{\log k\gamma}) \cdot |\mathsf{OPT}(\sigma)|$.

For completing the analysis of Case 5 it remains to show the above bound on the potential generated by elements of S. For this, we first show that the cardinality of the set S is large. We have

$$|S| = \sum_{c \in C'} s_c \geq \sum_{c \in C'} w_c/2\phi \geq \frac{1}{20\phi} \sum_{c \in C} w_c \geq \frac{1}{20} \sum_{c \in C} s_c \geq \frac{k}{20d} = \Omega(k/\log k\gamma),$$

where the first and third inequality follows since $\phi \leq w_c/s_c \leq 2\phi$ and the second inequality holds since $\sum_{c \in C'} w_c = \mathrm{cost}(C') \geq \mathrm{cost}(C)/10 \geq \sum_{c \in C} w_c/10$. The last inequality follows since the algorithm LCC selects a class that occupies the largest space in the buffer and hence occupies at least space k/d, where d denotes the number of classes.

CLAIM 3.6. *Let S denote a set of elements that are opt-exclusive at time t, and let s_c denote the number of elements of color c in S. Assume that there is a value ϕ such that $\phi \leq \frac{w_c}{s_c} \leq 2\phi$ holds $\forall$ colors with elements in S. Then the contribution to the potential by elements from S generated after time t is at least $\frac{1}{8}|S| \cdot \mathrm{cost}(S)$.*

Proof. Let $c_1, \ldots, c_\ell$ denote the colors of the elements in S, ordered according to the times $\tau_1 < \cdots < \tau_\ell$ at which the first element of a color is evicted by OPT. Let i denote the smallest number such that $\sum_{j=1}^{i} w_{c_j} \geq \frac{1}{2}\mathrm{cost}(S)$.

We show that the number of elements with colors $c_i, \ldots, c_\ell$ is large. For any j we have $\phi \leq \frac{w_{c_j}}{s_{c_j}} \leq 2\phi$, and hence $\mathrm{cost}(S) \geq \phi|S|$. Therefore

$$\sum\nolimits_{j=i}^{\ell} s_{c_j} \geq \frac{1}{2\phi} \sum\nolimits_{j=i}^{\ell} w_{c_j} \geq \frac{1}{4\phi}\mathrm{cost}(S) \geq \frac{1}{4}|S|.$$

Each element e of a color in $\{c_i, \ldots, c_\ell\}$ generates potential w_{c_j} at time τ_j for $1 \leq j \leq i$. The contribution of e to the potential generated after time t is therefore at least $\sum_{j=1}^{i} w_{c_j} \geq \frac{1}{2}\mathrm{cost}(S)$. As the number of elements with color in $\{c_i, \ldots, c_\ell\}$ is at least $\frac{1}{4}|S|$, the potential generated by them after time t is at least $\frac{1}{8}|S| \cdot \mathrm{cost}(S)$. ∎

Applying the claim with t being the last step of the phase gives that the elements from S generate potential $\Omega(\frac{k}{\log k\gamma}) \cdot \text{cost}(C')$ after the end of the phase. This finishes the analysis of Case 5.

The above cases show that the contribution of all phases to the cost of LCC is $O(\sqrt{\log k\gamma}) \cdot |\mathsf{OPT}(\sigma)|$. This gives the theorem. ∎

Appendix A. Proof of Claim 2.9. The analysis of the random variable X is based on a straightforward coupon collector–type argument and is included for completeness.

CLAIM 2.9. $\mathbf{Pr}[X \geq N_i \ln(1/\alpha)/4] \geq 1/2$.

Proof. Consider a phase, and let X_j be the number of rounds between the promotion of the $(j-1)$th distinct slot and the jth distinct slot. Then $X = X_1 + X_2 + \cdots + X_{N_i - \alpha N_i} - 1$. The variable X_j has value ℓ with probability $(\frac{j-1}{N_i})^{\ell-1} \cdot \frac{N_i - j + 1}{N_i}$ for any integer $\ell \geq 1$. We have

$$\mathbf{E}[X_j] = \frac{N_i}{N_i - j + 1}, \quad \mathbf{E}[X_j^2] = \frac{(N_i + j - 1)N_i}{(N_i - j + 1)^2}, \quad \text{and}$$
$$\mathbf{Var}[X_j] = \mathbf{E}[X_j^2] - \mathbf{E}[X_j]^2 = \frac{(j-1)N_i}{(N_i - j + 1)^2}.$$

We get

$$\begin{aligned}
\mathbf{E}[X] &= \frac{N_i}{N_i} + \frac{N_i}{N_i - 1} + \cdots + \frac{N_i}{\alpha N_i + 1} - 1 \\
&= N_i(H_{N_i} - H_{\alpha N_i}) - 1 \\
&\geq N_i \cdot \ln N_i - N_i \cdot (\ln(\alpha N_i) + 1) - 1 \\
&= N_i \ln(1/\alpha) - (N_i + 1) \geq N_i \ln(1/\alpha)/2,
\end{aligned}$$

where the first inequality uses the fact that $\ln a < H_a \leq \ln a + 1$ for $a \geq 1$ and the second inequality holds for sufficiently small α (i.e., for sufficiently large k).

From Chebyshev's inequality we get

$$\begin{aligned}
&\mathbf{Pr}\left[X \leq N_i \ln(1/\alpha)/4\right] \\
&\qquad \leq \mathbf{Pr}\left[|X - \mathbf{E}[X]| \geq N_i \ln(1/\alpha)/4\right] \\
&\qquad \leq \frac{16}{N_i^2 \ln^2(1/\alpha)} \cdot \mathbf{Var}[X] \\
&\qquad = \frac{16}{N_i^2 \ln^2(1/\alpha)} \cdot \sum_{j=1}^{(1-\alpha)N_i} \mathbf{Var}[X_j] \\
&\qquad \leq \frac{16}{\ln^2(1/\alpha)} \cdot \sum_{j=1}^{(1-\alpha)N_i} \frac{1}{(N_i - j + 1)^2} \\
&\qquad \leq \frac{16}{\ln^2(1/\alpha)} \cdot \sum_{j=1}^{\infty} \frac{1}{j^2} \leq \frac{32}{\ln^2(1/\alpha)} \leq \frac{1}{2},
\end{aligned}$$

where the third step follows since the X_j's are independent and the last step follows for sufficiently small α. ∎

REFERENCES

[1] A. Aboud, *Correlation Clustering with Penalties and Approximating the Reordering Buffer Management Problem*, Master's thesis, The Technion — Israel Institute of Technology, 2008.

[2] A. Adamaszek, A. Czumaj, M. Englert, and H. Räcke, *Optimal online buffer scheduling for block devices*, in Proceedings of the 44th ACM Symposium on Theory of Computing (STOC), 2012, pp. 589–598.

[3] A. Adamaszek, M. P. Renault, A. Rosén, and R. van Stee, *Reordering buffer management with advice*, J. Sched., 20 (2016), pp. 423–442.

[4] Y. Asahiro, K. Kawahara, and E. Miyano, *NP-hardness of the sorting buffer problem on the uniform metric*, Discrete Appl. Math., 160 (2012), pp. 1453–1464.

[5] N. Avigdor-Elgrabli, S. Im, B. Moseley, and Y. Rabani, *On the randomized competitive ratio of reordering buffer management with non-uniform costs*, in Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP), Part I, 2015, pp. 78–90.

[6] N. Avigdor-Elgrabli and Y. Rabani, *A constant factor approximation algorithm for reordering buffer management*, in Proceedings of the 24th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2013, pp. 973–984.

[7] N. Avigdor-Elgrabli and Y. Rabani, *An optimal randomized online algorithm for reordering buffer management*, in Proceedings of the 54th IEEE Symposium on Foundations of Computer Science (FOCS), 2013, pp. 1–10.

[8] N. Avigdor-Elgrabli and Y. Rabani, *An improved competitive algorithm for reordering buffer management*, ACM Trans. Algorithms, 11 (2015), 35.

[9] R. Bar-Yehuda and J. Laserson, *Exploiting locality: Approximating sorting buffers*, J. Discrete Algorithms, 5 (2007), pp. 729–738.

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

[11] D. Blandford and G. Blelloch, *Index compression through document reordering*, in Proceedings of the Data Compression Conference (DCC), 2002, pp. 342–351.

[12] N. Buchbinder and J. Naor, *The design of competitive online algorithms via a primal-dual approach*, Found. Trends Theor. Comput. Sci., 3 (2009), pp. 93–263.

[13] H. Chan, N. Megow, R. Sitters, and R. van Stee, *A note on sorting buffers offline*, Theor. Comput. Sci., 423 (2012), pp. 11–18.

[14] M. Englert and H. Räcke, *Reordering buffers with logarithmic diameter dependency for trees*, in Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2017, pp. 1224–1234.

[15] M. Englert, H. Räcke, and M. Westermann, *Reordering buffers for general metric spaces*, Theory Comput., 6 (2010), pp. 27–46.

[16] M. Englert and M. Westermann, *Reordering buffer management for non-uniform cost models*, in Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP), 2005, pp. 627–638.

[17] M. Englert and M. Westermann, *Scheduling with a reordering buffer*, in Encyclopedia of Algorithms, Springer, New York, 2016, pp. 1905–1910.

[18] H. Esfandiari, M. Hajiaghayi, M. R. Khani, V. Liaghat, H. Mahini, and H. Räcke, *Online stochastic reordering buffer scheduling*, in Proceedings of the 41st International Colloquium on Automata, Languages and Programming (ICALP), Part I, 2014, pp. 465–476.

[19] I. Gamzu and D. Segev, *Improved online algorithms for the sorting buffer problem on line metrics*, ACM Trans. Algorithms, 6 (2009), 15.

[20] K. Gutenschwager, S. Spiekermann, and S. Voss, *A sequential ordering problem in automotive paint shops*, Int. J. Prod. Res., 42 (2004), pp. 1865–1878.

[21] S. Im and B. Moseley, *New approximations for reordering buffer management*, in Proceedings of the 25th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2014, pp. 1093–1111.

[22] S. Im and B. Moseley, *Weighted reordering buffer improved via variants of knapsack covering inequalities*, in Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP), 2015, pp. 737–748.

[23] R. Khandekar and V. Pandit, *Online and offline algorithms for the sorting buffers problem on the line metric*, J. Discrete Algorithms, 8 (2010), pp. 24–35.

[24] M. Kohler and H. Räcke, *Reordering buffer management with a logarithmic guarantee in general metric spaces*, in Proceedings of the 44th International Colloquium on Automata, Languages and Programming (ICALP), Leibniz International Proceedings in Informatics 80, 2017, 33.

[25] J. S. Kohrt and K. Pruhs, *A constant factor approximation algorithm for sorting buffers*, in Proceedings of the 6th Latin American Symposium on Theoretical Informatics (LATIN), 2004, pp. 193–202.

[26] J. Krokowski, H. Räcke, C. Sohler, and M. Westermann, *Reducing state changes with a pipeline buffer*, in Proceedings of the 9th International Fall Workshop Vision, Modeling, and Visualization (VMV), 2004, pp. 217–224.

[27] H. Räcke, C. Sohler, and M. Westermann, *Online scheduling for sorting buffers*, in Proceedings of the 10th European Symposium on Algorithms (ESA), 2002, pp. 820–832.