Online Scheduling for Sorting Buffers*

Harald Räcke[1], Christian Sohler[1], and Matthias Westermann[2]

[1] Heinz Nixdorf Institute and
Department of Mathematics and Computer Science
Paderborn University, D-33102 Paderborn, Germany
{harry,csohler}@uni-paderborn.de
[2] International Computer Science Institute
Berkeley, CA 94704, USA
marsu@icsi.berkeley.edu

Abstract. We introduce the online scheduling problem for sorting buffers. A service station and a sorting buffer are given. An input sequence of items which are only characterized by a specific attribute has to be processed by the service station which benefits from consecutive items with the same attribute value. The sorting buffer which is a random access buffer with storage capacity for k items can be used to rearrange the input sequence. The goal is to minimize the cost of the service station, i.e., the number of maximal subsequences in its sequence of items containing only items with the same attribute value. This problem is motivated by many applications in computer science and economics.
The strategies are evaluated in a competitive analysis in which the cost of the online strategy is compared with the cost of an optimal offline strategy. Our main result is a deterministic strategy that achieves a competitive ratio of $O(\log^2 k)$. In addition, we show that several standard strategies are unsuitable for this problem, i.e., we prove a lower bound of $\Omega(\sqrt{k})$ on the competitive ratio of the First In First Out (FIFO) and Least Recently Used (LRU) strategy and of $\Omega(k)$ on the competitive ratio of the Largest Color First (LCF) strategy.

1 Introduction

In the online scheduling problem for sorting buffers, we are given a service station and a sorting buffer. An input sequence of items which are only characterized by a specific attribute has to be processed by the service station which benefits from consecutive items with the same attribute value. The sorting buffer which is a random access buffer with limited storage capacity can be used to rearrange the input sequence. Whenever a new item arrives it has to be stored in the sorting buffer. Accordingly, items can also be removed from the sorting buffer and then assigned to the service station. Hence, the service station has to process a sequence of items that is a partial rearrangement of the input sequence. The goal is to minimize the cost of the service station, i.e., the number of

* The first two authors are partially supported by the DFG-Sonderforschungsbereich 376, DFG grant 872/8-1, and the Future and Emerging Technologies program of the EU under contract number IST-1999-14186 (ALCOM-FT). The third author is supported by a fellowship within the ICSI-Postdoc-Program of the German Academic Exchange Service (DAAD).

maximal subsequences of items with the same attribute value in the rearranged sequence. This problem is motivated by many applications in computer science and economics. In the following we give some examples.

In computer graphics, the process of displaying a representation of 3D objects, i.e., polygons, is denoted as *rendering*. Each polygon is characterized by several attributes, e.g., color and texture. In the rendering process, a sequence of polygons is transmitted to the graphic hardware. A change of attributes for consecutive polygons in this sequence is denoted as *state change*. One of the determining factors for the performance of a rendering system is the number of state changes in the sequence of polygons. Of course, unless you want to model a black ninja during night, there have to be some state changes. However, the number of state changes can be reduced by preceding a rendering system with a sorting buffer.

Communication in computer systems connected by a network is usually realized by the transfer of data streams. The sending of a data stream by a computer is denoted as *startup*. The overhead induced by the startup process (inclusive the overhead of the receiving computer) is denoted as *startup cost*. Sending many small data streams has a negative impact on the performance of the network, since in this case the startup cost dominates the total cost. The startup cost can be reduced by combining several data streams that are directed to the same destination. Hence, with the assistance of sorting buffers at the computers on the network the performance of the network can be improved.

File server are computer and high-capacity storage devices which each computer on a network can access to retrieve files. Hence, a file server receives a sequence of read and write accesses to files from the computers on the network. A file is denoted as *open*, if it is ready to be accessed. Otherwise, it is denoted as *closed*. By technical reasons, the number of open files on a file server is limited. Since the overhead induced by the opening and closing process takes a lot of time for a file server, it is preferable if as many as possible accesses to an open file can be processed before closing it. The number of opening and closing procedures can be minimized by preceding a file server with a multi service sorting buffer which is a generalization of our sorting buffer. A *multi service sorting buffer* has m identical service stations and each item can be processed by any of these stations. In this example, m is the maximum number of open files on a file server. Note that the multi service sorting buffer problem in which the sorting buffer has storage capacity for only one item is equivalent to the classical paging problem (see, e.g., [2,10]).

In the painting center of a car plant, a sequence of cars traverses the final layer painting where each car is painted with its own top coat. If two consecutive cars have to be painted in different colors then a *color change* is required. Such a color change causes cost due to the wastage of paint and the use of cleaning chemicals. These costs can be minimized by rearranging the sequence of cars in such a way that cars of the same color preferably appear in consecutive positions. For this purpose, the final layer painting is preceded by a queue sorting buffer which is a generalization of our sorting buffer. A *queue sorting buffer* consists of several queues each with limited storage capacity. Whenever a new car arrives, it has to be transferred to the tail of any of the queues. Accordingly, cars at the head of any queue can also be removed and then assigned to the final layer painting.

1.1 The model

We are given a *service station* and a *sorting buffer*. An *input sequence* $\sigma = \sigma_1\sigma_2\cdots\sigma_n$ of items which are only characterized by a specific attribute has to be processed by the service station which benefits from consecutive items with the same attribute value. To simplify matters, we suppose that the items are characterized by their color. The sorting buffer which is a random access buffer with storage capacity for k items can be used to rearrange the input sequence in the following way.

The *current input item* σ_i, i.e., the first item of σ that is not handled yet, can be stored in the sorting buffer, or items currently in the sorting buffer can be removed and then assigned to the service station. These removed items result in an *output sequence* $\rho = \rho_1\rho_2\cdots\rho_n$ which is a partial rearrangement of σ. We suppose that the sorting buffer is initially empty and, after processing the whole input sequence, the buffer has to be empty again. In addition, we use the following notations. Let the *current input color* denote the color of the current input item. Further, let the *current output item* denote the item that was last assigned to the service station and let the *current output color* denote the color of this item.

The goal is to rearrange the input sequence in such a way that items with the same color preferably appear in consecutive positions in the output sequence. Let each maximal subsequence of the output sequence containing only items with the same color denote as *color block*. Between two different color blocks there is a *color change* at the service station. Let the *cost* $C(\sigma)$ *of the scheduling strategy* denote the number of color blocks in the output sequence. Then, the goal is to minimize the cost $C(\sigma)$.

The notion of an online strategy is intended to formalize the realistic scenario, where the scheduling strategy does not have knowledge about the whole input sequence in advance. Instead, it learns the input piece by piece, and has to react with only partial knowledge of the input. Online strategies are typically evaluated in a competitive analysis. In this kind of analysis which was introduced by Sleator and Tarjan [10] the cost of the online strategy is compared with the cost of an optimal offline strategy.

In order to obtain a simple, unambiguous model, we assume that an adversary initiates the input sequence $\sigma = \sigma_1\sigma_2\cdots\sigma_n$ of items. The online strategy has to serve these items one after the other, that is, it is assumed that σ_{i+1} is not issued before σ_i is not stored in the sorting buffer. For a given sequence σ, let $C_{\mathrm{op}}(\sigma)$ denote the minimum cost produced by an optimal offline strategy. An online strategy is said to be *c-competitive* if it produces cost at most $c \cdot C_{\mathrm{op}}(\sigma) + a$, for each sequence σ, where a is a term that does not depend on σ. The value c is also called the *competitive ratio* of the online strategy.

W.l.o.g., we only consider *lazy* scheduling strategies, i.e., strategies that fulfill the following two properties:

- If an item whose color is equal to the current output color is stored in the sorting buffer, a lazy strategy does not make a color change.
- If there are items in the input sequence that can be stored in the sorting buffer, a lazy strategy does not remove an item from the sorting buffer.

Note that an optimal offline strategy can be transformed into a lazy strategy without increasing the cost.

1.2 Our contribution

We introduce the online scheduling problem for sorting buffers and present deterministic scheduling strategies for this problem. The strategies aim to rearrange an input sequence in such a way that items of the same color preferably appear in consecutive positions in the output sequence. They are evaluated in a competitive analysis.

At first, we show in Section 2 that several standard strategies are unsuitable for this problem, i.e., we prove a lower bound of $\Omega(\sqrt{k})$ on the competitive ratio of the First In First Out (FIFO) and Least Recently Used (LRU) strategy and of $\Omega(k)$ on the competitive ratio of the Largest Color First (LCF) strategy. Our main result which we present in Section 3 is the deterministic Bounded Waste strategy which achieves a competitive ratio of $O(\log^2 k)$. We believe that the Bounded Waste strategy is well suited for the application in practice because the strategy is simple and has a provably good performance. The main part of the proof concerning the upper bound of the Bounded Waste strategy is given in Section 4.

1.3 Previous work

Scheduling is continuously an active research area, reflecting the changes in theoretical computer science. When the theory of NP-completeness was developed, many scheduling problems have been shown to be NP-complete (see, e.g., [4]). After the NP-completeness results, the focus shifted to the design of approximation algorithms (see, e.g., [6,8]). Many natural heuristics for scheduling are in fact online strategies. Hence, when the study of online strategies using competitive analysis became usual, this approach was naturally and quite successfully applied to scheduling (see, e.g., [9]).

The online bin-coloring problem is related to our scheduling problem. The goal in the bin-coloring problem is to pack unit size colored items into bins, such that the maximum number of different colors per bin is minimized. The packing process is subject to the constraint that at any moment at most k bins are partially filled. Moreover, bins may only be closed if they are filled completely. Krumke et al. [7] present a deterministic strategy that achieves a competitive ratio of $(2k + 1)$. In addition, they give an $\Omega(k)$ lower bound on the competitive ratio of any deterministic strategy.

Feder et al. [1] study an online caching problem that is similar to our scheduling problem. In their r-reordering problem, a caching strategy can reorder the sequence of requests $\sigma = \sigma_1\sigma_2\cdots$ as long as no request is delayed inordinately, i.e., in the new ordering a request σ_i has to be served before a request σ_j, if $i + r \leq j$. For cache size one, they present a deterministic greedy strategy that achieves competitive ratio two and a lower bound of 1.5 on the competitive ratio of any deterministic strategy. For the case that lookahead l is in addition possible, they give a deterministic strategy that achieves competitive ratio $1 + O(r/l)$ and a lower bound of $1 + \Omega(r/l)$ on the competitive ratio of any strategy.

The following variant of an offline problem for queue sorting buffers is well studied. Suppose we are given a sorting buffer with k queues of infinite storage capacity and all items of the input sequence are already stored in the queues. Now, it remains only to determine a sequence of remove operations to empty the queues. This problem is equivalent to the shortest common super-sequence problem. Fraser and Irving [3] present

a polynomial time algorithm for this problem that calculates a $(4k + 12)$-approximation. Further, Jiang and Li [5] show that this problem is not in APX, i.e., for this problem exists no polynomial time algorithm with constant approximation ratio, unless P = NP.

In several practical work, heuristic scheduling strategies are used for the sorting buffer problem (see, e.g., [11]). This work is almost always motivated by the demand for efficient strategies for sorting buffers in manufacturing control. Spieckermann and Voss [11] evaluate some simple heuristic strategies by simulation and come to the conclusion that there is a lack of efficient strategies.

2 Lower bounds

In this section, we give lower bounds on the competitive ratio of several strategies that have previously been applied to other scheduling problems.

First In First Out (FIFO). Each time an item σ_i is stored in the sorting buffer the FIFO strategy checks whether there is another item of the same color stored in the sorting buffer. If there is no such item, the color of σ_i gets a time stamp. If there is no item of the current output color in the sorting buffer, FIFO selects the color with the "oldest" time stamp.

Theorem 1. *The competitive ratio of the FIFO strategy is at least* $\Omega(\sqrt{k})$.

Proof. W.l.o.g., we assume that $\ell = \sqrt{k-1}$ is integral and even. We consider the colors $c_1, \dots c_\ell, x, y$ and the sequence $\sigma = (c_1 \cdots c_\ell x^k c_1 \cdots c_\ell y^k)^{\ell/2}$.

At first, we show that σ can be processed with 2ℓ color changes. Consider the sequence $(xy)^{\ell/2} c_1 \cdots c_\ell$ of 2ℓ color changes. During the first ℓ color changes of this sequence the items of colors $c_1, \dots c_\ell$ are accumulated in the sorting buffer. Note that the total number of these items is $2\ell \cdot \ell/2 = k - 1$.

Now it remains to count the color changes if σ is processed by the FIFO strategy. Whenever a block of x^k or y^k appears FIFO empties the whole sorting buffer, since all other colors stored in it are "older". For each block, this produces ℓ color changes. Hence, if σ is processed by FIFO, at least $2\ell \cdot \ell/2 = k - 1$ color changes occur. □

Least Recently Used (LRU). Similar to FIFO the LRU strategy assigns to each color a time stamp. LRU updates the time stamp of a color each time a new item of that color is stored in the sorting buffer. Thus the time stamp of a color is the time when the most recent item of that color was stored. If there is no item of the current output color in the sorting buffer, LRU selects the color with the "oldest" time stamp.

Theorem 2. *The competitive ratio of the LRU strategy is at least* $\Omega(\sqrt{k})$.

Proof. It is easy to see that for the sequence defined in the proof of Theorem 1 the LRU strategy produces the same output sequence as FIFO. □

Largest Color First (LCF). Another fairly natural strategy for the sorting buffer problem is to free as many locations in the sorting buffer as possible, if the strategy has to make a color change. If there is no item of the current output color in the sorting buffer, the LCF strategy selects a color that has the most items in the sorting buffer.

Theorem 3. *The competitive ratio of the LCF strategy is at least $\Omega(k)$.*

Proof. W.l.o.g., we assume that $k \geq 4$ is even. We consider the colors $c_1, \ldots c_{k-2}, x, y$ and the sequence $\sigma = c_1 \cdots c_{k-2}(xxyy)^{n \cdot k}$, for some integer n.

At first, we show that σ can be processed with $k - 2 + 8n$ color changes. Consider the sequence $c_1 \cdots c_{k-2}(xy)^{4n}$ of $k - 2 + 8n$ color changes. After the items of colors $c_1, \ldots c_{k-2}$ have been removed from the sorting buffer every further color change removes at least $k/2$ items.

Now it remains to count the color changes if σ is processed by the LCF strategy. LCF does not remove the items of colors $c_1, \ldots c_{k-2}$ from the sorting buffer before the items of colors x and y are processed. Hence, if σ is processed by LCF, $2n \cdot k + k - 2$ color changes occur. □

3 Bounded Waste strategy

How should a good scheduling strategy for sorting buffers look like? On the one hand, no item should be kept in the sorting buffer for a too long, possibly infinite, period of time. This would waste valuable storage capacity that could be used for efficient scheduling otherwise. For example the LCF strategy from the previous section fails to achieve a good competitive ratio because some items are kept in the sorting buffer for nearly the whole sequence.

On the other hand, there is a benefit from keeping an item in the sorting buffer if items of the same color arrive in the near future. Thus, a strategy may fail as well if it removes items too early. Good examples for this phenomenon are the LRU and FIFO strategy from the previous section. These strategies tend to remove items too early from the sorting buffer and, hence, cannot build large color blocks if additional items of the same color arrive.

We need a trade-off between the space wasted by items of a color and the chance to benefit from future items of the same color. Such a trade-off is provided by the *Bounded Waste strategy*. This strategy continues to remove items of the current output color from the sorting buffer as long as this is possible, i.e., until all items in the sorting buffer have a color different from the current output color. Then the strategy has to decide which color is removed next from the sorting buffer. For this purpose we introduce, for each color c, the *penalty* P_c. The penalty P_c for color c is a measure of the space that has been wasted by all items of color c that are currently in the sorting buffer. Initially, the penalty for each color is zero. At each color change, the penalty for each color c is increased by the number of items of color c currently stored in the sorting buffer. Then a color c' with maximal penalty $P_{c'}$ is chosen, an item of color c' is removed from the sorting buffer, and $P_{c'}$ is reset to zero.

For the competitive analysis, we have to compare the Bounded Waste strategy with an optimal offline strategy, for each input sequence σ. In the following we assume that

an optimal offline strategy and an arbitrary input sequence σ is fixed. Then the sequence of color changes of the Bounded Waste and optimal offline strategy are fixed as well.

In order to compare both strategies, we introduce the notation of waste. At first, a color change of the Bounded Waste or optimal offline strategy is called *online* or *offline color change*, respectively. We say that an (online or optimal offline) strategy *produces waste* w for color c at an online color change if it has w item of color c in its sorting buffer at this color change. Note that the waste of the optimal offline strategy is also produced at online color changes. Further, we define the (total) *online* and *optimal offline waste* for color c as the total waste produced for color c at all online color changes by the online and optimal offline strategy, respectively. The waste for color c is strongly related to the penalty for c: The penalty for c is equivalent to the waste produced for c by the online strategy since the most recent online color change to c.

In the following we describe how the notion of waste can be used to derive an upper bound on the competitive ratio of the Bounded Waste strategy. Let W^c_{on} and W^c_{op} denote the online and optimal offline waste for color c, respectively, and let C_{on} and C_{op} denote the number of color blocks in the output sequence of the Bounded Waste strategy and the optimal offline strategy, respectively. Then

$$\sum_{\text{color } c} W^c_{\mathrm{op}} \leq k \cdot C_{\mathrm{on}} \qquad \text{and} \qquad \sum_{\text{color } c} W^c_{\mathrm{on}} \geq k \cdot C_{\mathrm{on}} - k^2 \ ,$$

since at all but the last k color changes the online as well as the optimal offline strategy have k items in the sorting buffer, because both strategies are lazy, i.e., they do not make a color change if they have free space in their sorting buffer and remaining items in the input sequence. Let $\Delta^c = W^c_{\mathrm{op}} - W^c_{\mathrm{on}}$ denote the difference between online and optimal offline waste for color c. Then

$$\sum_{\text{color } c} \Delta^c \leq k^2 \ .$$

The main technical contribution of this paper is to show the following lemma. Its proof is postponed to the next section.

Lemma 1 (Main Lemma). *For each color c,*

$$\Delta^c \geq W^c_{\mathrm{on}} - C^c_{\mathrm{op}} \cdot O(k \cdot \log^2 k) \ ,$$

with C^c_{op} denoting the number of blocks with color c in the output sequence of an optimal offline strategy.

The following theorem gives an upper bound of the competitive ratio of the Bounded Waste strategy.

Theorem 4. *The Bounded Waste strategy achieves a competitive ratio of $O(\log^2 k)$.*

Proof. With the Main Lemma, we can conclude

$$\begin{aligned} k^2 &\geq \sum_{\text{color } c} \Delta^c \\ &\geq \sum_{\text{color } c} \left(W^c_{\mathrm{on}} - C^c_{\mathrm{op}} \cdot O(k \cdot \log^2 k) \right) \geq k \cdot C_{\mathrm{on}} - k^2 - C_{\mathrm{op}} \cdot O(k \cdot \log^2 k) \ . \end{aligned}$$

Hence, $C_{\mathrm{op}} \cdot O(\log^2 k) + 2k \geq C_{\mathrm{on}}$, which implies the theorem. □

4 Proof of the Main Lemma

In this section, we present the postponed proof of Lemma 1 (Main Lemma).

Proof (of Lemma 1 (Main Lemma)). At first, we show that, for each color c, the penalty P_c is at most $O(k \cdot \log k)$. By the relationship between waste and penalty, this means that the waste produced by the Bounded Waste strategy for color c between two consecutive online color changes to c is at most $O(k \cdot \log k)$. Let $P_c(i)$ denote the penalty for color c directly after the i-th online color change and define $P_c(0) = 0$.

Lemma 2 (Bounded Waste). *For each color c and each online color change i, $P_c(i) = O(k \cdot \log k)$.*

Proof. For the analysis, we use a potential function $\Phi(i)$ that depends on the penalties currently assigned to all colors. The intuition behind this potential function is as follows. We assume that we have $P_c(i)$ units of waste for color c. These units are put on a stack (one stack for all units of the same color). If a unit is at position j on the stack then it contributes with the value $\phi(j)$ to the potential function. The function ϕ is monotonously increasing. Hence, the higher the stack the more expensive become the units of waste.

We define the potential function as follows:

$$\Phi(i) = \sum_{\text{color } c} \sum_{j=1}^{P_c(i)} \phi(j) \quad , \quad \text{with} \quad \phi(j) = 2^{\lfloor \frac{j}{k} \rfloor} + \frac{j}{k} \quad .$$

Our goal is to show that $\Phi(i) = O(k^2)$. This immediately implies that, for each color c, $P_c(i) = O(k \cdot \log k)$. For this purpose, we need the following two propositions.

Proposition 1. *If, for each colors c, $P_c(i) \leq 5k$, then $\Phi(i) = O(k^2)$.*

Proof. It is easy to verify that $\phi(5k)$ is a constant. Hence, it remains to prove that $\sum_{\text{color } c} P_c(i) = O(k^2)$. This follows from the fact that there are at most k colors c with $P_c(i) > 0$. □

Proposition 2. *If there exists a color c, with $P_c(i) \geq 4k$, then $\Phi(i+1) < \Phi(i)$.*

Proof. Let $\Delta\Phi(i+1) = \Phi(i+1) - \Phi(i)$, i.e., the change in the potential at the $(i+1)$-th online color change. Recall that the Bounded Waste strategy first increases the penalty for each color. Then it chooses a color c' with maximal penalty $P_{c'}$, removes an item of color c' from the sorting buffer, and resets $P_{c'}$ to zero.

Let m be the maximal penalty $P_{c'}$ after it has been increased and before it is reset to zero. Then

$$\begin{aligned}
\Delta\Phi(i+1) &\leq k \cdot \phi(m) - \sum_{j=1}^{m} \phi(j) \\
&= k \cdot 2^{\lfloor \frac{m}{k} \rfloor} + m - \sum_{j=1}^{m} 2^{\lfloor \frac{j}{k} \rfloor} - \sum_{j=1}^{m} \frac{j}{k} \\
&\leq k \cdot 2^{\lfloor \frac{m}{k} \rfloor} - k \cdot \sum_{j=1}^{\lfloor \frac{m}{k} \rfloor - 1} 2^j + m - \frac{m \cdot (m+1)}{2k} \\
&= k - \frac{m \cdot (m - 2k + 1)}{2k} \quad .
\end{aligned}$$

Hence, $\Delta\Phi(i+1) < 0$ for $m \geq 4k$ which implies the proposition. □

We prove by induction on i that $\Phi(i) = O(k^2)$. Obviously $\Phi(0) = 0$. For the induction step suppose $\Phi(i) = O(k^2)$. We distinguish between two cases according to the maximal penalty. First, suppose that there exists a color c with $P_c(i) \geq 4k$. Then, we can conclude with Proposition 2 that $\Phi(i+1) < \Phi(i) = O(k^2)$. Now, suppose that, for each colors c, $P_c(i) < 4k$. Then, after increasing the penalties, $P_c(i+1) < 5k$, for each color c. Hence, Proposition 1 yields that $\Phi(i+1) = O(k^2)$. This completes the proof of Lemma 2 (Bounded Waste). □

Now, we introduce the notion of online and offline intervals. Consider the sequence of all online color changes. The offline color changes to color c induce a partition of this sequence into *offline c-intervals*. In addition, the online color changes to color c induce a partition of this sequence into *online c-intervals*. Obviously, there are $C^c_{\text{op}} + 1$ offline c-intervals and $C^c_{\text{on}} + 1$ online c-intervals, with C^c_{op} and C^c_{on} denoting the total number of color changes to color c made by the optimal offline and Bounded Waste strategy, respectively.

Fix a color c. We show for each offline c-interval I that

$$\Delta^c(I) = W^c_{\text{op}}(I) - W^c_{\text{on}}(I) \geq W^c_{\text{on}}(I) - O(k \cdot \log^2 k) \quad , \tag{1}$$

with $W^c_{\text{op}}(I)$ and $W^c_{\text{on}}(I)$ denoting the optimal offline and online waste for color c, respectively, produced in the offline c-interval I. Then the Main Lemma follows immediately, since $\sum_{\text{off. interval } I} W^c_{\text{op}}(I) = W^c_{\text{op}}$, $\sum_{\text{off. interval } I} W^c_{\text{on}}(I) = W^c_{\text{on}}$, and there are $C^c_{\text{op}} + 1$ offline intervals.

In the following, we fix an offline c-interval I. We partition this interval into two *phases* as follows: The first phase lasts from the beginning of I until the first offline color change to a color different from c. The second phase contains the remaining part of the interval I.

In the remaining part of the proof, we show that an inequality being analogous to Inequality 1 holds for each phase. We will denote the waste produced for color c in a phase of the interval with W^c_{on} and W^c_{op}, if there is no ambiguity.

Lemma 3 (Phase 1). *Let W^c_{on} and W^c_{op} denote the waste produced for color c in the first phase of an offline c-interval. Then*

$$\Delta^c = W^c_{\text{op}} - W^c_{\text{on}} \geq W^c_{\text{on}} - O(k \cdot \log^2 k) \ .$$

Proof. In the following, the numeration of online intervals and online color changes is with respect to the first phase of the offline c-interval, e.g., if we mention the i-th online color change to color c, we refer to the i-th online color change to color c within this phase. Then let $W^c_{\text{on}}(i)$ and $W^c_{\text{op}}(i)$ denote the waste for color c produced by the Bounded Waste and optimal offline strategy, respectively, in the i-th online interval. Further, let $n^c_{\text{on}}(i)$ and $n^c_{\text{op}}(i)$ the number of items of color c in the sorting buffer of the Bounded Waste and optimal offline strategy, respectively, at the i-th online color change to color c. Note that $n^c_{\text{op}}(i)$ is monotonously decreasing, since the optimal offline strategy always removes items of color c in the first phase.

In the following proposition, we prove that either an online c-interval satisfies the inequality in Lemma 3 or $n^c_{\text{op}}(i)$ decreases by at least a factor of $1/2$.

Proposition 3. *Let i denote the number of an online c-interval that is completely contained in the first phase of an offline c-interval. Then*

$$\Delta^c(i) = W^c_{\text{op}}(i) - W^c_{\text{on}}(i) \geq W^c_{\text{on}}(i) \quad \textit{or} \quad n^c_{\text{op}}(i+1) \geq 2 \cdot n^c_{\text{op}}(i+2) \ .$$

Proof. We distinguish between the two cases $n^c_{\text{op}}(i+1) \geq 2 \cdot n^c_{\text{on}}(i+1)$ and $n^c_{\text{op}}(i+1) < 2 \cdot n^c_{\text{on}}(i+1)$. In the first case, we immediately get $W^c_{\text{op}}(i) \geq 2 \cdot W^c_{\text{on}}(i)$. In the second case, we get $n_{\text{op}}(i+1) \geq 2 \cdot n^c_{\text{op}}(i+2)$, since in the $(i+1)$-th online c-interval at least $n^c_{\text{on}}(i+1)$ items with a color different from c have to be stored in the sorting buffer. □

Since the sorting buffer has capacity k, there exists at most $O(\log k)$ online c-intervals that do not satisfy the first inequality in Proposition 3 (including the at most two online c-intervals that are not completely contained within the phase). By Lemma 2 (Bounded Waste) we know that in each online c-interval at most $O(k \cdot \log k)$ waste is produced. Hence, Lemma 3 (Phase 1) follows. □

Lemma 4 (Phase 2). *Let W^c_{on} and W^c_{op} denote the waste produced for color c in the second phase of an offline c-interval. Then*

$$\Delta^c = W^c_{\text{op}} - W^c_{\text{on}} \geq W^c_{\text{on}} - O(k \cdot \log^2 k) \ .$$

Proof. Let C^c_{on} denote the number of online color changes to color c within the second phase of the offline c-interval. We consider the online c-intervals that are completely contained within the phase. There are at most $C^c_{\text{on}} - 1$ such intervals. We define the length ℓ_i of the i-th online c-interval in the phase as the total number of online color changes within this interval. Furthermore, we define the distance d_i of the i-th online c-interval to the end of the phase as the number of online color changes in the phase that take place after the end of the i-th online c-interval.

In the following, we compare the waste that is produced by items appearing during a certain interval. Let N^c_i denote the set of items of color c that appear in the input sequence during the i-th online c-interval of the phase. Note, that these items are all

removed from the sorting buffer of the Bounded Waste strategy at the end of the online c-interval while they remain in the sorting buffer of the optimal offline strategy until the end of the phase. This holds, because the phase starts with an offline color change to a color different from c and the next offline color change to c is at the end of the phase.

Now, we study the additional waste that is produced by the optimal offline strategy for items in N_i^c during the phase. Let $W_{\text{on}}^c(N_i^c)$ and $W_{\text{op}}^c(N_i^c)$ denote the online and optimal offline waste for color c, respectively, produced for items in N_i^c. Then

$$W_{\text{op}}^c(N_i^c) - W_{\text{on}}^c(N_i^c) = |N_i^c| \cdot d_i \ ,$$

since after the i-th online c-interval is finished, there follow d_i further color changes in the phase, and at each of these color changes the optimal offline strategy produces waste $|N_i^c|$ for items in N_i^c while the online strategy produces waste zero for items in N_i^c.

Obviously, $W_{\text{on}}^c(N_i^c) = W_{\text{on}}^c(i)$, i.e., the online waste for color c produced for items in N_i^c is the same as the online waste for color c produced during the i-th online c-interval. Unfortunately, this equality does not hold for the optimal offline waste $W_{\text{op}}^c(N_i^c)$ and $W_{\text{op}}^c(i)$. However, the total optimal offline waste for color c produced during the phase is $\sum_i W_{\text{op}}^c(N_i^c) = \sum_i W_{\text{op}}^c(i)$.

We call an online c-interval *problematic* if

$$\Delta^c(N_i^c) = W_{\text{op}}^c(N_i^c) - W_{\text{on}}^c(N_i^c) < W_{\text{on}}^c(N_i^c) \ ,$$

i.e., the additional waste produced by the optimal offline strategy is less than the online waste for the respective items. The following proposition gives an upper bound on the total number of problematic intervals in the phase.

Proposition 4. *There are at most $O(\log k)$ problematic online c-intervals in the second phase of an offline c-interval.*

Proof. In order to proof the proposition we make the following observation. Suppose all problematic online c-intervals are numbered according to their appearance. Let s and t, with $s < t$, denote the numbers of two problematic online c-intervals. Then $d_s > 2 \cdot d_t$.

Assume for contradiction that $d_t \geq d_s/2$. Then the length ℓ_t of the t-th online c-interval is at most $d_s/2$. Hence, for the the online waste for color c in this interval holds $W_{\text{on}}^c(N_t) \leq \ell_t \cdot |N_t| \leq d_s/2 \cdot |N_t|$. But, since the t-th online c-interval is problematic, it follows that $W_{\text{on}}^c(N_t) > |N_t| \cdot d_t$. This is a contradiction.

It remains to give an upper bound on the distance of an problematic online c-interval. Let i denote the number of a problematic online c-interval. According to Lemma 2 (Bounded Waste), $W_{\text{on}}^c(N_i^c) = O(k \cdot \log k)$. In addition, $W_{\text{on}}^c(N_i) > d_i \cdot |N_i|$, since the i-th online c-interval is problematic. Then $d_i = O(k \cdot \log k)$.

Because the distance of two consecutive problematic online c-intervals increase by a factor of two and the distance is bounded by $O(k \cdot \log k)$, there can only be $O(\log(k \cdot \log k)) = O(\log k)$ such intervals. This yields the proposition. □

Now, we can show the lemma as follows. We first sum up the waste produced by items of problematic online c-intervals

$$\sum_{\text{probl. interval } i} (W^c_{\text{op}}(N^c_i) - W^c_{\text{on}}(N^c_i)) \geq \sum_{\text{probl. interval } i} W^c_{\text{on}}(N^c_i) - O(k \cdot \log^2 k) \ .$$

The above inequality holds, since there are at most $O(\log k)$ problematic intervals and for each problematic interval i holds $W^c_{\text{on}}(N^c_i) = O(k \cdot \log k)$, according to Lemma 2 (Bounded Waste). For each non-problematic online c-interval i that is completely contained in the phase, $W^c_{\text{op}}(N^c_i) - W^c_{\text{on}}(N^c_i) \geq W^c_{\text{on}}(N^c_i)$, by definition of problematic interval.

It only remains to consider the waste produced by items of the at most two online c-intervals that are not completely contained within the phase. According to Lemma 2 (Bounded Waste), the online waste for color c produced in these intervals is at most $O(k \cdot \log k)$. Hence, $W^c_{\text{op}}(N^c) - W^c_{\text{on}}(N^c) \geq W^c_{\text{on}}(N^c) - O(k \cdot \log k)$, with N^c denoting the set of items of color c that occur in these intervals.

Finally, we can sum up the waste produced by items of all online c-intervals, which yields Lemma 4 (Phase 2). □

This completes the proof of Lemma 1 (Main Lemma). □

References

1. T. Feder, R. Motwani, R. Panigrahy, and A. Zhu. Web caching with request reordering. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 104–105, 2002.
2. A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(2):685–699, 1991.
3. C. B. Fraser and R. W. Irving. Approximation algorithms for the shortest common supersequence. *Nordic Journal on Computing*, 2(3):303–325, 1995.
4. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Co., San Francisco, CA, USA, 1979.
5. T. Jiang and M. Li. On the approximation of shortest common supersequences and longest common subsequences. In *Proceedings of the 21st International Colloquium on Automata, Languages and Programming (ICALP)*, pages 191–202, 1994.
6. D. R. Karger, C. Stein, and J. Wein. Scheduling algorithms. In M. J. Atallah, editor, *Handbook of Algorithms and Theory of Computation*, chapter 4. CRC Press, 1997.
7. S. O. Krumke, W. de Paepe, J. Rambau, and L. Stougie. Online bin coloring. In *Proceedings of the 9th European Symposium on Algorithms (ESA)*, pages 74–85, 2001.
8. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In S. C. Graves, A. H. G. Rinnooy Kan, and P. H. Zipkin, editors, *Handbooks in Operations Research and Management Science, Vol. 4: Logistics of Production and Inventory*, chapter 9, pages 44–552. North-Holland, 1993.
9. J. Sgall. On-line scheduling. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms: The State of the Art*, volume 1442, pages 298–231. Springer LNCS, 1998.
10. D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.
11. S. Spieckermann and S. Voß. Paint shop simulation in the sutomotive industry. In *Proceedings of the Workshop for Simulation and Animation in Planning, Education, and Presentation*, pages 367–380, 1996.