Alcom-FT Technical Report Series
ALCOMFT-TR-03-93

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, Paderborn, Germany
{harry,csohler}@uni-paderborn.de
[2] International Computer Science Institute
Berkeley, California, USA
marsu@icsi.berkeley.edu

Abstract. In this paper, we introduce the online scheduling problem for sorting buffers and present deterministic scheduling strategies for this problem. 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 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.

We analyze our strategies in a competitive model in which the cost of the online strategies are compared with the cost of an optimal offline strategy. First, we show that several standard strategies are unsuitable for sorting buffers. In detail, we prove an $\Omega(\sqrt{k})$ lower bound on the competitive ratio for the First In First Out (FIFO) and Least Recently Used (LRU) strategy and an $\Omega(k)$ lower bound on the competitive ratio for the Largest Color First (LCF) strategy. Our main result 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.

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 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 present 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

* The first two authors are partially supported by the DFG-Sonderforschungsbereich 376 and the EU ESPRIT Long Term Research Project ALCOM-FT. The third author is supported by a fellowship within the ICSI-Postdoc-Program of the German Academic Exchange Service (DAAD).

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 often realized by the transfer of data packets. The sending of a data packet by a computer is denoted as *startup*. The overhead induced by the startup procedure (inclusive the overhead of the receiving processor) is denoted as *startup cost*. Sending many small packets has a negative impact on the performance, since in this case the startup cost dominate the total cost [1]. The startup cost can be reduced by combining several small data packets that are directed to the same destination. Thus, with the assistance of sorting buffers at the computers on the network the startup cost can be reduced.

File server are computer and high-capacity storage devices that 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 *open*, if it is ready to be accessed. Otherwise, it is denoted *closed*. By technical reasons, the number of open files on a file server is limited. Since the overhead induced by the opening and closing procedure takes a lot of time for a file server, it would be preferable if as many as possible accesses to an open file can be processed before closing it. Thus, 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. Such a *multi service sorting buffer* has m identical service stations and each item can be processed by any of the stations. In this example, m is the maximum number of open files on the 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., [3, 11]).

In a painting center of a car plant, a sequence of cars traverses the following two steps [2]: The first step is the basic layer painting in which each car is painted with a uniform basic coat, and the second step is the final layer painting in which each car is painted with its own top coat. If two consecutive cars are painted in different colors in the final layer painting 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. Such 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 paint station.

We analyze our online scheduling strategies for sorting buffers in a competitive model which was introduced by Sleator and Tarjan [11]. In this kind of model the cost of the online strategies are compared with the cost of an optimal offline strategy. The notion of an online process is intended to formalize the realistic scenario, where the online 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 a partial knowledge of the input.

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 assigned to the service station at last 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 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* $\mathrm{cost}(\sigma)$ *of the scheduling strategy* denote the number of color blocks in the output sequence. Then, the goal is to minimize the cost for each input sequence.

Scheduling strategies can be viewed as online processes that are typically evaluated in a competitive analysis. In this kind of analysis which was introduced by Sleator and Tarjan [11] the cost of the online strategies are 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 strategies have 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 $\mathrm{cost}_{\mathrm{opt}}(\sigma)$ 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 $c \cdot \mathrm{cost}_{\mathrm{opt}}(\sigma) + a$, 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 study only *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, then 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, then a lazy strategy does not remove an item from the sorting buffer.

1.2 Our contribution

We introduce the online scheduling problem for sorting buffers and present deterministic scheduling strategies for this problem. Our 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. We analyze our strategies in a competitive model in which the cost of the online strategies are compared with the cost of an optimal offline strategy.

We show in Section 2 that several standard strategies are unsuitable for sorting buffers. In detail, we prove an $\Omega(\sqrt{k})$ lower bound on the competitive ratio for the First In First Out (FIFO) and Least Recently Used (LRU) strategy and an $\Omega(k)$ lower bound on the competitive ratio for 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.

In addition, we have considered queue sorting buffers. The Bounded Waste strategy can be used to achieve a scheduling strategy for queue sorting buffers with k queues each with storage capacity for

d items. This is done by simulating the Bounded Waste strategy which is running on a sorting buffer with storage capacity for $(k \cdot d)$ items on the queue sorting buffer. This deterministic strategy achieves a competitive ratio of $O(d \cdot \log(d \cdot k) \cdot \log^3 k)$. Further, we have studied queue sorting buffers with k queues of infinite storage capacity. For this case, we have developed a deterministic strategy that achieves a competitive ratio of $O(k)$ which is asymptotically optimal due to a lower bound of $\Omega(k)$ on the competitive ratio of any deterministic scheduling strategy. Due to space limitations we omit these results concerning queue sorting buffers in this extended abstract.

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., [5]). After the NP-completeness results, the focus shifted to designing approximation algorithms (see, e.g., [7, 9]). Many natural heuristics for scheduling are in fact online strategies. Hence, when the study of online algorithms using competitive analysis became usual, this approach was naturally and quite successfully applied to scheduling (see, e.g., [10]).

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. [8] present a simple deterministic strategy that achieves competitive ratio $(2k + 1)$. Moreover, they give an $\Omega(k)$ lower bound on the competitive ratio of deterministic strategies.

The following variant of the static scheduling 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 supersequence problem. Fraser and Irving [4] present a polynomial time algorithm for this problem that calculates a $(4k + 12)$-approximation. Further, Jiang and Li [6] show that this problem is not in APX, i.e., for this problem exists no polynomial time algorithm with constant approximation ratio.

In several practical work, heuristic scheduling strategies are used for the queue sorting buffer problem (see, e.g., [2, 12]). This work is almost always motivated by the demand of efficient scheduling strategies for queue sorting buffers in manufacturing control. The main focus of Brückner et al. [2] lies on the implementation of a multi-agent based approach. Spieckermann and Voss [12] evaluate by simulation some simple heuristic scheduling strategies. As a result, they determine a lack of efficient scheduling strategies.

2 Lower Bounds

We consider some strategies that have previously been applied to other scheduling problems. We prove lower bounds on the competitive ratio of these strategies.

2.1 First In First Out (FIFO)

The FIFO strategy works as follows: Each time an item σ_i is stored in the sorting buffer we check whether there is another item of the same color present in the sorting buffer. If there is no such item, the color of σ_i gets a time stamp which are increasing integer numbers. Each time there is no item of the current output color in the sorting buffer the algorithm removes an item of the color that has the smallest time stamp.

Theorem 1. *The FIFO strategy is $\Omega(\sqrt{k})$-competitive for a sorting buffer of size k.*

Proof. W.l.o.g., we assume that $\ell := \sqrt{k}$ is integral and even. Let $S = \{c_1, \ldots, c_\ell, x, y\}$ be a set of colors and let τ denote a sequence of colors with $\tau = c_1 \ldots c_\ell$. For the proof we use the sequence $\sigma = (\tau x^k \tau y^k)^{\frac{\ell}{2}}$.

First, we argue that this sequence can be processed with only 2ℓ color changes. Consider the sequence $(x, y)^{\frac{\ell}{2}}, c_1, \ldots, c_\ell$ of color changes. During the first ℓ color changes of this sequence the items of colors in $\{c_1, \ldots, c_\ell\}$ are accumulated in the sorting buffer. Note that the total number of these items is only $2\ell \cdot \frac{\ell}{2} = k$, which does not exceed the size of the sorting buffer. We can conclude that the above sequence of color changes is valid and, hence, can be processed with only 2ℓ color changes.

Now, we count how many color changes are performed by the FIFO strategy. Clearly, whenever a block of x^k or y^k appears the strategy empties the whole sorting buffer, because all colors that are stored in it are "older". Each time this costs ℓ color changes because there are ℓ different colors. Hence, for the whole sequence FIFO needs at least $2\ell \cdot \frac{\ell}{2} = k$ color changes.

Thus we can conclude that the sequence σ^n can be processed with $2\sqrt{k} \cdot n$ color changes while the FIFO strategy needs $k \cdot n$ color changes. This implies that FIFO is $\Omega(\sqrt{k})$-competitive. $\square$

2.2 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. Each time the strategy makes a color change LRU selects the color with the smallest time stamp.

Theorem 2. *The LRU strategy is $\Omega(\sqrt{k})$-competitive for a sorting buffer of size k.*

Proof. It is easy to see that for the sequence defined in the proof of Theorem 1 the LRU strategy constructs the same output sequence as FIFO. Thus, LRU is also $\Omega(\sqrt{n})$-competitive. $\square$

2.3 Largest Color First (LCF)

Another fairly natural strategy for the sorting buffer problem is to free as many slots in the sorting buffer as possible, whenever the strategy has to make a color change. This leads to the so called Largest Color First (LCF) strategy which always chooses the color that has the largest number of items in the sorting buffer for the next color block. Similarly to FIFO and LRU we can also prove a large lower bound on the competitive ratio of LCF.

Theorem 3. *The LCF strategy is $\Omega(k)$-competitive for a sorting buffer of size k.*

Proof. W.l.o.g., we assume that k is even. Let $S = \{c_1, ..., c_{k-2}, x, y\}$ be a set of colors. We consider the sequence $\sigma = c_1 ... c_{k-2}(xxyy)^n$, for some integer n.

First, we show that σ can be processed with $\frac{8n}{k} + k = O(\frac{n}{k})$ color changes. Consider the sequence $c_1, ..., c_{k-2}, (x, y)^{\lceil \frac{4n}{k} \rceil}$ of color changes. This sequence is valid, since after the colors in $\{c_1, ..., c_{k-2}\}$ have been removed from the sorting buffer every further color change removes $\frac{k}{2}$ items. Hence, the $4n$ items with color x and y can be removed in only $\lceil \frac{8n}{k} \rceil$ color changes.

In contrast to this, the LCF does not remove items with colors in $\{c_1, ..., c_{k-2}\}$. Each color change to x or y only removes two items. Therefore, the LCF strategy needs at least $4n$ color changes and, hence, the strategy is $\Omega(k)$-competitive. $\square$

3 The 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 space 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 processing time.

On the other hand, we benefit from keeping an item in the sorting buffer if items with the same color arrive in the near future. We conclude that a strategy may fail as well if it removes items "too early". Good examples for this phenomenon are the LRU and FIFO strategies 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 that color appear.

What we need is 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.

In the following we describe the Bounded Waste strategy. Like all other strategies, Bounded Waste is a lazy strategy, i.e., it 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 algorithm 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 of c. The penalty of a 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 of each color is 0. At each online color change we increase the penalty of each color c by the number of items of color c currently stored in the sorting buffer. Then we choose the color with highest penalty, remove an item with this color from the sorting buffer, and reduce its penalty to 0. In the following steps the Bounded Waste strategy continues to remove items of this output color, until this is not possible anymore.

For the competitive analysis of the algorithm we have to compare the Bounded Waste strategy to an optimal offline strategy, for any input sequence σ. In the following we assume that an arbitrary sequence σ is fixed. Then the sequence of color changes of the optimal offline strategy and the online strategy, i.e., Bounded Waste, are fixed as well. In order to compare both strategies we introduce a term that we call waste for the online and for the offline algorithm. We say that an algorithm (online or offline) *produces waste w for color c at an online color change* if it has w color c items in its sorting buffer at this color change. Note that the waste of the offline algorithm is also produced at *online* color changes. Further, we define the (total) online and offline waste for color c as the total waste produced for color c at all online color changes, by the online and offline algorithm, respectively. The waste is strongly related to the penalty of a color c. The penalty for color c counts the waste produced for c by the online algorithm since the most recent online color change to c.

In the following we describe how the notion of waste can be used to derive a bound on the competitive ratio of the Bounded Waste strategy. Let W^c_{onl} and W^c_{off} denote the online and offline waste for color c. Then

$$\sum_{\text{color } c} W^c_{\text{off}} \leq k \cdot C_{\text{onl}} \qquad \text{and} \qquad \sum_{\text{color } c} W^c_{\text{onl}} \geq k \cdot C_{\text{off}} - k^2 \ .$$

These inequalities hold, since at all but the last k color changes the online as well as the offline algorithm have k items in the sorting buffer, because both algorithm 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. Each of the items in the sorting buffer produces waste and, therefore, the total waste is at most $k \cdot$

$C_{\text{onl}} - k^2$ and at most $k \cdot C_{\text{onl}}$, respectively. Let $\Delta^c = W^c_{\text{off}} - W^c_{\text{onl}}$ denote the difference between online and offline waste for color c. We immediately get

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

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 a color c let C^c_{off} denote the number of blocks with color c in the output sequence of an optimal offline algorithm. Then*

$$\Delta^c \geq W^c_{\text{onl}} - C^c_{\text{off}} \cdot O(k \cdot \log^2 k) \ .$$

We use this lemma to prove our main theorem.

Theorem 4. *The Bounded Waste strategy is $O(\log^2 k)$-competitive for a sorting buffer of size k.*

Proof. Let C_{onl} denote the number of color blocks in the output sequence of the Bounded Waste strategy and let C_{off} denote the number of color blocks in the output sequence of an optimal offline algorithm. Then

$$k^2 \geq \sum_{\text{color } c} \Delta^c \geq \sum_{\text{color } c} \left(W^c_{\text{onl}} - C^c_{\text{off}} \cdot O(k \cdot \log k^2)\right) \geq k \cdot C_{\text{onl}} - k^2 - C_{\text{off}} \cdot O(k \cdot \log^2 k) \ .$$

And hence,

$$C_{\text{off}} \cdot O(\log^2 k) + 2k \geq C_{\text{onl}} \ ,$$

which implies that Bounded Waste is $O(\log^2 k)$-competitive. □

4 The proof of the Main Lemma

In this section we present the proof of Lemma 1 (Main Lemma) which directly implies that the Bounded Waste strategy is $O(\log^2 k)$-competitive.

Proof (of Lemma 1). First we show that for each color c the maximum penalty P_c is $O(k \cdot \log k)$. By the relationship between waste and penalty this means that the waste produced by the online algorithm for color c between two consecutive online color changes to c is bounded by $O(k \cdot \log k)$. Let $P_c(i)$ denote the penalty of color c directly after the i-th online color change and let $P_c(0) = 0$.

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

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

$$\Phi(i) = \sum_{\text{color } c} \sum_{j=1}^{P_c(i)} \phi(j) \enspace ,$$

with

$$\phi(j) = 2^{\lfloor \frac{j}{k} \rfloor} + \frac{j}{k} \enspace .$$

Our goal is to show that $\Phi(i) = O(k^2)$. This immediately implies that $P_c(i) = O(k \cdot \log k)$, for each color c. We do this by combining the observation that the start potential $\Phi(0) = 0$ with the following two propositions.

Proposition 1. *If for all colors c holds $P_c(i) \leq 5k$ 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 is a color c with $P_c(i) \geq 4k$ then $\Phi(i+1) < \Phi(i)$.*

Proof. Let us define $\Delta\Phi(i+1) = \Phi(i+1) - \Phi(i)$, i.e., the change in potential at the $i+1$-th online color change. Recall that the Bounded Waste strategy first increases the penalty of each color, then it chooses the color with the highest penalty and evicts one item of that color from the sorting buffer and then reduces the highest penalty to zero. Let m denote the penalty of the evicted color after it has been increased (and before it is set to 0).

Then

$$\Delta\Phi(i+1) \leq k \cdot \phi(m) - \sum_{j=1}^{m} \phi(j) \enspace ,$$

because each of the k items in the sorting buffer contributes at most $\phi(m)$ to the potential. Thus

$$\Delta\Phi(i+1) \leq k \cdot 2^{\lfloor \frac{m}{k} \rfloor} - \sum_{j=1}^{m} 2^{\lfloor \frac{j}{k} \rfloor} + m - \sum_{j=1}^{m} \frac{j}{k} \enspace .$$

By exploiting that $\sum_{j=1}^{\ell} 2^j = 2^{\ell+1} - 1$, we get

$$\Delta\Phi(i+1) \leq k + m - \frac{m \cdot (m+1)}{2k} = k - \frac{m \cdot (m - 2k + 1)}{2k} \enspace .$$

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

Combining the Propositions 1 and 2 we yield the following proposition.

Proposition 3. *Let i denote an arbitrary online color change. Then $\Phi(i) = O(k^2)$.*

Proof. We proceed by induction on i. Obviously $\Phi(0) = 0$. By induction hypothesis we get $\Phi(i) = O(k^2)$. We distinguish between two cases according to the maximum penalty of a color. First, suppose that there is a color c with $P_c(i) \geq 4k$. Then, by Proposition 2 it follows that $\Phi(i+1) < \Phi(i) = O(k^2)$. Now, suppose that, for all colors c, $P_c(i) < 4k$. Then, after adding the k new units of waste, we still have $P_c(i) < 5k$ for all colors c. By Proposition 1 it follows that $\Phi(i+1) = O(k^2)$. □

It is easy to see that $\Phi(i) = O(k^2)$ implies that $P_c(i) = O(k \cdot \log k)$ which proves Lemma 2. $\square$

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 intervals called *offline* c*-intervals*. In a similar way, this sequence is partitioned by the *online color changes to color* c. An interval of this induced partition is called an *online* c*-interval*.

Obviously, there are $C^c_{\text{off}} + 1$ offline c-intervals and $C^c_{\text{onl}} + 1$ online c-intervals, with C^c_{off} and C^c_{onl} denoting the total number of color changes to c made by the offline and online algorithm, respectively.

We fix a color c and consider the offline c-intervals. We show for each offline c-interval I that

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

with $W^c_{\text{onl}}(I)$ and $W^c_{\text{off}}(I)$ denoting the online and offline waste, respectively, produced in offline interval I for color c. If the above inequality holds for each interval I, the lemma follows immediately, since

$$\sum_{\text{off. interval } I} W^c_{\text{off}}(I) = W^c_{\text{off}} \qquad \text{and} \qquad \sum_{\text{off. interval } I} W^c_{\text{onl}}(I) = W^c_{\text{onl}} \ ,$$

and because there are only $C^c_{\text{off}} + 1$ offline intervals.

In the following, we fix a single 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 lasts from the first offline color change to a color different from c until the end of the interval and, thus, contains the rest of interval I.

Now, we show that an inequality that is analogously to inequality 1 holds for each phase. For convenience we will denote the waste produced for color c in a phase of the interval with W^c_{onl} and W^c_{off}, if there is no ambiguity.

Lemma 3 (Phase 1). *Let W^c_{onl} and W^c_{off} denote the waste produced in the first phase of the offline c-interval I. Then*

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

Proof. In this proof, the numeration of online intervals and color changes is with respect to the first phase of the offline c-interval I. That is, when 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. In a similar way let us denote the waste produced by the online algorithm in the i-th online interval as $W^c_{\text{onl}}(i)$ and the waste produced by offline as $W^c_{\text{off}}(i)$. Further we denote by $n^c_{\text{onl}}(i)$ and $n^c_{\text{off}}(i)$ the number of items of color c in the online and offline sorting buffer, respectively, at the i-th color change to color c. We also observe that $n^c_{\text{off}}(i)$ is monotonously decreasing, since the offline algorithm always replaces items of color c.

Now we prove that either an online c-interval satisfies the requirements from Lemma 3 or $n^c_{\text{off}}(i)$ decreases by at least a factor of $1/2$.

Proposition 4. *Let i denote the number of an arbitrary online c-interval that is completely contained in the first phase of I. Then either*

$$\Delta^c(i) = W^c_{\text{off}}(i) - W^c_{\text{onl}}(i) \geq W^c_{\text{onl}}(i)$$

or

$$n^c_{\text{off}}(i+1) \geq 2 \cdot n^c_{\text{off}}(i+2) \ .$$

Proof. We distinguish between the case that $n^c_{\text{off}}(i+1) \geq 2 \cdot n^c_{\text{onl}}(i+1)$ and the case that $n^c_{\text{off}}(i+1) < 2 \cdot n^c_{\text{onl}}(i+1)$. In the first case, we get $n^c_{\text{off}}(i) \geq n^c_{\text{onl}}(i+1)$, since $n^c_{\text{off}}(i) - n^c_{\text{onl}}(i)$ is monotonously decreasing and then $W^c_{\text{off}}(i) \geq 2 \cdot W^c_{\text{onl}}(i)$. In the second case, we get $n_{\text{off}}(i+1) \geq 2 \cdot n^c_{\text{off}}(i+2)$, because in the $(i+1)$-th online interval at least $n^c_{\text{onl}}(i+1)$ items with color c are replaced by items with another color. □

Since the sorting buffer has size k there can be at most $O(\log k)$ online c-intervals that do not satisfy the inequality (also counting the two online intervals not completely contained in the phase). By the Bounded Waste Lemma we know that in each interval at most $O(k \cdot \log k)$ waste is created. Hence, Lemma 3 follows. □

Lemma 4 (Phase 2). *Let W^c_{onl} and W^c_{off} denote the waste produced in the second phase of the offline interval. Then*

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

Proof. Let C^c_{onl} denote the number of online color changes to color c within the phase. We consider the online c-intervals that are completely contained in the phase. These are are at most $C^c_{\text{onl}} - 1$ intervals. We define the length ℓ_i of the i-th online interval of the phase as the total number of its (online) color changes. Furthermore, we denote with d_i the number of color changes in the phase that take place after the end of the i-th interval. This will be called the distance of the i-th interval to the end of the phase.

We have to compare the online and offline waste for color c that is produced during the phase. Therefore, we compare the waste that is produced by items that appear during a certain interval. Let N_i denote the set of color c items that occur in the input sequence during the i-th online c-interval of the phase. Note, that these items are all removed from online's sorting buffer at the end of the online interval while they remain in offline's sorting buffer until the phase ends. This holds, because the phase starts with an offline color change to a color different from c. Hence, in order to remove a color c item, the offline strategy has to do a color change to c. But this is not possible, because the next offline color change to c is at the end of the phase.

Now, we can specify the additional waste that is produced by the optimal offline strategy for items in N_i during the phase as follows:

$$W^c_{\text{off}}(N_i) - W^c_{\text{onl}}(N_i) = |N_i| \cdot d_i \ ,$$

with $W^c_{\text{onl}}(N_i)$ and $W^c_{\text{off}}(N_i)$ denoting the online and offline waste, respectively, produced for items in N_i for color c. The above equality holds, since after the i-th interval is finished, there follow d_i further color changes in the phase. On each of these color changes the optimal offline strategy produces waste $|N_i|$ for items of N_i while the online strategy produces waste 0.

Note that $W^c_{\text{onl}}(N_i)$ equals $W^c_{\text{onl}}(i)$, i.e., the online waste for items in N_i is the same as the online waste produced for color c during the i-th interval. However, this does not hold for the offline waste(s) $W^c_{\text{off}}(N_i)$ and $W^c_{\text{off}}(i)$. But obviously the total offline waste produced during the phase can be expressed as $\sum_i W^c_{\text{off}}(N_i) = \sum_i W^c_{\text{off}}(i)$.

We call an online interval *problematic* if

$$\Delta^c(N_i) = W^c_{\text{off}}(N_i) - W^c_{\text{onl}}(N_i) > W^c_{\text{onl}}(N_i) \ ,$$

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 a bound on the total number of problematic intervals in the phase.

Proposition 5. *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: Let s and t, with $s < t$, denote the numbers of two problematic online intervals. Then $d_s > 2 \cdot d_t$.

Assume for contradiction that $d_t \geq \frac{d_s}{2}$. Then the length ℓ_t of the t-th online interval is smaller than $\frac{d_s}{2}$. This implies that the online waste $W^c_{\text{onl}}(N_t)$ for items of this interval is at most $\ell_t \cdot |N_t| \leq \frac{d_s}{2} \cdot |N_t|$. But, as the t-th interval is problematic, it follows that $W^c_{\text{onl}}(N_t) > |N_t| \cdot d_t$. This is a contradiction.

It remains to show that the distance of an interval is bounded. Let i denote the number of a problematic interval. Then $W^c_{\text{onl}}(N_i) = O(k \cdot \log k)$, according to the Bounded Waste Lemma. The fact, that for a problematic interval $d_s \cdot |N_s| < W^c_{\text{onl}}(N_s)$, yields $d_s = O(k \cdot \log k)$.

As the distances of two consecutive problematic online intervals increase by a factor of 2 and the distance is bounded by $O(k \cdot \log k)$ there can only be $\log(O(k \cdot \log k)) = O(\log k)$ 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 intervals

$$\sum_{\text{probl. interval } i} (W^c_{\text{off}}(N_i) - W^c_{\text{onl}}(N_i)) \geq \sum_{\text{probl. interval } i} W^c_{\text{onl}}(N_i) - O(k \cdot \log^2 k) \ .$$

The above equation holds, since there are only $O(\log k)$ problematic intervals and for each interval i is $W^c_{\text{onl}}(N_i) = O(k \cdot \log k)$.

For all other online intervals that are completely contained in the phase we know that $W^c_{\text{off}}(N_i) - W^c_{\text{onl}}(N_i) \geq W^c_{\text{onl}}(N_i)$, due to the definition of problematic intervals. It only remains to consider the waste produced by items of the at most 2 online intervals that are not completely contained in the phase. According to Lemma 2 the online waste for color c produced in these intervals is $O(k \cdot \log k)$ and, thus, $W^c_{\text{off}}(N_i) - W^c_{\text{onl}}(N_i) \geq W^c_{\text{onl}}(N_i) - O(k \cdot \log k)$ holds for these intervals, as well.

Now, we can sum up the waste produced by items of all intervals and we yield Lemma 4. □

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

References

1. O. Bonorden, B. Juurlink, I. von Otte, and I. Rieping. The Paderborn University BSP (PUB) library – design, implementation and performance. In *Proceedings of the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing (IPPS/SPDP)*, 1999.
2. S. Brückner, J. Wyns, P. Peeters, and M. Kollingbaum. Designing agents for manufacturing control. In *Proceedings of the 2nd AI and Manufacturing Research Planning Workshop*, 1998.
3. 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.
4. C. B. Fraser and R. W. Irving. Approximation algorithms for the shortest common supersequence. *Nordic Journal of Computing*, 2:303–325, 1995.
5. M. J. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. Freeman, 1979.
6. 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.
7. D. Karger, C. Stein, and J. Wein. Scheduling algorithms. In M. J. Atallah, editor, *Handbook of Algorithms and Theory of Computation*. CRC Press, 1997.
8. S. O. Krumke, W. E. De Paepe, J. Rambau, and L. Stougie. Online bin-coloring. In *Proceedings of the 9th European Symposium on Algorithms (ESA)*, 2001.

9. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling: Algorithms and comlexity. In S. C. Graves, A. H. G. Rinnooy Kan, and P. Zipkin, editors, *Handbooks in Operations Research and Management Science, Vol. 4: Logistics of Production and Inventory*, pages 44–552. North-Holland, 1993.
10. J. Sgall. On-line scheduling. In A. Fiat and G. Woeginger, editors, *Online Algorithms: The State of the Art*, volume 1442, pages 298–231. Springer LNCS, 1998.
11. D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.
12. S. Spieckermann and S. Voss. Paint shop simulation in the automotive industry. In *Proceedings of the Workshop for Simulation and Animation in Planning, Education, and Presentation*, pages 367–380, 1996.