Evaluation of Online Strategies for Reordering Buffers*

Matthias Englert Heiko Röglin Matthias Westermann

Department of Computer Science
RWTH Aachen, D-52056 Aachen, Germany
{englert, roeglin, marsu}@cs.rwth-aachen.de

Abstract

A sequence of objects which are characterized by their color has to be processed. Their processing order influences how efficiently they can be processed: Each color change between two consecutive objects produces costs. A reordering buffer which is a random access buffer with storage capacity for k objects can be used to rearrange this sequence online in such a way that the total costs are reduced. This concept is useful for many applications in computer science and economics.

The strategy with the best known competitive ratio is MAP. An upper bound of $O(\log k)$ on the competitive ratio of MAP is known and a non-constant lower bound on the competitive ratio is not known [EW05]. Based on theoretical considerations and experimental evaluations, we give strong evidence that the previously used proof techniques are not suitable to show an $o(\sqrt{\log k})$ upper bound on the competitive ratio of MAP. However, we also give some evidence that in fact MAP achieves a competitive ratio of $O(1)$.

Further, we evaluate the performance of several strategies on random input sequences experimentally. MAP and its variants RC and RR clearly outperform the other strategies FIFO, LRU, and MCF. In particular, MAP, RC, and RR are the only known strategies whose competitive ratios do not depend on the buffer size. Furthermore, MAP achieves the smallest competitive ratio.

*The first and the last author are supported by DFG grant WE 2842/1. The second author is supported by DFG grants VO 889/2 and WE 2842/1. An earlier version of this work appeared in Proceedings of the 5th International Workshop on Experimental Algorithms (WEA), 2006 [ERW06].

1 Introduction

Frequently, a number of tasks has to be processed and their processing order influences how efficiently they can be processed. Hence, a reordering buffer can be expedient to influence the processing order. This concept is useful for many applications in computer science and economics. In the following, we give an example (for further examples see [BL05, EW05, GSV04, KP04, KP06b, RSW02]).

In computer graphics, a rendering system displays 3D scenes which are composed of primitives. In current rendering systems, the state changes performed by the graphics hardware are a significant factor for the performance. A state change occurs when two consecutively rendered primitives differ in their attribute values, e. g., in their texture or shader program. These state changes slow down a rendering system. To reduce the costs of the state changes, a reordering buffer can be included between application and graphics hardware. Such a reordering buffer which is a random access buffer with limited memory capacity can be used to rearrange the incoming sequence of primitives online in such a way that less state changes occur [KRSW04].

1.1 The Model

An *input sequence* $\sigma = \sigma_1\sigma_2\cdots$ of objects which are only characterized by a specific attribute has to be processed. To simplify matters, we suppose that the objects are characterized by their color, and, for each object σ_i, let $c(\sigma_i)$ denote the color of σ_i. A *reordering buffer* which is a random access buffer with storage capacity for k objects can be used to rearrange the input sequence in the following way.

The first object of σ that is not handled yet can be stored in the reordering buffer, or objects currently stored in the reordering buffer can be removed. These removed objects result in an *output sequence* $\sigma_{\pi^{-1}} = \sigma_{\pi^{-1}(1)}\sigma_{\pi^{-1}(2)}\cdots$ which is a permutation of σ. We suppose that the reordering buffer is initially empty and, after processing the whole input sequence, the buffer is empty again.

For an input sequence σ, let $C^A(\sigma)$ denote the *costs of a strategy* A, i. e., the number of color changes in the output sequence. The goal is to minimize the costs $C^A(\sigma)$.

The notion of an online strategy is intended to formalize the realistic scenario, where the strategy does not have knowledge about the whole input sequence in advance. The online strategy has to serve the input sequence

σ one after the other, i. e., a new object is not issued before there is a free location in the reordering buffer.

Online strategies are typically evaluated in a competitive analysis. In this kind of analysis the costs of the online strategy are compared with the costs of an optimal offline strategy. For an input sequence σ, let $C^{\mathrm{OPT}}(\sigma)$ denote the costs produced by an optimal offline strategy. An online strategy is denoted as α-*competitive* if it produces costs at most $\alpha \cdot C^{\mathrm{OPT}}(\sigma) + \kappa$, for each sequence σ, where κ is a term that does not depend on σ. The value α is also called the *competitive ratio* of the online strategy.

1.2 The Strategies

We only consider *lazy* strategies, i. e., strategies that fulfill the following two properties.

- If an additional object can be stored in the buffer, a lazy strategy does not remove an object from the buffer.
- An *active color* is selected, and, as long as objects with the active color are stored in the buffer, a lazy strategy does not make a color change.

Hence, a lazy strategy has only to specify how to select a new active color. Note that every strategy, in particular every optimal offline strategy, can be transformed into a lazy strategy without increasing the costs.

First-In-First-Out (FIFO). This strategy assigns time stamps to each color stored in the buffer. Initially, the time stamps of all colors are undefined. When an object is stored in the buffer and the color of this object has an undefined time stamp, the time stamp is set to the current time. Otherwise, it remains unchanged. FIFO selects as new active color the color with the oldest time stamp and resets this time stamp to undefined. This is a very simple strategy that does not analyze the input stream. The buffer acts like a sliding window over the input stream in which objects with the same color are combined.

Least-Recently-Used (LRU). Similar to FIFO, this strategy assigns time stamps to each color stored in the buffer. Initially, the time stamps of all colors are undefined. When an object is stored in the buffer, the time stamp of its color is set to the current time. LRU selects as new active color the color with the oldest time stamp and resets this time stamp to undefined. LRU and also FIFO tend to remove objects from the buffer too early [RSW02].

Most-Common-First (MCF). This fairly natural strategy tries to clear as many locations as possible in the buffer, i. e., it selects as new active color a color that is most common among the objects currently stored in the buffer. MCF also fails to achieve good performance guarantees since it keeps objects with a rare color in the buffer for a too long period of time [RSW02]. This behavior wastes valuable storage capacity that could be used for efficient buffering otherwise.

Maximum-Adjusted-Penalty (MAP). This strategy, which is introduced and analyzed in a non-uniform variant of our model in [EW05], provides a trade-off between the storage capacity used by objects with the same color and the chance to benefit from future objects with the same color. We present an adapted version of MAP for our uniform model which is similar to the Bounded-Waste strategy [RSW02]. A penalty counter is assigned to each color stored in the buffer. Informally, the penalty counter for color c is a measure for the storage capacity that has been used by all objects of color c currently stored in the buffer. Initially, the penalty counters for all colors are set to 0. MAP selects as new active color a color with maximal penalty counter and the penalty counters are updated as follows: The penalty counter for each color c is increased by the number of objects of color c currently stored in the buffer, and the penalty counter of the new active color is reset to 0.

Random-Choice (RC). Since the computational overhead of MAP is relatively large, we present more practical variants of MAP. RC which is a randomized version of MAP selects as new active color the color of an uniformly at random chosen object from all objects currently stored in the buffer. Note that RC can also be seen as a randomized version of MCF. Even if RC is much simpler than MAP, random numbers have to be generated.

Round-Robin (RR). This strategy is a very efficient variant of RC. It uses a selection pointer which points initially to the first location in the buffer. RR selects as new active color the color of the object the selection pointer points to and the selection pointer is shifted in a round robin fashion to the next location in the buffer. We suppose that RR has the same properties as RC on typical input sequences.

1.3 Previous Work

Räcke, Sohler, and Westermann [RSW02] show that several standard strategies are unsuitable for a reordering buffer, i. e., the competitive ratio of FIFO and LRU is $\Omega(\sqrt{k})$ and the competitive ratio of MCF is $\Omega(k)$, where k denotes the buffer size. Further, they present the deterministic Bounded-Waste strategy (BW) and prove that BW achieves a competitive ratio of $O(\log^2 k)$.

Englert and Westermann [EW05] study a non-uniform variant of our model: Each color change to color c produces non-uniform costs b_c. As main result, they present the deterministic MAP strategy and prove that MAP achieves a competitive ratio of $O(\log k)$.

The offline variant of our model is studied in [BL05, KP04]. However, the goal is to maximize the number of saved color changes. Note that an approximation of the minimum number of color changes is preferable, if it is possible to save a large number of color changes. Kohrt and Pruhs [KP04] present a polynomial-time offline algorithm that achieves an approximation ratio of 20. Further, they mention that optimal algorithms with running times $O(n^{k+1})$ and $O(n^{m+1})$ can be obtained by using dynamic programming, where k denotes the buffer size and m denotes the number of different colors. Bar-Yehuda and Laserson [BL05] study a more general non-uniform maximization variant of our model. They present a polynomial-time offline algorithm that achieves an approximation ratio of 9.

Khandekar and Pandit [KP06a, KP06b] consider reordering buffers on a line metric. This metric is motivated by an application to disc scheduling: Requests are categorized according to their destination track on the disk, and the costs are defined as the distance between start and destination track. For a disc with n uniformly-spaced tracks, they present a randomized online strategy and show that this strategy achieves a competitive ratio of $O(\log^2 n)$ in expectation against an oblivious adversary [KP06b]. They also present a quasi-polynomial-time offline algorithm that achieves a constant approximation ratio [KP06a].

Krokowski et al. [KRSW04] examine the previously mentioned rendering application. They use a small reordering buffer (storing less than hundred references) to rearrange the incoming sequence of primitives online in such a way that the number of state changes is reduced. Due to its simple structure and its low memory requirements, this method can easily be implemented in software or even hardware. In their experimental evaluation, this method typically reduces the number of state changes by an order of magnitude and the rendering time by roughly 30%. Furthermore, this method typi-

cally achieves almost the same rendering time as an optimal, i. e., presorted, sequence without a reordering buffer.

1.4 Our Contributions

In Section 2, we study the worst case performance of MAP. Recall that an upper bound of $O(\log k)$ on the competitive ratio of MAP is known and a non-constant lower bound on the competitive ratio is not known [EW05]. Hence, a natural question is whether it is possible to improve the upper bound on the competitive ratio of MAP. The proof of the upper bound consists of two parts. First, it is shown that the competitive ratio of MAP_{4k} against OPT_k is 4, where A_n denotes the strategy A with buffer size n and OPT denotes an optimal offline strategy. Finally, it is proven that the competitive ratio of OPT_k against OPT_{4k} is $O(\log k)$. As we see, the logarithmic factor is lost solely in the second part of the proof.

We present theoretical considerations and experimental results which give strong evidence that the competitive ratio of OPT_k against OPT_{4k} is $\Omega(\sqrt{\log k})$. This implies that the previously used proof techniques are not suitable to prove an $o(\sqrt{\log k})$ upper bound on the competitive ratio of MAP. However, we also give some evidence that in fact MAP achieves a competitive ratio of $O(1)$.

In Section 3, we evaluate the performance of several strategies on random input sequences experimentally. MAP and its variants RC and RR clearly outperform the other strategies FIFO, LRU, and MCF. In particular, MAP, RC, and RR are the only known strategies whose competitive ratios do not depend on the buffer size.

2 Worst Case Performance of MAP

In Section 2.1, we give an alternative proof that the competitive ratio of OPT_k against OPT_{4k} is $O(\log k)$ in our uniform model. This proof is based on a potential function. In Section 2.2, we exploit properties of this potential function to deterministically generate input sequences which give strong evidence that this result cannot be improved much. In more detail, based on our experimental evaluation in Section 2.3, we conjecture that the competitive ratio of OPT_k against OPT_{4k} is $\Omega(\sqrt{\log k})$. As a consequence, the proof technique in [EW05], which is also implicitly contained in the proof of [RSW02], is not suitable to show an $o(\sqrt{\log k})$ upper bound on the competitive ratio of MAP.

2.1 Theoretical Foundations

In this section, we give an alternative proof for the following theorem.

Theorem 1 ([EW05]). *The competitive ratio of* OPT_k *against* OPT_{4k} *is* $O(\log k)$.

Proof. Fix an input sequence σ and an optimal offline strategy OPT_{4k}. Let $\sigma_{\pi^{-1}}$ denote the output sequence of OPT_{4k}. Suppose that $\sigma_{\pi^{-1}}$ consists of m color blocks $B_1, \dots B_m$, i. e., $\sigma_{\pi^{-1}} = B_1 \cdots B_m$ and all objects in each color block have the same color and the objects in each color block B_i have a different color than the objects in color block B_{i+1}. Let $c(B_i)$ denote the color of the objects in color block B_i. Without loss of generality assume that $c(B_1) = 1, c(B_2) = 2, \dots c(B_m) = m$, i. e., the color of each color block is different from the colors of the other color blocks. This does not change the costs of OPT_{4k} and can obviously only increase the costs of OPT_k.

Consider the execution of a strategy A, and fix a time step (a new time step begins with each storage or removal of an object). We denote a color c as *finished* if all objects of color c have occurred in the output sequence of A. Otherwise, color c is denoted as *unfinished*. Let $f = \min\{c | c \text{ is unfinished}\}$ denote the *first unfinished color*, and let $d(c) = c - f$ denote the *distance of color* c. Then, the potential of color c is defined as $\Phi(c) = n(c) \cdot d(c)$, where $n(c)$ denotes the number of objects of color c currently stored in the buffer of A. For each color c, we define a counter $p(c)$, initially set to 0. Intuitively, the counter $p(c)$ indicates how many objects with a color strictly larger than c have occurred in the output sequence of A. Whenever A moves an object of color c to the output sequence, for each $f \leq i < c$, $p(i)$ is increased by one.

Now, we describe the simple algorithm GREEDY_k (f, $d(c)$, $n(c)$, $\Phi(c)$, and $p(c)$ are defined w. r. t. GREEDY_k). Note that the accumulated potential Φ, which is initially set to 0, is introduced but not used in this algorithm.

1. Calculate the first unfinished color f. As long as $n(f) \neq 0$, move objects of color f to the output sequence. If color f becomes finished, repeat step 1.

2. Calculate a color $q \in \arg\max_c \Phi(c)$ with maximum potential. Move $n(q)$ objects of color q to the output sequence. Increase the accumulated potential Φ by $\Phi(q)$. Proceed with step 1.

Observe that GREEDY_k is an offline algorithm since it has to know the output sequence of OPT_{4k}. In the following, it is shown that the competitive ratio of GREEDY_k against OPT_{4k} is $O(\log k)$.

The following lemma provides an upper bound on the counters. It implies for the accumulated potential $\Phi \leq 8k \cdot m$ since the accumulated potential Φ can also be expressed as $\Phi = \sum_c p(c)$.

Lemma 2. *For each color c, $p(c) \leq 8k$.*

Proof. Observe that $p(f) \geq p(f+1) \geq \cdots \geq p(m)$ and that counters for colors less than f do not change their values anymore. Hence, it suffices to show that $p(f) \leq 8k$. This is done by induction over the iterations of GREEDY$_k$. Fix an iteration of GREEDY$_k$. We distinguish the following two cases.

- Suppose that $p(f) \leq 7k$ at the beginning of the iteration.

  Then, $p(f) \leq 8k$ at the end of this iteration since $p(f)$ is increased by at most k in step 2 and the counters are only increased in step 2.

- Suppose that $p(f) > 7k$ at the beginning of the iteration.

  Then, GREEDY$_k$ has moved more than $7k$ objects with a color larger than f to its output sequence. Due to its buffer size, OPT$_{4k}$ has moved more than $3k$ of these objects to its output sequence. However, this implies that OPT$_{4k}$ has moved the last object of color f to its output sequence more than $3k$ time steps ago. Hence, the last object of color f has already entered the buffer of GREEDY$_k$. As a consequence, the unfinished color f becomes finished in step 1 of this iteration.

This concludes the proof of the lemma. □

Due to the following lemma, each iteration of GREEDY$_k$ increases the accumulated potential Φ by at least $\frac{k}{1+\ln k}$.

Lemma 3. *If $n(f) = 0$ and the buffer contains k objects,* $\max_c \Phi(c) \geq \frac{k}{1+\ln k}$.

Proof. First of all, observe that $\sum_{c>f} n(c) = k$, since, for each color $c \leq f$, $n(c) = 0$ and the buffer contains k objects. Define $q = \max_c \Phi(c)$. Obviously, for each $i \geq 1$, $n(f+i) \leq \lfloor q/i \rfloor$. In particular, for each $i > q$, $n(f+i) = 0$. Hence,

$$k = \sum_{i=1}^{q} n(f+i) \leq \sum_{i=1}^{q} \frac{q}{i} = q \cdot H_q \ ,$$

where $H_q = \sum_{i=1}^{q} \frac{1}{i}$ denotes the q-th harmonic number.

Suppose that $q < \frac{k}{1+\ln k}$. Then

$$k \leq q \cdot H_q < q \cdot H_k \leq q \cdot (1 + \ln k) < k \ ,$$

which is a contradiction. □

Combining the results of the two lemmata above yields that there are at most $8m \cdot (1 + \ln k)$ executions of step 2 of GREEDY$_k$. The number of executions of step 1 can exceed the number of executions of step 2 by at most m since step 1 is only repeated when a color becomes finished. Hence, GREEDY$_k$ generates at most $16m \cdot (1 + \ln k) + m + k$ color changes. Recall that OPT$_{4k}$ generates $m - 1$ color changes. This concludes the proof of the theorem. □

2.2 Generating Input Sequences

In this section, we describe our approach to deterministically generate input sequences for which MAP$_k$ loses a logarithmic factor compared to OPT$_{8k}$. To some extend, the buffers sizes are chosen arbitrarily. Our construction can be generalized canonically to MAP$_k$ and OPT$_a$, for each $a > k$.

The main idea is to use the accumulated potential Φ defined in the proof of Theorem 1. The generated input sequences consist of objects with m different colors, and at most $8k - 1$ objects of each of the m colors. The sequences are intended to have the property that MAP$_k$ can increase the accumulated potential Φ by only $O(k/\log k)$ with each color change, and the accumulated potential Φ is $\Omega(m \cdot k)$ after the sequences are processed. As a consequence, MAP$_k$ makes $\Omega(m \cdot \log k)$ color changes for these input sequences. However, OPT$_{8k}$ is able to rearrange these input sequences in such a way that the objects of each color form a consecutive block, i. e., the number of color changes made by OPT$_{8k}$ is $m - 1$. Hence, MAP$_k$ should lose an $\Omega(\log k)$ factor compared to OPT$_{8k}$.

The following algorithm for generating deterministic input sequences is based on the proof of Lemma 3. The first $8k - 1$ objects are, for each $1 \leq i \leq \Theta(k/\log k)$, $\lceil q/i \rceil$ objects of color i, with $q = 8k/\log k$. Then, the algorithm proceeds in phases corresponding to the last unfinished color f. At the beginning of phase f, let $n(c)$ denote the number of objects of color c currently stored in the buffer of MAP$_k$, and let $s(c)$ denote the number of objects of color c included in the input sequence so far. In phase f, $s(f)$ objects of colors larger than f followed by the last object of color f are appended to the input sequence.

At the beginning of phase f, the algorithm tries to restore a situation in which the accumulated potential Φ can only be increased by $O(k/\log k)$ and OPT_{8k} is still able to rearrange the input sequence properly. At the beginning of phase f, the length of the input sequence created so far is $8k-1+s(1)+s(2)+\cdots+s(f-1)$. Observe that $s(1)+s(2)+\cdots+s(f-1)$ of these objects have colors smaller than f and $8k-1$ of these objects have colors larger or equal to f. Hence, the number of objects having a color larger than f so far is $8k-1-s(f)$. Due to the restriction that OPT_{8k} is able to rearrange the input sequence into an output sequence with only $m-1$ color changes, at most $8k-1$ objects with a color larger than f can precede the last object of color f. Hence, at most $s(f)$ objects of colors larger than f can be appended before the last object of color f is appended to the input sequence.

According to Lemma 3, the algorithm should achieve $n(f+i) \approx q/i$. Hence, $\max\{0, \lceil q \rceil - n(f+1)\}$ objects of color $f+1$, $\max\{0, \lceil q/2 \rceil - n(f+2)\}$ objects of color $f+2$, ... are appended to the input sequence, until altogether $s(f)$ objects have been appended in this phase. Then, the phase is finished by appending the last object of color f to the input sequence. In Figure 1, we present the pseudocode of the algorithm for generating the input sequences.

We expect that the accumulated potential Φ is $\Omega(m \cdot k)$ after the generated input sequence has been processed by MAP_k. To see this, recall that, for each color f, the last object of color f is preceded by $8k-1$ objects of colors larger than f. Hence, MAP_k moves at least $7k-1$ of these objects to the output sequence before moving the last object of color f to the output sequence, and, as a consequence, $p(f) \geq k$. For GREEDY_k, we know that $\Phi = \sum_c p(c)$. This is not necessarily true for MAP_k. However, this is true for a slightly differently defined potential $\Phi(c) = n'(c) \cdot d(c)$. This potential is not based on the number $n(c)$ of objects of color c currently stored in the buffer, but on the number $n'(c)$ of objects of color c moved to the output sequence when changing to color c. Observe that $n(c)$ and $n'(c)$ differ only if during moving the objects of color c to the output sequence additional objects of this color arrive. We expect that for the generated input sequences $n(c)$ and $n'(c)$ usually do not differ much.

2.3 Experimental Evaluation

Figure 2 depicts the competitive ratios of MAP_k against OPT_{8k} on the generated input sequences for buffer sizes $k_1, \ldots k_{85}$ with $k_1 = 1000$ and $k_i = \lfloor k_{i-1} \cdot 11/10 \rfloor + 1$. A regression analysis with functions of the type $a \cdot \ln k + b$ results in $0.918109 \cdot \ln k + 1.33176$ where the sum of the squared

```
// notations
q := 8k/ log k;
q_i := ⌈q⌉ + ⌈q/2⌉ + ··· + ⌈q/i⌉ for i ≥ 1;

// first 8k − 1 objects
j := max_i{q_i < 8k − 1};
for i := 1 to j do append(⌈q/i⌉, i);
append(8k − 1 − q_j, j + 1);

// phases f = 1, ... m − 1
for f := 1 to m − 1 do
    n(c) := # objects of color c currently stored in the buffer of MAP_k;
    s(c) := # objects of color c included in the input sequence so far;

    // append s(f) + 1 elements to the sequence
    rem := s(f);
    for j := 1 to m − f do
        append(min{max{0, ⌈q/j⌉ − n(f + j)}, rem}, f + j);
        rem := rem − min{max{0, ⌈q/j⌉ − n(f + j)}, rem};
    end for

    while rem > 0 do
        for j := 1 to m − f do
            append(min{⌈q/j⌉, rem}, f + j);
            rem := rem − min{⌈q/j⌉, rem};
        end for
    end while

    append(1, f);
end for
```

Figure 1: The pseudocode of the algorithm for generating the input sequences. The statement *append*(x, y) is used to add x elements of color y to the sequence.

residuals is 0.0303205. Using functions of the type $a \cdot \ln k + b \cdot \ln \ln k + c$ yields $0.843897 \cdot \ln k + 0.786948 \cdot \ln \ln k + 0.279742$ where the sum of the squared residuals is only 0.00472631.

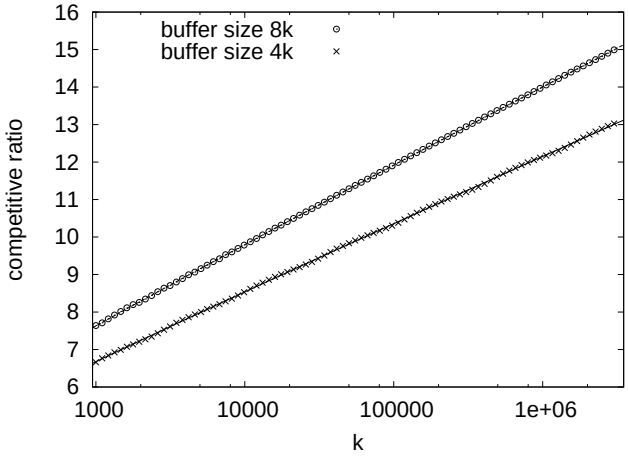

Figure 2: Competitive ratios of MAP_k against OPT_{4k} and OPT_{8k} on the generated input sequences and resulting functions for regression analysis with $a \cdot \ln k + b \cdot \ln \ln k + c$.

Further, Figure 2 depicts the competitive ratios of MAP_k against OPT_{4k} on the generated input sequences. Unfortunately, there are periodic fluctuations in these competitive ratios which results in a larger sum of squared residuals. However, a regression analysis with functions of the type $a \cdot \ln k + b \cdot \ln \ln k + c$ results in $0.730515 \cdot \ln k + 0.615999 \cdot \ln \ln k + 0.440412$ where the sum of the squared residuals is 0.0373066 and no residual is greater than 0.043559.

Based on the experimental evaluation, we conjecture the following.

Conjecture 4. *The competitive ratio of* MAP_{4k} *against* OPT_{32k} *is* $\Omega(\log k)$.

Now, we can conclude the following theorem. If we take the experimental evaluation for smaller factors between the buffer sizes into account, we can make the stronger conjecture that the competitive ratio of MAP_{4k} against OPT_{16k} is $\Omega(\log k)$, and the $o(\sqrt[3]{\log k})$ term in the following theorem improves to $o(\sqrt{\log k})$.

Theorem 5. OPT_k *cannot achieve a competitive ratio of* $o(\sqrt[3]{\log k})$ *against* OPT_{4k} *if Conjecture 4 is true.*

Proof. Suppose for contradiction that the competitive ratio of OPT_k against OPT_{4k} is $o(\sqrt[3]{\log k})$. Then, the competitive ratio of OPT_k against OPT_{64k}

is $o(\log k)$. In the first part of the proof of Theorem 4 in [EW05] it is shown that the competitive ratio of MAP_{4k} against OPT_k is 4. As a consequence, the competitive ratio of MAP_{4k} against OPT_{64k} is $o(\log k)$ which is a contradiction to Conjecture 4. □

Our actual interest is the competitive ratio of MAP. Is it possible to show a non-constant lower bound on the competitive ratio of MAP or to improve the upper bound? Based on our experimental evaluation, the proof technique in [EW05, RSW02] is not suitable to show an $o(\sqrt{\log k})$ upper bound on the competitive ratio of MAP since this would require that the competitive ratio of OPT_k against OPT_{4k} is $o(\sqrt{\log k})$.

However, we have evidence that MAP achieves in fact a competitive ratio of $O(1)$ in our uniform model. MAP is always optimal, i. e., it achieves a competitive ratio of 1, for the generated input sequences. In addition to the following observations, this indicates a small competitive ratio of MAP. Each $\Omega(\sqrt{\log k})$ lower bound on the competitive ratio of MAP implies an $\Omega(\sqrt{\log k})$ lower bound on the competitive ratio of OPT_k against OPT_{4k}. Hence, the input sequences used in such a lower bound have to assure that the potential gained in step 2 of GREEDY_k is not too large. However, our sequences are constructed to have exactly this property. As a consequence, any major modification to our generated input sequences will probably fail to show an $\Omega(\sqrt{\log k})$ lower bound on the competitive ratio of MAP.

3 Random Input Sequences

In this section, we evaluate the performance of several strategies on random input sequences experimentally. Since an efficient optimal offline algorithm is not known, we cannot simply generate random input sequences and evaluate the performance of the strategies by comparing their number of color changes with the optimal number of color changes. Therefore, we first introduce a technique to generate random input sequences with known optimum. Finally, the experimental evaluation is presented in detail.

3.1 Input Sequences with Known Optimum

Fix an input sequence σ and an optimal offline strategy OPT_k. Let σ^{opt} denote the output sequence of OPT_k. Suppose that σ^{opt} consists of m color blocks $B_1, \ldots B_m$, i. e., $\sigma^{\text{opt}} = B_1 \cdots B_m$ and all objects in each color block have the same color and the objects in each color block B_i have a different color than the objects in color block B_{i+1}. Without loss of generality assume

that the color of each color block is different from the colors of the other color blocks. This does not change the costs of OPT_k and can obviously only increase the costs of any other strategy.

The following result is given in [EW05]: For each input sequence σ, the permutation $\sigma_{\pi^{-1}} = \sigma_{\pi^{-1}(1)}\sigma_{\pi^{-1}(2)}\cdots$ of $\sigma = \sigma_1\sigma_2$ is an output sequence of a strategy with buffer size k if and only if $\pi^{-1}(i) < i+k$, for each i. Hence, a random input sequence with known optimal number of color changes can be generated as follows. First, we determine an output sequence σ^{opt} of OPT_k. This output sequence is completely characterized by the number of color blocks m and the color block lengths $l_1, \ldots l_m$, i. e., l_i denotes the number of objects in the i-th color block. Then, a permutation π with $\pi^{-1}(i) < i + k$, for each i, is chosen uniformly at random among all permutations with this property. In this way, we get a random input sequence $\sigma_\pi^{\mathrm{opt}}$ for which OPT_k makes $m-1$ color changes. Observe that different permutations can lead to the same input sequence.

3.2 Experimental Evaluation

We evaluate the performance of FIFO, LRU, MAP, MCF, RC, and RR on different kinds of random input sequences experimentally.

Constant color block lengths. Figure 3 depicts the competitive ratios of the strategies for buffer sizes $k_1, \ldots k_{139}$ with $k_1 = 10$ and $k_i = \lfloor k_{i-1} \cdot 21/20 \rfloor + 1$ on generated input sequences with $m = k_i + 1$ and color block lengths $l_1 = \cdots = l_m = k_i$. For each buffer size, we average over 50 runs. The variances are very small and decreasing with increasing buffer sizes. For buffer sizes larger than 1000, the variances are below 0.006.

The competitive ratios of FIFO and LRU increase with the buffer size on these non-malicious inputs. RC and RR presumably achieve small constant competitive ratios. MCF and MAP achieve the best competitive ratios. MCF is optimal for buffer sizes greater than 49, and, for buffer sizes greater than 317, MAP is also optimal.

Uniformly chosen color block lengths. Figure 4 depicts the competitive ratios of the strategies for buffer sizes $k_1, \ldots k_{139}$ on the following generated input sequences. Let $u_1, u_2, \ldots$ denote a sequence of independent random variables distributed uniformly between 1 and k. Then, $m = \max_i\{u_1 + \cdots + u_i < k^2 + k\} + 1$ and, for $1 \le i < m$, $l_i = u_i$ and $l_m = k^2 + k - (u_1 + \cdots + u_{m-1})$. For each buffer size, we average over 50

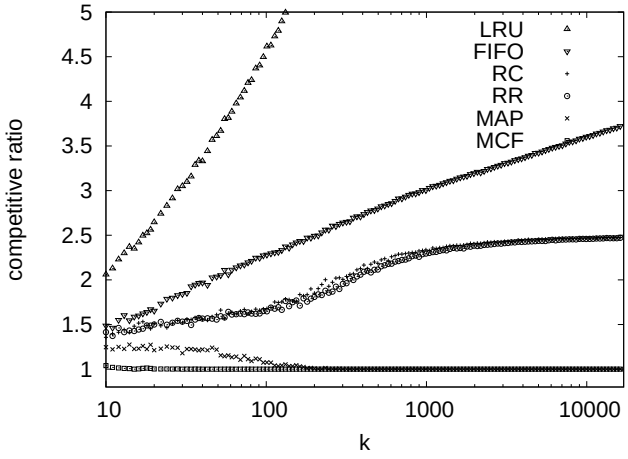

Figure 3: Competitive ratios on random input sequences with constant color block lengths.

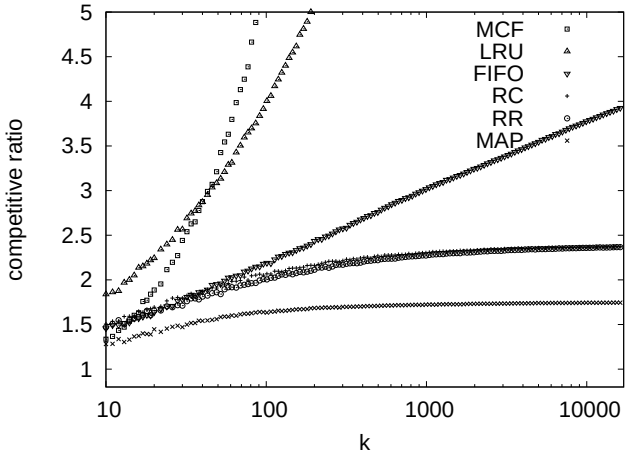

Figure 4: Competitive ratios on random input sequences with uniformly chosen color block lengths.

runs. The variances, except for MCF, are very small and decreasing with increasing buffer sizes. For buffer sizes larger than 1000, the variances, except for MCF, are below 0.006.

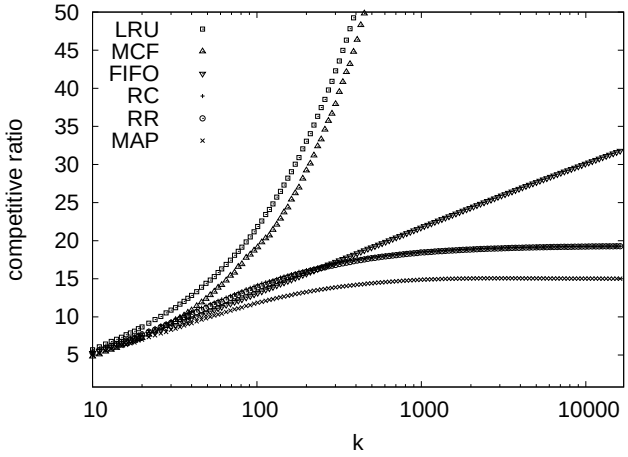

Figure 5: Competitive ratios of the strategies with buffer size k against an optimal offline strategy with buffer size $8k$ on random input sequences with constant color block lengths.

The competitive ratios of FIFO, LRU, and, in contrast to the first set of input sequences, MCF increase with the buffer size on these non-malicious input sequences. MAP, RC, and RR presumably achieve small constant competitive ratios.

Different buffer sizes. Figure 5 depicts the competitive ratios of the strategies with buffer size k_i against an optimal offline strategy with buffer size $8k_i$ for $k_1, \ldots k_{139}$ on generated input sequences with $m = k_i + 8$ and color block lengths $l_1 = \cdots = l_m = k_i$. For each buffer size, we average over 50 runs. The variances, except for LRU and MCF, are very small and decreasing with increasing buffer sizes. For buffer sizes larger than 1000, the variances, except for LRU and MCF, are below 0.003.

The competitive ratio of MAP with buffer size k against an optimal offline strategy with buffer size $8k$ is presumably constant. Hence, these experiments justify the sophisticated deterministic generation of input sequences we used to obtain Conjecture 4, as they show that random input sequences do not suffice for that purpose.

References

[BL05] R. Bar-Yehuda and J. Laserson. 9-approximation algorithm for sorting buffers. In *Proceedings of the 3rd Workshop on Approximation and Online Algorithms (WAOA)*, 2005.

[ERW06] M. Englert, H. Röglin, and M. Westermann. Evaluation of online strategies for reordering buffers. In *Proceedings of the 5th International Workshop on Experimental Algorithms (WEA)*, pages 183–194, 2006.

[EW05] 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)*, pages 627–638, 2005.

[GSV04] K. Gutenschwager, S. Spieckermann, and S. Voss. A sequential ordering problem in automotive paint shops. *International Journal of Production Research*, 42(9):1865–1878, 2004.

[KP04] J. Kohrt and K. Pruhs. A constant approximation algorithm for sorting buffers. In *Proceedings of the 6th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 193–202, 2004.

[KP06a] R. Khandekar and V. Pandit. Offline sorting buffers on line. In *Proceedings of the 17th International Symposium on Algorithms and Computation (ISAAC)*, 2006. To appear.

[KP06b] R. Khandekar and V. Pandit. Online sorting buffers on line. In *Proceedings of the 23rd Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 584–595, 2006.

[KRSW04] 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)*, pages 217–224, 2004.

[RSW02] H. Räcke, C. Sohler, and M. Westermann. Online scheduling for sorting buffers. In *Proceedings of the 10th European Symposium on Algorithms (ESA)*, pages 820–832, 2002.