Reordering Buffer Management with Advice

Anna Adamaszek · Marc P. Renault · Adi Rosén · Rob van Stee

Abstract In the reordering buffer management problem, a sequence of coloured items arrives at a service station to be processed. Each colour change between two consecutively processed items generates some cost. A reordering buffer of capacity k items can be used to preprocess the input sequence in order to decrease the number of colour changes. The goal is to find a scheduling strategy that, using the reordering buffer, minimizes the number of colour changes in the given sequence of items.

We consider the problem in the setting of online computation with advice. In this model, the colour of an item becomes known only at the time when the item enters the reordering buffer. Additionally, together with each item entering the buffer, we get a fixed number of advice bits, which can be seen as information about the future or as information about an optimal solution (or an approximation thereof) for the whole input sequence. We show that for any $\varepsilon > 0$ there is a $(1+\varepsilon)$-competitive algorithm for the problem which uses only a constant (depending on ε) number of advice bits per input item. This also immediately implies a $(1+\varepsilon)$-approximation algorithm which has $2^{O(n \log 1/\varepsilon)}$ running time (this should be compared to the trivial optimal algorithm which has a running time of $k^{O(n)}$).

We complement the above result by presenting a lower bound of $\Omega(\log k)$ bits of advice per request for any 1-competitive algorithm.

Keywords reordering buffer management · online algorithms · online algorithms with advice · competitive analysis

1 Introduction

The reordering buffer management problem was introduced in 2002 by Räcke et al. [24]. The problem models a service station that must process a sequence of coloured items. At any time, the service station is configured to process items of a certain colour c. Changing the configuration of a service station to a different colour generates a cost. The total cost is the number of colour switches performed while serving a sequence of items. In order to reduce this cost, the service station is equipped with a reordering buffer that has the capacity to hold k items. The service station can process any of the items contained in its buffer. Using this reordering buffer, the goal is to find a schedule that minimizes the number of colour switches. In the online setting, the colour of each item is known only once the item enters the reordering buffer (see Section 2 for a formal definition).

A preliminary version of this paper appeared in the Proc. of the 11th Workshop on Approximation and Online Algorithms (WAOA 2013); LNCS, 2013, pp.132-143.

The work was performed while the first and the fourth author were at the Max-Planck-Institut für Informatik, Saarbrücken, Germany.

The first author is supported by the DFF-MOBILEX mobility grant from the Danish Council for Independent Research. The second and third authors were partially supported by ANR project NeTOC.

Anna Adamaszek
University of Copenhagen, Denmark
E-mail: anad@di.ku.dk

Marc P. Renault
CNRS and Université Paris Diderot, France
E-mail: mrenault@liafa.univ-paris-diderot.fr

Adi Rosén
CNRS and Université Paris Diderot, France
E-mail: adiro@liafa.univ-paris-diderot.fr

Rob van Stee
University of Leicester, Leicester, United Kingdom
E-mail: rob.vanstee@le.ac.uk

This problem is a natural model for job shops such as paint shops and blow moulding facilities, where there is a large cost in terms of time and materials to switch colours (paint or plastic) due to the need to clean out the old colour and set-up the equipment in order to correctly produce the new colour. For this reason, such production plants are equipped with a reordering buffer to preprocess the sequence of incoming items (for example cars that need to be painted) or production orders [19]. This framework has many applications in various areas, such as production engineering, storage systems, network optimization and computer graphics (see [5,8, 19,22,24] for more details).

In this paper, we study the problem in the setting of *online computation with advice* [16]. This setting has received much attention in recent years as it relaxes the traditional online setting of no information about the future, and allows the online algorithm to get some partial information about future requests or about the structure of an optimal (or near-optimal) solution. In this model, the amount of information about the future available to the online algorithm is quantified and the interplay between the amount of information and the attainable competitive ratio is studied. Informally (see Section 2.1 for a formal definition), in this setting, the online algorithm receives with each request some b bits of *advice* which are the value of a function, defined by the algorithm, of the whole input sequence (including the future). In this manner the online algorithm receives some information about the future. Note that we view the advice as given by an oracle, i.e., we do not consider whether or not the advice can be computed efficiently when the whole input sequence is known.

1.1 Related work

Since its introduction in 2002 [24], the reordering buffer management problem has been extensively studied. In the online setting, the best known results are a deterministic $O(\sqrt{\log k})$-competitive algorithm by Adamaszek et al. [2], and a randomized $O(\log\log k)$-competitive algorithm by Avigdor-Elgrabli and Rabani [7]. To complement this, there are nearly matching lower bounds of $\Omega(\sqrt{\log k/\log\log k})$ and $\Omega(\log\log k)$ on the competitive ratio of any online deterministic and randomized algorithms, respectively [2]. In the offline setting, the problem is known to be NP-hard [12,4], and the best known result is a constant-factor deterministic approximation algorithm by Avigdor-Elgrabli and Rabani [6].

The deterministic online algorithm by Adamaszek et al. [2] combines two different mechanisms. One is based on penalizing colours for using up space in the buffer at the time when a colour change occurs, ensuring that colours do not stay in the buffer for too long, blocking valuable resources. The other mechanism marks colours with a similar number of elements, which together occupy a large fraction of the space in the buffer. When a colour change is required, one of the marked colours is chosen to be output. The randomized online algorithm by Avigdor-Elgrabli and Rabani [7] has two steps, which are executed in parallel. The first one computes a fractional solution to an LP relaxation to the problem, based on the online primal-dual schema. The solution is updated deterministically after each step of the algorithm. In the second stage, the fractional solution is rounded, using a randomized online algorithm. The offline algorithm by Avigdor-Elgrabli and Rabani [6] is based on an intricate rounding of a solution of an LP relaxation of the problem.

More general versions of the reordering buffer management problem have been studied, where the switching cost for switching from an item of colour c to an item of colour c' depends on c' (e.g. [18,2]), or on both c and c' [17].

The model of online computation with advice considered in the present paper was introduced by Emek et al. [16]. In that paper, the authors give tight bounds of $\Theta(\log n/b)$ on the competitive ratio of deterministic and randomized online algorithms with advice for metrical task systems, where n is the number of states of the system and b is the number of advice bits per request. They also give a deterministic online algorithm with advice for the k-server problem which is $k^{O(\frac{1}{b})}$-competitive, where $\Theta(1) \leq b < \log k$. This has been first improved by Böckenhauer et al. [10], and subsequently by Renault and Rosén [25] to $\left\lceil \frac{\lceil \log k \rceil}{b-2} \right\rceil$. In this model, the advice is online and the total amount of advice is at least linear in the length of the request sequence.

Böckenhauer et al. [11] introduced a different model for online algorithms with advice. In their model, all bits of advice are written to an infinite advice tape prior to the execution of the algorithm. The algorithm can read the bits of advice as needed, and therefore it is possible that the algorithm uses a total number of advice bits that is sub-linear with respect to the length of the request sequence. The advice in this model is semi-online, as the algorithm can read the entire advice string prior to serving the request sequence. Both models are inspired by the model of online computation with advice originally proposed by Dobrev et al. [13], wherein the oracle could provide information without a cost by using an empty string as the advice for a request, and the focus was on optimality. Several results have been given in the semi-online model [13,11, 20,21,10]. For example, in [13,11], the authors explore

the number of bits of advice required for deterministic and randomized paging algorithms, algorithms for the DiffServ problem, algorithms for a special case of the job shop scheduling problem, and algorithms for the disjoint path allocation problem, to be 1-competitive.

Algorithms for the online advice model can be run for the semi-online advice model, and lower bounds for the semi-online advice model apply to the online advice model.

In this work, we present an algorithm for the reordering buffer management problem that, for any $\varepsilon > 0$, has a competitive ratio of $1 + \varepsilon$, using only $O(\log(1/\varepsilon))$ advice bits per request. This algorithm has a flavour of a polynomial-time approximation scheme (PTAS). That is, with the number of advice bits per request that is a constant depending on ε (as compared to a running time that is polynomial in ε for a PTAS), we achieve a solution that has a competitive ratio of $1 + \varepsilon$. We term this type of algorithm with advice as a *linear advice approximation scheme (LAAS)*. Renault et al. [26] present a LAAS for the Bin Packing problem, and for a family of scheduling problems on m identical machines that includes the objectives of minimizing the makespan, minimizing the ℓ_p norm, and machine covering. Also, an algorithm for the scheduling problem on m identical machines with the objective of minimizing the makespan is presented by Albers and Hellwig [3]. That paper does not explicitly consider the advice model, but it considers an online algorithm that is able to maintain multiple schedules and outputs the best schedule at the end of the execution. In particular, they present an algorithm that has a competitive ratio of $(1+\varepsilon)$ and maintains $(m/\varepsilon)^{O(\log(1/\varepsilon)/\varepsilon)}$ schedules. In the semi-online advice model, the advice consisting of $O(\log(1/\varepsilon)/\varepsilon)\log(m/\varepsilon)$ advice bits in total would indicate the index of the best schedule. This result was rediscovered by Dohrau [14] explicitly for the semi-online advice model.

Dorrigiv et al. [15] consider a transmission control problem with a buffer, called the buffer management problem. Their problem is different from the problem considered in this work. Specifically, their buffer is a FIFO buffer of capacity k items, meaning that each item inserted into the buffer is always placed at the end, and the items from the buffer are processed sequentially. Each item is assigned a value, which corresponds to the profit gained by processing the item. When a new item arrives, the algorithm has to decide whether the item should be inserted into the buffer and therefore eventually processed, or rejected. If the buffer is full, the item must be rejected. The goal is to maximize the total value of the processed items (i.e., the total value of the items inserted into the buffer). Dorrigiv et al. consider the problem when there are items with two different values (high and low). They show that $\Theta((n/k)\log k)$ bits of advice in total are required for an optimal solution. Notice that for this problem one can easily get an optimal solution with one bit of advice per item, where the bit of advice tells us whether the item should be inserted into the buffer or rejected.

1.2 Contributions

This is the first work on the reordering buffer management problem in the advice framework. First, we present an online algorithm with advice that uses two bits of advice per request and has a competitive ratio of 1.5. We then extend this algorithm by allowing it to use more bits of advice per request in order to obtain a better competitive ratio. Specifically, for any $\varepsilon > 0$, the algorithm has a competitive ratio of $1 + \varepsilon$, using $O(\log(1/\varepsilon))$ advice bits per request, i.e., it is a LAAS. Unlike for other problems where a LAAS is known [26, 3, 14], there is no offline PTAS known for the reordering buffer management problem. In fact, our result can be interpreted as an offline $(1 + \varepsilon)$-approximation algorithm with a running time of $2^{O(n \log 1/\varepsilon)}$ (see Conclusions). This is a favorable trade-off compared to the trivial optimal algorithm which uses a dynamic program of size $k^{O(n)}$, and shows an unexpected benefit of studying online algorithms with advice.

Similar to the other linear advice approximation schemes, the advice sequence is constructed based on some feasible solution. That is, when the advice sequence is constructed based an optimal solution, as described below, the algorithm has a competitive ratio of $1 + \varepsilon$. If the advice was constructed based on some c-approximation solution, then the algorithm would output a $c(1 + \varepsilon)$-competitive solution. This makes the algorithm useful even if the oracle constructing the advice sequence based on the complete input sequence is computationally bounded. For instance, the known approximation algorithm [6] could be used for that.

For any input sequence, we show how to construct the advice, based on an optimal solution for the given sequence, which allows us to obtain a good competitive ratio. The overview of the construction is as follows. The advice bits for each element of a colour c encode how the algorithm should handle all "adjacent" elements of colour c, i.e., whether the algorithm should keep all elements of colour c in the buffer until more items of colour c arrive, output them at once, or output them, but only after a certain waiting period. The idea is that the order of processing the colours with the latter property is not contained in the advice of single elements (as that would require too many bits of advice),

but it is encoded among all advice bits of the elements of the given colour. The key obstacle is that with a small number of advice bits per item we cannot encode the exact order in which the colours should be output, in particular when a small number of elements of some colour has to wait for a long time in the buffer. To deal with this problem, we modify the optimal solution (increasing its cost, i.e., making the solution sub-optimal) by selecting some elements which will be removed earlier from the buffer. This frees additional space in the buffer, which allows us to keep some other elements longer in the buffer, until one more item of their colour arrives, after which we can output these elements immediately. This significantly shortens the list of colours which have to be output "soon, but not yet". After this operation, we can encode the desired position in the list for all but a small fraction of elements (which also must be removed earlier from the buffer). We bound the increase in the cost of the generated output sequence from above by charging the additional colour changes to the colour changes of the optimal solution.

We complement these results by presenting a lower bound of $\Omega(\log k)$ on the number of bits of advice per item needed to obtain a competitive ratio of 1. Notice that if we allow using $\lceil \log k \rceil$ bits of advice per item, we can easily obtain an optimal algorithm, as we can use the advice to encode the index in the buffer of an item which should be processed at the current step of the algorithm.

2 Preliminaries

2.1 The Advice Model

We use the definition of *deterministic online algorithms with advice* as presented in [16]. An online algorithm is defined as a request-answer game that consists of a request set R, a sequence of finite non-empty answer sets $A_1, A_2, \ldots$, and a sequence of cost functions $\text{cost}_n : R^n \times A_1 \times A_2 \times \cdots \times A_n \to \mathbb{R}^+ \cup \{\infty\}$ for $n = 1, 2, \ldots$. In addition, there is an advice space U of size 2^b, where $b \geq 0$ is the number of bits of advice provided to the algorithm with each request. With each request, the online algorithm receives some advice that is defined by a function, $u_i : \sigma \to U$, where i is the request index, that is applied to the whole request sequence $\sigma = R^*$, including future requests. A deterministic online algorithm with advice can, thus, be represented as a sequence of pairs (g_i, u_i), where g_i is the function defining the action of the online algorithm at step i and is defined $g_i : R^i \times U^i \to A_i$ for $i = 1, 2, \ldots$. The action taken by the online algorithm after receiving request r_i is therefore a function of the first i requests, $r_1, \ldots, r_i$, and the advice received so far, $u_1(\sigma), \ldots, u_i(\sigma)$.

We use the standard definitions of competitive analysis. We say that an algorithm ALG is γ-*competitive* if, for every finite request sequence σ, we have $\text{ALG}(\sigma) \leq \gamma \cdot \text{OPT}(\sigma) + \zeta$, where ζ is a constant which does not depend on the request sequence σ. Here $\text{ALG}(\sigma)$ denotes the cost of the solution generated by ALG for σ, and $\text{OPT}(\sigma)$ is the cost of an optimal solution for σ.

In this paper, we use regular expressions to define the language to which strings of advice belong, and we follow the standard notations (cf. [23], Section 1.8, Finite representations of languages and [1], Section 9, Regular Expressions).

2.2 The Lower Bound Technique

The general lower bound technique which we are using for lower bounding the number of advice bits per request was introduced by Emek et al. [16]. They introduced the generalized matching pennies problem (GMP), showed a lower bound on the number of advice bits required for GMP, and then presented reductions from GMP to other online problems to obtain lower bounds for them.

An instance of the (Φ, τ)-GMP problem consists of a sequence of requests σ, where each request $r_i \in \sigma$ corresponds to a value from the set $\{1, \ldots, \Phi\}$. The online algorithm has to output a sequence of actions A, where for each action $a_i \in A$ we have $a_i \in \{1, \ldots, \Phi\}$. The action a_i generates cost $(1 + 1/\tau)$ if $a_i \neq r_i$, otherwise it generates only a dummy cost of $1/\tau$. The difficulty of the problem is that the action a_i has to be output before the algorithm has access to the request r_i. In [16], Emek et al. prove that for $\Phi \geq 4$, any deterministic or randomized algorithm with advice for GMP that uses $b \leq \frac{\log \Phi}{3}$ bits of advice per request is $(1 + \Omega(\tau))$-competitive.

In [9], Böckenhauer et al. introduce a similar problem of string guessing with known history, defined formally as follows.

Definition 1 ([9]) The *string guessing problem with known history* over an alphabet Σ of size $q \geq 2$ (q-SGKH) is an online minimization problem. The input consists of an integer n and a request sequence $\sigma = r_1, \ldots, r_n$, where $r_i \in \Sigma$. An online algorithm A outputs a sequence of answers $a_1, \ldots, a_n$ such that $a_i = f_i(n, r_1, \ldots, r_{i-1}) \in \Sigma$ for some computable function f_i, i.e., the algorithm knows the requests $r_1, \ldots, r_{i-1}$ before it has to output a_i. The cost of A is the Hamming distance between $a_1, \ldots, a_n$ and $r_1, \ldots, r_n$, i.e., the number of wrongly guessed characters.

The q-SGKH problem is equivalent to the GMP problem without the dummy cost of $1/\tau$, where the parameter q corresponds to Φ from GMP. In [9], Böckenhauer et al. prove the following result, which is a refinement of the result of [16] for the case of deterministic algorithms. Notice that this result holds for any $\Phi \geq 2$, and the range for b is increased to the full range of $[1, \log \Phi]$.

Theorem 1 ([9]) *Consider an input string of length n for q-SGKH. The minimum number of advice bits per request for any online algorithm that is correct for more than αn characters, for $1/q \leq \alpha < 1$, is $(1 - H_q(1 - \alpha)) \log_2 q$, where $H_q(p) = p \log_q(q-1) - p \log_q p - (1-p) \log_q(1-p)$ is the q-ary entropy function.*

This result is useful for establishing lower bounds on the number of bits of advice required for obtaining a certain competitive ratio. We note that Theorem 1 is stronger than stated, as it holds even when the whole advice sequence is received in advance, since, in the proof of Theorem 1 in [9], such an assumption is made. When we apply this theorem in Section 7.4, we also make the assumption that the whole advice sequence is known in advance.

2.3 Reordering Buffer Management Problem

In the *reordering buffer management problem* (RBM), we are given a random access buffer of size k, and a finite request sequence $\sigma = r_1, \ldots, r_n$. Each request $r_i \in \sigma$ (also called *item*) is characterized by a *colour* $c(r_i)$. We denote by $\mathcal{C}_\sigma = \langle c(r_1), \ldots, c(r_n) \rangle$ the sequence of colours corresponding to the consecutive requests of σ. At each time, some colour c is set as the *service colour*. Initially, the buffer is empty and the service colour is not set to any colour. The algorithm does not know the length n of the input sequence, nor the number of colours in the sequence.

Definition 2 ([24]) The *current request* (or *current item*) of the request sequence σ is the first item in σ that has not yet entered the buffer.

Let the request $r \in \sigma$ be the current request. The algorithm does not know the colour $c(r)$ until r enters the buffer. As long as the buffer is not empty or there exists a current request, the algorithm can perform exactly one of the following actions.

- Any item in the buffer with the same colour as the service colour can be output, i.e., removed from the buffer.
- The service colour can be switched to a different colour.
- If there exists a current request $r \in \sigma$ and the buffer is not full, r can enter the buffer and the next request in σ becomes the current request.

We say that an item $r \in \sigma$ is *served* when it is removed from the buffer. The cost of serving a request sequence σ equals the number of colour switches performed. The goal is to serve σ generating minimum cost.

The following observation is immediate from the fact that any algorithm must perform a number of colour switches greater or equal to the number of distinct colours in the request sequence.

Observation 1 *The number of distinct colours in the request sequence σ is a lower bound on the optimal cost.*

Definition 3 ([24]) For an input sequence σ and an algorithm ALG, a *colour block* is a maximal sequence of items from σ with the same colour that are output consecutively by ALG.

Note that all elements of a colour block are output by the algorithm ALG on a single colour switch. Note also that ALG can have items from more than one colour block of the same colour in the buffer if the algorithm switches to another colour before serving all elements of the current service colour.

Definition 4 ([24]) An algorithm for the reordering buffer management problem is called *lazy* if it has the following two properties.

- If the buffer is not full and there exists a current item $r \in \sigma$, r is immediately brought into the buffer (i.e., the algorithm can output an item or change the service colour only when the buffer is full or there is no current item).
- The algorithm only changes the service colour when there are no items in the buffer with the current service colour.

Without loss of generality, we will always assume that OPT is lazy. We can do this as any algorithm ALG can be transformed into a lazy algorithm ALG_{LAZY} at no additional cost [24]. Note that a lazy algorithm can have only one colour block of a given colour in the buffer.

In this paper, the notion of time corresponds to the index of the current request. That is, when the current request is r_τ, it is time τ. Hence, at time τ, the requests $r_1, \ldots, r_{\tau-1}$ have already entered the buffer and, possibly, have been output. An online algorithm does not know the colour $c(r_\tau)$ at time τ (or the associated advice bits). An algorithm with advice has received the advice bits for $r_1, \ldots, r_{\tau-1}$. After the last element of the sequence r_n has entered the buffer, the time advances to $n+1$ and the algorithm must serve all the

remaining items in the buffer. We say that an item r_i is *older* than r_j, and that r_j is *younger* that r_i, if $i < j$. Without loss of generality, we assume that the oldest item is output whenever a lazy algorithm has more than one item of the current service colour in the buffer.

Consider a colour block B of items output by a lazy OPT, beginning at some time τ. In the following lemma, we show that a lazy algorithm ALG can output all the items of B with a single colour switch, beginning at time τ or later. Before we prove the lemma, we describe some notations.

Definition 5 For a colour c, let a *c-buffer* be a buffer that ignores items of colour c and stores items of any colour other than c.

We can view the serving of a colour as follows. Consider a colour block B that is served by a lazy algorithm ALG. Suppose that ALG has z items of colour c in its buffer at time τ, when it switches to the colour c. Algorithm ALG stops serving colour c exactly after z items that are not of colour c enter the buffer after time τ. For such a colour block B and algorithm ALG, we define an algorithm ALG_B that starts at time τ with an empty c-buffer of size z. Then, the algorithm ALG_B begins reading the pending input of ALG and stops when the c-buffer is full. Let τ' be the time step when the c-buffer of ALG_B is full. At the time τ', the buffer of ALG no longer contains any items of colour c. It contains the z items that the c-buffer of ALG_B contains at time τ', together with the $k - z$ items that are not of colour c that were in the buffer of ALG at time τ.

Lemma 1 *Let $B = \{e_0, \ldots, e_{\ell-1}\}$ be a colour block output by a lazy* OPT *with a single colour switch that occurs at time τ. Consider a lazy algorithm* ALG*. If, for some i, $0 \le i \le \ell - 1$, at some time $\tau' \ge \tau + i$, e_i is the oldest item of B in the buffer of* ALG*, then, with a single colour switch that occurs at time τ',* ALG *can output all the items of B not output before τ'.*

Proof Recall that time corresponds to the index of the current request. That is, the time advances when the current request enters the buffer, and serving items does not affect the time.

At time τ', if $e_{\ell-1}$ is in the buffer of ALG, then ALG can clearly output all the items of B not yet output with a single colour switch.

Consider the case when $e_{\ell-1}$ is not in the buffer of ALG at time τ'. In this case, OPT is still outputting B at time τ'. At time τ, OPT has m items of colour c in its buffer for some $m \in \{1, \ldots, \ell - 1\}$. Thus, OPT_B has a c-buffer of size m. Denote the set of items not of colour c that arrive in the time interval $[\tau+1, \tau']$ by Y, and the set of items of colour c that arrive in the time interval $[\tau + 1, \tau']$ by C. Note that

$$|C| = \tau' - \tau - |Y| \ge i - |Y| \ . \tag{1}$$

The items of $|Y|$ are stored by OPT (and OPT_B) in the buffer. Hence, by the time τ', OPT_B has $m - |Y|$ free slots in its c-buffer.

Algorithm ALG switches to colour c at time τ'. It has already read all the items of Y (the buffer may still contain some of the items of Y) and read $m + |C|$ items of B. Since ALG no longer has the items $e_0, \ldots, e_{i-1}$ in its buffer at time τ', ALG has $m + |C| - i$ items of colour c in its buffer at time τ' and ALG_B has a c-buffer of size $m + |C| - i \ge m - |Y|$ (from (1)), which is not smaller than the free space of OPT_B at time τ'. Therefore, ALG_B will read the input at least as far as OPT_B and, in particular, ALG will output colour block B completely. □

3 1.5-Competitive Algorithm with Two Bits of Advice per Request

In this section, we present an algorithm with a competitive ratio of exactly 1.5 and uses two bits of advice per request. The advice received with each item of the input indicates the manner in which the item is served by a lazy optimal algorithm.

3.1 Definition of the Advice and the Algorithm

The Advice. Each item $e \in \sigma$ is assigned a *type*, denoted by t_e. The three possible values for t_e are HOLD, LIST, and COMP. For an item $e \in \sigma$, the two advice bits corresponding to e encode the value of t_e. The sequence of the item types t_e for all the elements $e \in \sigma$ constructed as described below will be denoted by $T_{2\text{BL}}(\sigma)$.

The sequence $T_{2\text{BL}}(\sigma)$ of types is defined based on an arbitrary lazy optimum OPT in the following manner. For each $e \in \sigma$, consider the colour block B of OPT that contains e (see Figure 1). Let f be the youngest item in B that was in the buffer of OPT at the time when OPT began serving B. We set t_e to

- COMP if e is the last (youngest) item of B; B is then considered to be *complete*,
- HOLD if e is older than f, i.e., if OPT will only start serving B after at least one more item of B has arrived,
- LIST for all other items, i.e., e is not older than f and not the youngest item of B.

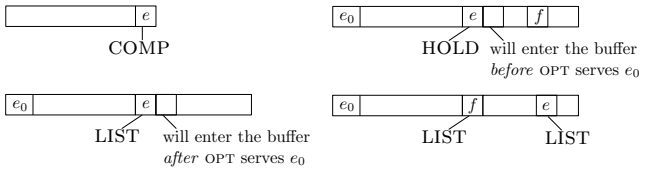

Fig. 1 An illustration of a colour block B and the three possible types of items as based on an arbitrary lazy OPT. Let e_0 be the oldest item in B and let f be the youngest item in B that was in the buffer at the time that OPT began serving B. Top-left: item e is assigned type COMP since e is the youngest item of B. Top-right: item e is assigned type HOLD since e is older than f. Bottom-left: item e is assigned LIST since $e = f$. Bottom-right: item e is assigned LIST since e is younger than f and e is not the youngest item of B. The LIST type indicates that OPT starts serving this colour block before all items have been read into the buffer; the remaining items enter the buffer while the colour block is being served (and while other items of the colour block are being output).

From the definition of the advice, we get the following observation that each colour block of OPT contains 0 or more HOLD items followed by 0 or more LIST items followed by a COMP item.

Observation 2 *The string produced by the concatenation of types of the items in a colour block when ordered from oldest to youngest forms an expression in the language defined by the regular expression*

$\text{HOLD}^*\text{LIST}^*\text{COMP}$.

We can now define the notion of the type of a colour block at time τ. Contrary to an item type that is fixed, the type of a colour block changes over time depending on the youngest item of the colour block currently in the buffer of a given algorithm. At time τ, let e be the youngest item of a colour block B in the buffer. The type of B, denoted by $t_B(\tau)$, is t_e. Colour blocks of type LIST and COMP are ranked based on the age of their items. At time τ, let two colour blocks, B_1 and B_2, be of the same type t. Let r_{B_1} be the oldest item of type t in B_1, and let r_{B_2} be the oldest item of type t in B_2. (Note that r_{B_1} and r_{B_2} may have already been output at time τ.) We say that B_1 is older than B_2 if r_{B_1} is older than r_{B_2} and B_1 is younger than B_2 if r_{B_1} is younger than r_{B_2}.

The Algorithm 2 BIT LAZY (2BL). 2BL is a lazy algorithm that works in the following manner. At a time t, if the buffer is not full, the current request, r_t, is brought into the buffer. If the buffer is full and contains an item with the same colour as the active colour c, then the oldest item of colour c is output. Otherwise, the buffer is full and there are no items with the same colour as the active colour. At this point, the algorithm will perform a colour switch.

In order to choose the next colour to use, we need to define the notion of an advice block.

Definition 6 Let $\sigma_\tau = \{r_1, \ldots, r_{\tau-1}\} \subseteq \sigma$ be the prefix of the input sequence which has entered the buffer until time τ. An *advice block* A is a maximal subsequence of items from σ_τ such that (i) all elements in A have the same colour c, (ii) for $i < j < \ell$, if $r_i, r_\ell \in A$ and $c(r_j) = c$, then $r_j \in A$, (iii) if $r \in A$ and $t_r = \text{COMP}$, then r is the last item in A.

Note that the advice blocks are pairwise disjoint, and for the type sequence $T_{\text{2BL}}(\sigma)$ each advice block corresponds to a different colour block of OPT. That is, an advice block B consists of all elements of a corresponding colour block B_{OPT} of OPT which have been read from the input sequence so far (i.e., which are contained in $\{r_1, \ldots, r_{\tau-1}\}$).

We say that an advice block B is in the buffer at some time τ, if there is an item $e \in B$ that is in the buffer at the beginning of time step τ. The type and the age of an advice block at a given time are defined in the same manner as the type of a colour block at a given time.

At each time step τ, the next colour is chosen based on the advice blocks in the buffer of 2BL, according to the following rules.

1. If there is an advice block B with type COMP, then switch to the colour of the oldest COMP advice block.
2. Else, if there is an advice block B with type LIST and containing at least two items of type LIST, switch to the colour of B.
3. Else, switch to the colour of the advice block B with type LIST that has maximum cardinality. (Ties are broken by choosing the colour of the oldest advice block.)

3.2 The Analysis

The next two lemmas show that the algorithm 2BL is well defined for any request sequence, and it will be able to output all the items.

Lemma 2 *Whenever the buffer of* 2BL *is full, it contains an item of type* LIST *or* COMP.

Proof If an item of some advice block B is of type HOLD, then another item of the advice block B will arrive before OPT starts to serve B_{OPT}.

Assume for the sake of contradiction that, at some time τ, the buffer of 2BL is full of items of type HOLD. By definition, OPT does not serve any of the colour blocks of these items until after time τ. This means that OPT

also has all of these items in its buffer at time τ. This is a contradiction, as OPT has to remove an item at time τ and must be serving some colour block. □

Lemma 3 *At time $n+1$, the buffer of* 2BL *contains only advice blocks of type* COMP.

Proof At time $n+1$, all the items have entered the buffer and, by the definition of the advice, each advice block in the buffer will be terminated by a COMP item. □

For an advice block B, we denote by B_{OPT} the corresponding colour block of OPT. Based on the order of precedence for the colour switches of 2BL, the first priority are advice blocks of type COMP. By the definition of the advice, for such an advice block B, the youngest item of the colour block B_{OPT} is already in the buffer, and all the items of B_{OPT} that have not yet been output can be output on a single colour switch. The second priority are advice blocks of type LIST such that at least two LIST items have entered the buffer (and possibly have been output). The following two lemmas show that there can be at most one such advice block B in the buffer of 2BL, and that the remaining items of B_{OPT} can be output on a single colour switch.

Lemma 4 *At any time, the buffer of* 2BL *contains at most one advice block B of type* LIST *containing at least two items of type* LIST.

Proof By the definition of the advice and 2BL, if, at some time τ, the buffer of 2BL contains an advice block B with two items of type LIST and no COMP item, this indicates that OPT is serving B_{OPT} at time τ. Since OPT can only serve one colour block at a time, there cannot be two such advice blocks at any time τ. □

Lemma 5 *At a time τ', let B be an advice block of type* LIST *containing at least two items of type* LIST. *If* 2BL *switches to the colour of B at time τ',* 2BL *can serve all the items of B_{OPT} that have not yet been output with a single colour switch.*

Proof Let c be the colour of advice block B. By the definition of type LIST and Lemma 4, OPT starts to serve B_{OPT} at some time $\tau < \tau'$ and is still serving B_{OPT} at τ'. By the definition of the advice, all the advice blocks corresponding to the colour blocks that OPT serves prior to τ must have type COMP at time τ and, therefore, by the definition of the algorithm, 2BL must have served those advice blocks by time τ', as otherwise it would not switch to colour c at the τ'.

Hence, at time τ', all the items that are not of colour c in the buffer of 2BL are also in the buffer of OPT. We conclude that the c-buffer of $\text{OPT}_{B_{\text{OPT}}}$ is no larger than the c-buffer of 2BL_B at time τ'. This implies that 2BL_B (and 2BL) reads the input at least as far as $\text{OPT}_{B_{\text{OPT}}}$ and, in particular, completes the colour block B_{OPT}. □

The third priority are advice blocks that are of type LIST with only one LIST item. 2BL will only switch to the colour of such an advice block if its buffer does not contain advice blocks of a higher priority, which implies that the buffer of 2BL only contains advice blocks of the third priority or HOLD advice blocks. The following lemma shows that in this case, after switching to the colour of the largest-cardinality LIST advice block B, either all items of B_{OPT} will be output, or an item of type LIST or COMP of another LIST advice block B' in the buffer will enter the buffer, ensuring a higher priority advice block for the subsequent colour switch.

Lemma 6 *Let $\mathcal{B}_{\text{LIST}}$ be the set of* LIST *advice blocks in the buffer of* 2BL *at a time τ, and assume that the buffer of* 2BL *contains no advice blocks of type* COMP *or with at least two* LIST *items. Then switching to the colour of the largest advice block $B^* \in \mathcal{B}_{\text{LIST}}$ at time τ guarantees that either (i) all the items of B^*_{OPT} will be output before the next colour switch, or (ii) a* LIST *or* COMP *item from another advice block $B' \in \mathcal{B}_{\text{LIST}}$ enters the buffer while* 2BL *is outputting the items of B^*_{OPT}.*

Proof At time τ, OPT starts or continues serving some colour block B_{OPT} (where the corresponding advice block $B \in \mathcal{B}_{\text{LIST}}$) of colour c. All items from the buffer of 2BL which are not of colour c are also in the buffer of OPT at time τ. Hence, if 2BL switches to B at time τ, it will serve the entire colour block B_{OPT}. If 2BL switches to $B^* \neq B$ of colour $c^* \neq c$, then the c^*-buffer of 2BL_{B^*} is at least as large as the c-buffer of $\text{OPT}_{B_{\text{OPT}}}$ as, by definition, $|B^*| \geq |B|$. Hence, a new item e with type LIST or COMP of colour c must arrive before the c^*-buffer is full, otherwise OPT could not serve e together with the rest of the colour block B_{OPT}. Thus, after serving B^*, 2BL will have an advice block with at least two items of type LIST or with an item of type COMP in its buffer. □

From Lemma 5 and Lemma 6 we can observe that every time 2BL has to perform two colour switches to serve a colour block B_{OPT} (containing items of type LIST), there is another colour block that is output by 2BL with a single colour switch. That allows us to prove the following theorem.

Theorem 2 *The algorithm* 2BL *is* 1.5*-competitive.*

Proof Let b^{COMP} be the set of colour blocks of OPT without a LIST item. By the definition of the algorithm and the advice, 2BL will perform $|b^{\text{COMP}}|$ colour switches to serve those colour blocks.

Consider the colour blocks of OPT that contain a LIST item. Let b_1^{LIST} be the set of colour blocks of OPT that 2BL serves with a single colour switch, and let b_2^{LIST} be the set of colour blocks of OPT that 2BL serves with more than a single colour switch. Each colour block B_{OPT} in b_2^{LIST} will be served with two colour switches by 2BL. The first colour switch occurs after the first LIST item of B_{OPT} enters the buffer of 2BL and the second colour switch occurs after the second LIST item of B_{OPT} enters the buffer of 2BL. By Lemma 5, the items of B_{OPT} not yet output will be output by the second colour switch by 2BL. Hence, for 2BL, the colour blocks in b_2^{LIST} are output with $2|b_2^{\text{LIST}}|$ colour switches.

From the definition of the advice, the algorithm and Lemma 6, the b_2^{LIST} colour blocks can only occur when the buffer contains only HOLD advice blocks and LIST advice blocks with at most a single LIST item in the buffer. In such a case, a colour switch is made to the largest advice block B of type LIST. By Lemma 6, after serving B, when all the items of B_{OPT} are not output, either a LIST or a COMP type item e from another LIST advice block B' enters the buffer of 2BL. If t_e is COMP, all the items of B'_{OPT} are in the buffer and 2BL will output all of them on the next colour switch to $c(e)$. If t_e is LIST, then 2BL will output all the items of colour block B'_{OPT} on the next colour switch to $c(e)$ by Lemma 5. Therefore, for every colour block in b_2^{LIST}, there is a colour block that 2BL will serve in its entirety that is in b_1^{LIST} which implies $|b_2^{\text{LIST}}| \leq |b_1^{\text{LIST}}|$.

From the definition of the advice, the cost of OPT is the number of colours blocks which is $|b^{\text{COMP}}| + |b_1^{\text{LIST}}| + |b_2^{\text{LIST}}|$. From this and the fact that $|b_2^{\text{LIST}}| \leq |b_1^{\text{LIST}}|$, we get that $|b_2^{\text{LIST}}| \leq \text{OPT}(\sigma)/2$. Therefore, the number of colour switches of performed by 2BL is $|b^{\text{COMP}}| + |b_1^{\text{LIST}}| + 2|b_2^{\text{LIST}}| \leq \frac{3}{2}\text{OPT}(\sigma)$. □

In the following theorem, we complement the previous result by showing that the upper bound on 2BL is tight.

Theorem 3 *For $k \geq 3$, the competitive ratio of* 2BL *is at least* 1.5.

Proof For a colour c, let c^j denote items of colour c requested j times in a row. For colours c_x, c_y, c_z, let $R(c_x, c_y, c_z) := c_y^{\lfloor \frac{k}{2} \rfloor}, c_y^{\lceil \frac{k}{2} \rceil}, c_z^{\lceil \frac{k}{2} \rceil}, c_x$, i.e., we have k requests of the colour c_y, followed by $\lceil \frac{k}{2} \rceil$ requests of c_z, followed by a single request of c_x. We define an infinite family of request sequences parameterized by $p \geq 1$ as follows. Let

$$\Lambda_p = \Big\langle c_1^{\lceil \frac{k}{2} \rceil}, R(c_1, c_2, c_3), \\ R(c_3, c_4, c_5), \\ \vdots \\ R(c_{2p-1}, c_{2p}, c_{2p+1}) \Big\rangle ,$$

where each c_i is a distinct colour. Note that, for any $\sigma \in \Lambda_p$, the optimal cost is exactly the number of colours, i.e., $\text{OPT}(\sigma) = 2p + 1$. Further, the sequence of colour switches for an optimal algorithm is $\langle c_2, c_1, c_4, c_3, c_6, \ldots, c_{2p}, c_{2p-1}, c_{2p+1} \rangle$. From the construction of the advice sequence, for each odd i the $\lceil \frac{k}{2} \rceil$-th item of colour c_i is the first one of type LIST, and for each even i the $\lfloor \frac{k}{2} \rfloor$-th item of colour c_i is the first one of type LIST.

Initially and after completely serving every item of a colour switch to an odd indexed colour c_i, the buffer of OPT contains two advice blocks, $c_{i+2}^{\lceil \frac{k}{2} \rceil}$ and $c_{i+3}^{\lfloor \frac{k}{2} \rfloor}$. Each of these advice blocks has type LIST and contains one element of type LIST.

Now, we consider how the algorithm 2BL will serve a request sequence $\sigma \in \Lambda_p$. We define a phase of 2BL to begin whenever the buffer contains $c_i^{\lceil \frac{k}{2} \rceil}$ and $c_{i+1}^{\lfloor \frac{k}{2} \rfloor}$, where i is odd. Note that the first phase beings once the first k items enter the buffer.

For each phase, by the definition of the algorithm, 2BL will first serve the $\lceil \frac{k}{2} \rceil$ items of the colour c_i. Then, the buffer is filled with k items of the colour c_{i+1} and 2BL serves c_{i+1}. After removing all the items of colour c_{i+1}, the contents of the buffer of 2BL is $c_{i+2}^{\lceil \frac{k}{2} \rceil}, c_i$ and, if it is not the last phase, $c_{i+3}^{\lfloor \frac{k}{2} \rfloor - 1}$. By definition, 2BL serves c_i which is of type COMP and the buffer contains $c_{i+2}^{\lceil \frac{k}{2} \rceil}$ and, if it is not the last phase, $c_{i+3}^{\lfloor \frac{k}{2} \rfloor}$. The next phase starts. We have shown that during each phase, except of the last one, 2BL performed three colour changes. In the last phase, the algorithm performs one more colour switch to serve the items of colour c_{2p+1}. Overall, 2BL needs $3p + 1$ colour switches. □

4 An Optimal Algorithm with $2 + \lceil \log k \rceil$ Bits of Advice per Request

In this section, we consider the 1.5-competitive algorithm from the previous section and enhance it by giving it access to the order in which OPT serves the LIST colour blocks. This enables the algorithm to serve σ with no more colour switches than OPT. However, at any given time step, there could be $k - 1$ other LIST

advice blocks in the buffer. So, encoding the position of an advice block in an ordered list of LIST advice blocks may require $\lceil \log k \rceil$ bits (which is actually allows one to run the trivial algorithm with advice that indicates the colour switch at each time step). The ordered list of LIST advice blocks is called here the waiting list and the algorithm described in the present section is called 2 BIT LAZY WITH WAITING LIST (2BWL). In the two subsequent sections, we will modify the advice string of the algorithm so as to obtain a $(1+\varepsilon)$-competitive algorithm with a constant number of advice bits per request. More specifically, we will allow for a multiplicative factor of ε more colour switches but generate the item types such that the position of the LIST advice blocks in the waiting list can be encoded with $O(\log \frac{1}{\varepsilon})$ bits.

4.1 The Advice, the Waiting List and the Algorithm

The Advice. Given a request sequence σ and a fixed lazy OPT, each item $e \in \sigma$ is assigned a type $t_e \in \{\text{HOLD}, \text{LIST}, \text{COMP}\}$ as defined in the previous section, i.e., $T_{2\text{BWL}}(\sigma) = T_{2\text{BL}}(\sigma)$.

Consider the advice blocks of σ as described by the advice sequence $T_{2\text{BWL}}(\sigma)$. Each advice block B with an item of type LIST is assigned a value u_B that is an integer in $\{0, \ldots, k-1\}$. The value u_B is defined in the following manner. Let e be the oldest LIST item in some advice block B and let τ be the time that e enters the buffer of 2BWL. Consider all the colour blocks of OPT that correspond to the advice blocks of type LIST present in the buffer of 2BWL at time τ, and order those colour blocks according to the order in which OPT outputs them. The value of u_B is the index of B_{OPT} in such an ordering, where the initial index of the ordering is 0, i.e., the colour block with the index 0 is the first one to be output by OPT.

For the sake of analysis, we define another sequence, $U_{2\text{BWL}}(\sigma)$, to be the sequence of the values of u_B ordered, from oldest to youngest, by the age of the oldest LIST item in each advice block with an item of type LIST in σ.

The Waiting List. The algorithm maintains a *waiting list* of advice blocks of type LIST. This list contains only advice blocks that have, at some time, type LIST. At the beginning, the waiting list is empty. Whenever an advice block B of type LIST appears (i.e., the first item of type LIST for some advice block is read from the input), it is inserted into the waiting list. The *initial* position of the advice block on the waiting list, i.e., the position where the advice block is inserted, is defined by the value u_B of the advice block. A value of 0 denotes the head of the list. Note that when a new advice block is inserted into or removed from the waiting list, the position of the other advice blocks on the list can change and, whenever an advice block on the waiting list is output, that advice block is removed from the waiting list.

The Algorithm 2 BIT LAZY WITH WAITING LIST (2BWL). 2BWL is a lazy algorithm that runs just as 2BL. The difference is the manner in which 2BWL will choose the next colour for each colour switch. If there is an advice block B with type COMP in the buffer of 2BWL, then 2BWL will switch to the colour of B. Otherwise, if there is no advice block with type COMP, 2BWL switches to the colour of the advice block B at the front of the waiting list and removes B from the waiting list.

As can be seen from the proof of Theorem 2, 2BL only uses more colour switches than OPT if it chooses the wrong colour block of type LIST to serve. The advice of 2BWL allows it to avoid this error. Hence, 2BWL is optimal.

5 A $(1+\varepsilon)$-Competitive Algorithm with $2 + \lceil \log k \rceil$ Bits of Advice per Request

In this section, we modify the 2BWL algorithm and the advice from the previous section and present an algorithm that is $(1+\varepsilon)$-competitive and uses $2 + \lceil \log k \rceil$ bits of advice per request. This algorithm has a worse competitive ratio than 2BWL, but, for a cost of an additional multiplicative factor of ε colour switches, the number of LIST advice blocks that are placed in the waiting list is significantly reduced. In the next section, we will modify the algorithm presented in this section such that the position of the advice blocks in the waiting list will be encoded in a constant number of bits independent of k.

5.1 The Advice and the Algorithm

A New Item Type. Given a request sequence σ and a fixed lazy OPT, each item $e \in \sigma$ is a assigned a type $t_e \in \{\text{HOLD}, \text{LIST}, \text{READY}, \text{COMP}\}$, where HOLD, LIST and COMP are defined as in Section 3. READY is a new item type that will be used along with COMP to split some advice blocks with an item of type LIST into two separate advice blocks, such that the algorithm can force some items to be removed earlier from the buffer, or to delay the time when an advice block with an item of type LIST will be output (see below for the details). Note that the four item types can still be encoded in two bits.

We extend the notion of an advice block for items of type READY. The definition remains the same except that the string produced by the concatenation of the types of the items in an advice block is in the language HOLD*(LIST* ∪ READY*)COMP?. With the addition of the type READY as detailed below, the advice blocks are still disjoint, but there can be more advice blocks than colour blocks of OPT. Specifically, there could be two advice blocks that are the prefix and the suffix of a colour block of OPT.

An advice block B has type READY if it contains an item of type READY and no item of type COMP. At a time τ, for two advice blocks of type READY in the buffer, we say that B_1 is older than B_2 if the oldest READY item in B_1 is older than the oldest READY item in B_2.

The Algorithm 2 BIT LAZY WITH REDUCED WAITING LIST (2BRWL). 2BRWL is a lazy algorithm that runs just as 2BL and 2BWL. The difference is the manner in which 2BRWL chooses the next colour for a colour switch. At a time step τ, the next colour is chosen based on the advice blocks in the buffer of 2BRWL, according to the following rules.

1. If there is an advice block B with type COMP, then switch to the colour of the oldest COMP advice block,
2. else, if there is an advice block with type READY, then switch to the colour of the oldest READY advice block,
3. else, if there is an advice block with type LIST, switch to the advice block B at the front of the waiting list and remove B from the waiting list.

5.1.1 Building the Advice Sequences

The advice sequence containing the types of the items, $T_{2\text{BRWL}}(\sigma)$, and the advice sequence containing the waiting list positions, $U_{2\text{BRWL}}(\sigma)$, are constructed offline and based on an arbitrary *optimal* lazy solution OPT for the instance σ. The idea of the construction is as follows. We initially assign each input item e type $t_e \in \{\text{HOLD}, \text{LIST}, \text{COMP}\}$ as it is assigned for 2BL. That is, initially, $T_{2\text{BRWL}}(\sigma) = T_{2\text{BL}}(\sigma)$ and $U_{2\text{BRWL}}(\sigma) = U_{2\text{BWL}}(\sigma)$. Then, 2BRWL is simulated on σ with the advice sequences $T_{2\text{BRWL}}(\sigma)$ and $U_{2\text{BRWL}}(\sigma)$. Given some constant C (defined later), whenever the waiting list of 2BRWL contains at least C advice blocks of sizes that differ by a factor of at most 2, these advice blocks will be removed from the waiting list at a cost of only two additional colour switches. This will be accomplished by replacing LIST type items with type READY and type COMP in $T_{2\text{BRWL}}(\sigma)$. This modification introduces items of type READY and increases the number of advice blocks as defined by σ and $T_{2\text{BRWL}}(\sigma)$ (and updates $U_{2\text{BRWL}}(\sigma)$ as advice blocks with READY items are not put into the waiting list). The number of colour switches performed by 2BRWL increases as the number of advice blocks defined for σ by $T_{2\text{BRWL}}(\sigma)$ increases, but the number of advice blocks of similar size in the waiting list at any given time will be bounded by a constant and this will allow us, in the next section, with one more alteration, to encode the position of the advice blocks in the waiting list with a constant number of advice bits.

In order to formally define the procedure of generating $T_{2\text{BRWL}}(\sigma)$ and $U_{2\text{BRWL}}(\sigma)$, we will first define three procedures that are used to modify the advice data. The input to the procedures is a complete advice block B, i.e., the advice block defined with respect to the entire request sequence σ and $T_{2\text{BRWL}}(\sigma)$, containing at least one item of type LIST.

The first procedure removes an advice block B from the waiting list and updates the values of u'_B for the advice blocks B' inserted behind B in the waiting list.

Procedure Remove(B)

- Remove u_B from $U_{2\text{BRWL}}(\sigma)$.
- For each advice block B' that was inserted into the waiting list, at a position behind B, after B was inserted but prior to B being output, decrease $u_{B'}$ by one.

The next procedure will split an advice block B into two advice blocks, such that both will end with an item of type COMP, by reassigning the oldest LIST item to type COMP. Additionally, the other items in B with LIST type are changed to READY (see Figure 2). This ensures that neither of the new advice blocks will be placed on the waiting list. The first advice block is called an *early block* and the second advice block is called a *late block*.

Procedure Split(B)

- Run Remove(B).
- Reassign type COMP to the first item of B which has type LIST assigned.
- Reassign type READY to the remaining items of B which have type LIST assigned.

Let B be an advice block that is processed by the Split procedure that produces an early block B' and a late block B''. The colour block of OPT that corresponds to B, B_{OPT}, is the same colour block that corresponds to both B' and B'', i.e. $B_{\text{OPT}} = B'_{\text{OPT}} = B''_{\text{OPT}}$.

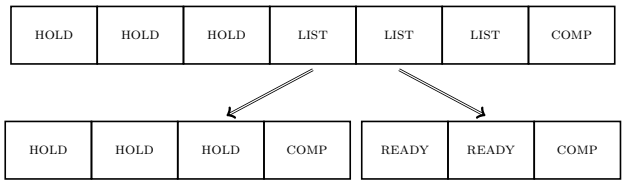

Fig. 2 An example of the procedure Split(B) as applied to an advice block B (top) of items with types HOLD, LIST and COMP. B is split into two new advice blocks (bottom). The early block (bottom left) is made up of the HOLD items and the first LIST item of B which is changed to COMP. The late block (bottom right) has the rest of the items of B with all the LIST items changed to READY.

The procedure Split(B) makes 2BRWL output the items from the early block of B before they are output by OPT, generating free space in the buffer of 2BRWL. This allows some advice blocks to stay in the buffer until the next item of the advice block is read from the input, and only then to be output by 2BRWL. These are the advice blocks for which we run procedure Postpone(B) (see Figure 3).

Procedure Postpone(B)

- Run Remove(B).
- Reassign type HOLD to the first item of B which has type LIST assigned.
- Reassign type READY to the remaining items of B which have type LIST assigned.

Advice blocks processed in this way are called *postponed blocks*. Note that no additional advice blocks are produced by the procedure Postpone. For an advice block B that is processed by the Postpone procedure, B remains the only advice block that corresponds to B_{OPT}.

An advice block B that is processed using the Split(B) or Postpone(B) procedure will never be inserted into the waiting list of 2BRWL. This is the reason why the procedure Remove(B) is run at the beginning of both procedures Split(B) and Postpone(B).

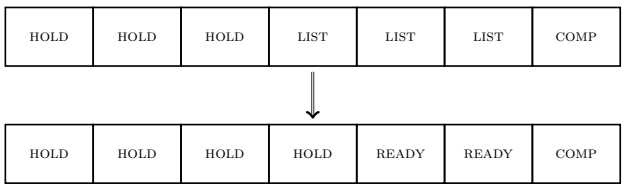

Fig. 3 An example of the procedure Postpone(B) as applied to an advice block B (top) of items with types HOLD, LIST and COMP. The first LIST item of B is changed to HOLD and the remaining LIST items are changed to READY as shown in the postponed block (bottom).

The goal of our algorithm is to ensure that there are not too many advice blocks of similar size in the waiting list. To formalize the notion of similar size, we define the *class* of a LIST advice block. In the definition, $h(B)$ is the number of items of B in the buffer of OPT when OPT starts serving B_{OPT}. That is, we count the first LIST item and all the HOLD items of B. Note that for an early block B' resulting from Split(B) the number of items in B' is $h(B)$.

Definition 7 The *class* of an advice block B of type LIST is the number $\lfloor \log h(B) \rfloor$, where $h(B)$ is the number of items of type HOLD in B increased by one.

From the definition of a class of a LIST advice block B, we have that, for a class i, $h(B) \in \{2^i, \ldots, 2^{i+1}-1\}$. Therefore, for a buffer of size k, there are $\lfloor \log k \rfloor + 1$ possible classes, i.e., the classes correspond to the integers from 0 to $\lfloor \log k \rfloor$.

We are now ready to formally define $T_{2\text{BRWL}}(\sigma)$ and $U_{2\text{BRWL}}(\sigma)$. Initially, $T_{2\text{BRWL}}(\sigma) = T_{2\text{BL}}(\sigma)$ and $U_{2\text{BRWL}}(\sigma) = U_{2\text{BWL}}(\sigma)$. Simulate 2BRWL on σ with $T_{2\text{BRWL}}(\sigma)$ and $U_{2\text{BRWL}}(\sigma)$. Let n_i^τ be the number of advice blocks of class i in the waiting list at time τ. Let (τ^*, i^*) be an arbitrary pair such that

$$(\tau^*, i^*) \in \operatorname*{arg\,max}_{\tau \in \{1,n\}, i \in \{0, \lfloor \log k \rfloor\}} n_i^\tau \ .$$

If $n_{i^*}^{\tau^*} \geq C$, let B_1 and B_2 be the two last advice blocks of class i^* in the waiting list at time τ^*. We perform the following two operations.

- Run Split(B_1) and Split(B_2).
- For all remaining advice blocks B' of class i^* in the waiting list at time τ^*, run Postpone(B').

By running these procedures, the advice sequences are updated so that 2BRWL will not insert any of these $n_{i^*}^{\tau^*}$ advice blocks into the waiting list. This process is repeated using the updated advice sequences until, for all $1 \leq \tau \leq n$ and $0 \leq i \leq \lfloor \log k \rfloor$, we get $n_i^\tau < C$.

5.2 The Analysis

For the analysis, unless explicitly stated, we will always consider the advice blocks as defined with respect to the entire request sequence, σ, and their types, $T_{2\text{BRWL}}(\sigma)$. The definition of 2BRWL and the two following lemmas show that 2BRWL is well defined for any request sequence. The proof of Lemma 7 follows that of Lemma 2 for the items of type HOLD in both $T_{2\text{BL}}(\sigma)$ and $T_{2\text{BRWL}}(\sigma)$. Additionally, we have to consider the items of type HOLD in $T_{2\text{BRWL}}(\sigma)$ created by the procedure Postpone(B), i.e., items that are type LIST in $T_{2\text{BL}}(\sigma)$.

Lemma 7 *Whenever* 2BRWL *has a full buffer, it contains a* LIST*,* READY *or* COMP *type item.*

Proof Assume for the sake of contradiction that at time τ the buffer of 2BRWL is full of items of type HOLD.

Due to Lemma 2 it cannot happen that all these elements had type HOLD before we started running the Postpone procedure.

Now consider the case that the buffer contains some HOLD items that were originally of type LIST (in $T_{2\text{BL}}(\sigma)$), then the procedure Postpone has been run on the advice blocks containing these items, reassigning the type of the items from LIST to HOLD and creating a postponed block. If, at time τ, OPT has already finished outputting the items of any postponed block contained in the buffer of 2BRWL, the item of type COMP from the postponed block would have been already read from the input and it would be in the buffer of 2BRWL. This contradicts the fact that the buffer is full of HOLD items. Hence, at time τ, OPT is outputting the items of one of the postponed blocks. Let B be the postponed block being output. Note that all $k-h(B)$ items in the buffer of 2BRWL of type HOLD, which are not in B, are still in the buffer of OPT. From the definition of $T_{2\text{BRWL}}(\sigma)$, B becomes postponed after two other advice blocks from the same class, B_1 and B_2, are split. B_1 and B_2 are two advice blocks output by OPT later than B, that are on the waiting list at the same time as B. Since B, B_1 and B_2 are in the same class, $h(B), h(B_1)$ and $h(B_2) \in \{2^i, \ldots, 2^{i+1}-1\}$ and, hence, the two early blocks obtained from B_1 and B_2 are of size greater than $h(B)/2$ each. At time τ, B_1 and B_2 are still in the buffer of OPT, since, by definition, they are output by OPT after B. 2BRWL must have output the two early blocks B_1 and B_2 at time τ, otherwise 2BRWL would have items not of type HOLD in its buffer. This is a contradiction as OPT does not have enough buffer space to keep at least $k - h(B) + 2(h(B)/2 + 1/2) = k + 1$ items in the buffer. □

The proof of the following lemma is analogous to that of Lemma 3.

Lemma 8 *At time* $n+1$*, the buffer of* 2BRWL *contains only advice blocks of type* COMP*.*

Lemma 7 leaves open the possibility that 2BRWL may be able to serve only *part* of an advice block. This could only happen if all the items of the entire advice block do not enter the buffer while 2BRWL is serving its colour. In such a case, 2BRWL would have to return to this colour several times, and its cost would be higher than the number of advice blocks given by the advice sequence $T_{2\text{BRWL}}(\sigma)$. We will show that this does not happen, and 2BRWL always outputs the complete advice blocks as defined for the entire request sequence σ and $T_{2\text{BRWL}}(\sigma)$. Then, we will bound from above the competitive ratio of 2BRWL.

First, we present the following technical lemma on the item types contained in an advice block of $T_{2\text{BRWL}}(\sigma)$ that shows that each advice block is composed of 0 or more HOLD items followed by 0 or more LIST or READY items followed by a COMP item.

Lemma 9 *In the advice sequence* $T_{2\text{BRWL}}(\sigma)$*, the string produced by the concatenation of types of the items in an advice block when ordered from oldest to youngest forms an expression in the language defined by the regular expression*

$$\text{HOLD}^*(\text{LIST}^* \cup \text{READY}^*)\text{COMP} \ .$$

Proof Initially, in the definition of $T_{2\text{BRWL}}(\sigma)$, $T_{2\text{BRWL}}(\sigma) = T_{2\text{BL}}(\sigma)$. So, from Observation 2, initially, all the advice blocks in $T_{2\text{BRWL}}(\sigma)$ are of the form HOLD*LIST*COMP. In the process of generating the final $T_{2\text{BRWL}}(\sigma)$, only the advice blocks of the required form are generated through the procedures Split and Postpone. Let B be an advice block of the form HOLD*LIST*COMP. Applying the procedure Split(B) generates an early block of the form HOLD*COMP and a late block of the form READY*COMP (see Figure 2). Applying the procedure Postpone(B) generates an advice block of the form HOLD*READY*COMP (see Figure 3). □

Now, we consider all the types of advice blocks which 2BRWL starts outputting before they become complete, and we show that OPT cannot keep all the items of such advice blocks longer in the buffer.

Lemma 10 *When the algorithm* 2BRWL *starts outputting an advice block, the complete advice block will be output with no additional colour switches.*

Proof The only advice blocks which 2BRWL will start outputting before reading all the items of the advice block from the input are READY blocks and LIST blocks. All the READY blocks are either postponed blocks from the Postpone procedure or late blocks from the Split procedure.

First, consider the postponed blocks of type READY. From the construction of $T_{2\text{BRWL}}(\sigma)$, we know that 2BRWL starts outputting a postponed block B later than OPT. That is, OPT begins outputting B_{OPT} strictly before reading the item f that follows the first item e of type LIST. The advice for e gets modified to HOLD in the Postpone procedure and, therefore, 2BRWL only begins outputting B *after* reading the item f of type

READY into the buffer. Also, we know that the oldest item of B_{OPT} is in the buffer of 2BRWL at that time. Hence, from Lemma 1, 2BRWL can finish outputting all the items of the block with a single colour switch.

Now, consider the late blocks of type READY. These blocks were produced by the Split procedure. Let τ' be the time when 2BRWL starts outputting a late READY block B. From the definition of the Split procedure, the first READY item of a late block B is the second LIST item of the original advice block. Let τ be the time when OPT starts outputting B_{OPT}. By the definition of the advice we have $\tau < \tau'$ and, since the COMP item of B is not in the buffer at time τ', OPT is still outputting the items of B_{OPT} at time τ'. The buffer of 2BRWL at time τ' only contains LIST advice blocks and HOLD advice blocks that contain items that are served by OPT after time τ'. The early block that corresponds to B_{OPT} has been output by 2BRWL before time τ'. Therefore, $\tau' \geq \tau + i$, where i is the number of items in the early block that corresponds to B_{OPT}. By Lemma 1, 2BRWL can output all the items of B without a colour change.

Finally, consider the advice blocks of type LIST. Let τ' be the time when 2BRWL starts outputting an advice block B of type LIST. By the definition of the advice, the advice block B was not processed by the procedures Split or Postpone and is, therefore, an advice block as defined for σ, using $T_{\text{2BL}}(\sigma)$. Also, the oldest item of B_{OPT} is in the buffer of 2BRWL at the time that 2BRWL begins outputting B. By showing that OPT starts outputting B_{OPT} at some time $\tau \leq \tau'$, the claim follows from Lemma 1.

For the sake of contradiction, assume that OPT starts outputting B_{OPT} at a time $\tau > \tau'$. As B has type LIST at time τ', we get that at time τ' there are no advice blocks of type READY or COMP in the buffer of 2BRWL. All the advice blocks in the buffer of 2BRWL at time τ' are

(1) original advice blocks of type HOLD as defined for σ, using $T_{\text{2BL}}(\sigma)$,
(2) original advice blocks of type LIST as defined for σ, using $T_{\text{2BL}}(\sigma)$,
(3) postponed blocks of type HOLD.

By the definition of the advice, OPT still has the items of the advice blocks of (1) in the buffer at time τ'. The advice blocks of (2) are in the waiting list behind B. This implies that OPT still has the items of the advice blocks of (2) in its buffer at time τ'. The advice blocks of (3) are postponed blocks and because of the reassignment of the type in the Postpone procedure, OPT can be in the process of outputting the first of them at time τ'. In this case, OPT would have the items of two early blocks of size $h(B_1)$ and $h(B_2)$ in its buffer, where B_1 and B_2 are in the same class as B. These two early blocks have a total size greater than $h(B)$, since $h(B), h(B_1)$ and $h(B_2) \in \{2^i, \ldots, 2^{i+1} - 1\}$, and the early blocks have already been output by 2BRWL at time τ'. This is a contradiction, as OPT does not have enough buffer space to keep at least $k - h(B) + 2(h(B)/2 + 1/2) = k + 1$ items in the buffer at time τ'. □

In the remaining part of this section, we will bound from above the competitive ratio of 2BRWL. For this, we will need a lower bound on the cost of OPT, as well as an upper bound on the cost of 2BRWL. From the construction of the advice sequence, we have that the number of advice blocks defined for σ by $T_{\text{2BRWL}}(\sigma)$ is at least the number of advice blocks defined for σ by $T_{\text{2BL}}(\sigma)$. We get the following two observations.

Observation 3 *The cost of* 2BRWL *is at most the number of advice blocks defined with respect to* σ *and* $T_{\text{2BRWL}}(\sigma)$.

Observation 4 *The cost of* OPT *is at least the number of non-late blocks defined with respect to* σ *and* $T_{\text{2BRWL}}(\sigma)$.

Therefore, to bound the competitive ratio, it is enough to bound the number of late blocks defined with respect to σ and $T_{\text{2BRWL}}(\sigma)$, as compared to the number of advice blocks of other types. Let $\text{LATE}_{\text{2BRWL}}$ denote the set of late blocks created by generating $T_{\text{2BRWL}}(\sigma)$ and $U_{\text{2BRWL}}(\sigma)$.

Lemma 11 *Let* POSTPONED *and* EARLY *denote the sets of postponed and early blocks created when constructing* $T_{\text{2BRWL}}(\sigma)$. *Then,* $|\text{LATE}_{\text{2BRWL}}| \leq \frac{2}{C}(|\text{POSTPONED}| + |\text{EARLY}|) \leq \frac{2}{C}(\text{OPT}(\sigma))$.

Proof Advice blocks from the set $\text{LATE}_{\text{2BRWL}}$ are a result of the Split procedure. The Split procedure is run on two advice blocks out of a set of at least C advice blocks. This procedure produces a total of two early blocks and two late blocks. The remaining advice blocks of which there are at least $C - 2$ become postponed blocks. The first inequality follows from the fact there are at least C early and postponed blocks for every two late blocks, and the second inequality follows from Observation 4. □

By setting $C = \lceil 2/\varepsilon \rceil$ and applying Lemma 11, we get the following theorem.

Theorem 4 *For any* $\varepsilon > 0$ *and* $C = \lceil 2/\varepsilon \rceil$, 2BRWL *is a* $(1+\varepsilon)$*-competitive algorithm for the reordering buffer management problem.*

6 A $(1+\varepsilon)$-Competitive Algorithm with $O(\log(1/\varepsilon))$ Bits of Advice per Request

In this section, we modify the 2BRWL algorithm and the advice in such a way so as to be able to encode the position of the LIST blocks in the waiting list, using only a constant number of bits of advice per request. The resulting algorithm is the main result of the paper.

The idea of the final modification is the following. Let B be an advice block containing items of type LIST, and let τ be the time when the first LIST item of B enters the buffer of the algorithm. It is at time τ that the algorithm needs to know the value of u_B, and there are at least $h(B)$ items of B in the buffer at this time. Recall that for an advice block B with an item of type LIST, $h(B)$ is defined to be the number of HOLD items of B plus one. This corresponds to the number of items of B in the buffer at time τ. Let D be a constant which will be defined later. If $u_B < D^{h(B)}$, then u_B can be encoded in base D with at most $h(B)$ digits. These digits can be encoded in the advice, each digit for one of the first $h(B)$ items of the advice block B. If $u_B \geq D^{h(B)}$, then u_B cannot be encoded in base D with $h(B)$ digits. In this case, the advice block B will be removed from the waiting list by splitting B into an early and late block, using the Split procedure defined previously. The idea is that the latter case will not happen too often, as the number of LIST advice blocks has been significantly reduced going from 2BWL to 2BRWL.

6.1 The Advice and the Algorithm

The Advice. Given a request sequence σ and a fixed lazy OPT, each item $e \in \sigma$ is assigned a type $t_e \in \{\text{HOLD}, \text{LIST}, \text{READY}, \text{COMP}\}$, where HOLD, LIST, READY and COMP are defined as in Section 5.1.

In addition, each item e will have a value $v_e \in \{0, \ldots, D-1\}$. For each advice block B with items of type LIST, the advice for the first $h(B)$ items of B contains a digit of u_B in base D, using $\lceil \log D \rceil$ bits. For all other items, v_e is unused and set to 0.

The Algorithm CONSTANT ADVICE WITH WAITING LIST (CAWL). CAWL is defined in essentially the same manner as the algorithm 2BRWL. The only difference is the manner in which CAWL determines the position in the waiting list for the LIST advice blocks. For an advice block B of type LIST, when the first LIST item enters the buffer it is placed in the waiting list. The position in which to insert B is determined by the v_e values of the first $h(B)$ items of B. Each v_e represents a digit of a base D number that is the position of B in the waiting list.

6.1.1 Building the Final Advice Sequences

We construct the advice sequence containing the types of the items, $T_{\text{CAWL}}(\sigma)$, and the advice sequence containing the waiting list positions, $U_{\text{CAWL}}(\sigma)$, in an offline manner, as follows. Initially, $T_{\text{CAWL}}(\sigma) = T_{\text{2BRWL}}(\sigma)$, $U_{\text{CAWL}}(\sigma) = U_{\text{2BRWL}}(\sigma)$ and $\tau = 0$. In a simulation of CAWL using the advice sequences $T_{\text{CAWL}}(\sigma)$ and $U_{\text{CAWL}}(\sigma)$, let an advice block B be the first advice block to change to type LIST after time τ. If $u_B < D^{h(B)}$, then set the values of v_e appropriately for each of the first $h(B)$ items of B. Otherwise, $u_B \geq D^{h(B)}$. In this case, run Split(B). This creates an early block and a late block, and removes B from the waiting list. Set τ to the time step when the first LIST item of B entered the buffer in the current simulation and repeat this procedure, using the updated advice sequences of $T_{\text{CAWL}}(\sigma)$ and $U_{\text{CAWL}}(\sigma)$.

6.2 The Analysis

For the analysis, unless explicitly stated, we will always consider the advice blocks as defined with respect to the entire request sequence, σ, and their types, $T_{\text{CAWL}}(\sigma)$. CAWL is defined essentially as 2BRWL, the only difference in the algorithm is the encoding of the positions of LIST advice blocks in the waiting list. The difference in the advice sequence is that some of the advice blocks with an item of type LIST as defined for σ by $T_{\text{2BRWL}}(\sigma)$ have been split into early and late blocks by the Split procedure when defining $T_{\text{CAWL}}(\sigma)$. Lemmas 7 to 10 from Section 5.2 hold for CAWL as they hold for 2BRWL because in the proofs we never used the fact that the Split procedure is only performed under the special circumstances of 2BRWL.

As with 2BRWL, in order to bound from above the competitive ratio, we need to bound from above the number of late blocks of $T_{\text{2BRWL}}(\sigma)$ as compared to the number of advice blocks of other types. Lemma 11 bounds from above the number of late blocks created by generating $T_{\text{2BRWL}}(\sigma)$ and $U_{\text{2BRWL}}(\sigma)$ which are initially equal to $T_{\text{CAWL}}(\sigma)$ and $U_{\text{CAWL}}(\sigma)$. Let $\text{LATE}_{\text{CAWL}}$ denote the set of late blocks created by generating $T_{\text{CAWL}}(\sigma)$ and $U_{\text{CAWL}}(\sigma)$ that are not in the set $\text{LATE}_{\text{2BRWL}}$.

Lemma 12 *Let* LISTED *denote the set of advice blocks which contain items of type* LIST *as defined with respect to* σ *and* $T_{\text{2BRWL}}(\sigma)$. *Then,* $|\text{LATE}_{\text{CAWL}}| \leq \frac{C}{D-1}|\text{LISTED}|$.

Proof Let $B' \in \text{LATE}_{\text{CAWL}}$ be a late block produced by applying Split(B) to some LIST advice block B when generating $T_{\text{CAWL}}(\sigma)$ and $U_{\text{CAWL}}(\sigma)$. This only occurs in the case when $u_B \geq D^{h(B)} \geq D^{2^i}$, where i is the class of B.

Let $\mathcal{B}(B)$ be the set of advice blocks in the waiting list at a position before u_B when the first LIST item of B enters the buffer. Therefore, $|\mathcal{B}(B)| = u_B$ and each advice block from the set $\mathcal{B}(B)$ belongs to the set LISTED.

Let B_0 be an advice block in LISTED, and let τ be the time step when the first item of B_0 was output and B_0 was removed from the waiting list. For any advice block $B' \in \text{LATE}_{\text{CAWL}}$ such that $B_0 \in \mathcal{B}(B)$, the advice block B must have been in the waiting list at time τ. From the construction of $T_{2\text{BRWL}}(\sigma)$ and $U_{2\text{BRWL}}(\sigma)$, the number of advice blocks of class i which are in the waiting list at time τ is less than C. So, $|B \text{ of class } i : B_0 \in \mathcal{B}(B)| < C$.

Now, for each advice block $B' \in \text{LATE}_{\text{CAWL}}$, we charge a cost of $\frac{1}{u_B} \leq \frac{1}{D^{2^i}}$ to the first u_B advice blocks from the set $\mathcal{B}(B)$. The total cost charged to all advice blocks in LISTED equals $|\text{LATE}_{\text{CAWL}}|$. Each advice block in LISTED is charged a total cost less than

$$\sum_{i=0}^{\lfloor \log k \rfloor} \frac{C}{D^{2^i}} \leq \frac{C}{D-1} \ .$$

Therefore, $|\text{LISTED}| \cdot \frac{C}{D-1} \geq |\text{LATE}_{\text{CAWL}}|$. □

Combining Lemma 11 and Lemma 12, and setting $C = \lceil 2/\varepsilon \rceil$ and $D = \lceil C/\varepsilon \rceil + 1$ gives us the main upper bound.

Theorem 5 *For any $\varepsilon > 0$, $C = \lceil 2/\varepsilon \rceil$ and $D = \lceil C/\varepsilon \rceil + 1$,* CAWL *is a $(1+\varepsilon)$-competitive algorithm for the reordering buffer management problem, using $O(\log(1/\varepsilon))$ bits of advice per input item.*

Proof Let $P = |\text{POSTPONED}|$, $E = |\text{EARLY}|$ and $L = |\text{LISTED}|$. From Lemma 11 and Lemma 12, we get that

$$\begin{aligned} &|\text{LATE}_{\text{2BRWL}}| + |\text{LATE}_{\text{CAWL}}| \\ &\quad \leq \frac{2}{C}(P+E) + \frac{C}{D-1}L \\ &\quad \leq \max\left\{\frac{2}{C}, \frac{C}{D-1}\right\} \cdot (P+E+L) \\ &\quad \leq \max\left\{\frac{2}{C}, \frac{C}{D-1}\right\} \cdot \text{OPT}(\sigma) \\ &\quad = \varepsilon \text{OPT}(\sigma) \ , \end{aligned}$$

where the third inequality follows from Observation 4 and the fact that the sets POSTPONED, EARLY, and LISTED are disjoint. That is, $\text{OPT}(\sigma)$ can be bounded from below by the non-late blocks defined with respect to σ and $T_{2\text{BRWL}}(\sigma)$. These non-late blocks include the advice blocks from the disjoint sets POSTPONED, EARLY and LISTED.

Overall, we have that $\text{ALG}(\sigma) \leq (1+\varepsilon)\text{OPT}(\sigma)$.

The number of advice bits per input item e is 2 bits to encode the item type (t_e) and $\lceil \log D \rceil$ bits to encode in binary a digit in base D, encoding the position of an advice block in the waiting list (v_e). In total, that is $2 + \lceil \log D \rceil = O(\log(1/\varepsilon))$ bits of advice per item. □

Notice that the algorithm from Theorem 5 is strictly competitive, i.e., the additive constant in the definition of the competitive ratio is 0.

7 A Lower Bound on the Advice Required for a Competitive Ratio Close to 1

In this section, we show that $\Omega(\log k)$ bits of advice per request are needed for a deterministic algorithm to be 1-competitive. More specifically, a deterministic algorithm ALG that uses $o(\log k)$ bits of advice per request cannot have a competitive ratio better than $1 + 1/(2k) - 1/(2k \cdot k!)$.

We first present an infinite family Ψ^k of request sequences, parameterized by an integer $k \geq 2$, where every request sequence $\sigma^k \in \Psi^k$ will consist of some p phases. Then, we define a class of *tidy* algorithms for serving requests from Ψ^k, and show that any online algorithm with a buffer of size k can be converted to an online tidy algorithm without increasing the cost of processing any $\sigma^k \in \Psi^k$. Next, we present a subclass of tidy algorithms called *predetermined tidy* algorithms and show that for any a tidy algorithm $\text{ALG}^k_{\text{TIDY}}$ using a buffer of size k there is a predetermined tidy algorithm $\text{ALG}^k_{\text{PTA}}$ using a buffer of the same size, which serves optimally at least as many phases of any sequence $\sigma^k \in \Psi^k$ as $\text{ALG}^k_{\text{TIDY}}$. Finally, we are able to reduce the q-SGKH problem (see Section 2.2) to the reordering buffer management problem with a buffer of size k, using a predetermined tidy algorithm and the family of request sequences Ψ^k. The lower bound follows from this reduction and Theorem 1 from [9].

Throughout this section, we assume without loss of generality that the algorithms considered are lazy. Also, the buffer size k is assumed to be at least 2.

7.1 The Family of Request Sequences Ψ^k

We first present an infinite family $\Psi^k = \left\{\sigma^k_{p,\pi}\right\}$ of request sequences, parameterized by an integer $k \geq 2$. Every request sequence $\sigma^k_{p,\pi} \in \Psi^k$ is parameterized by a positive even integer p and a family of p permutations of a k-element set $\pi = \{\pi_1, \pi_2, \ldots, \pi_p\}$, where each permutation π_i with an even index i is an identity permutation. The request sequence will be composed of $p+1$ phases, numbered from 0 to p. Let $c_1, \ldots, c_{pk}$ be

distinct colours. The request sequence $\sigma = \sigma^k_{p,\pi} \in \Psi^k$ consists of the requests of the following colours, where the i-th line corresponds to the phase number $i-1$.

$$\begin{aligned}\mathcal{C}_\sigma = \big\langle & c_1, c_2, \ldots, c_k, \\ & \pi_1(c_1), c_{k+1}, \pi_1(c_2), c_{k+2}, \ldots, \pi_1(c_k), c_{2k}, \\ & \pi_2(c_{k+1}), c_{2k+1}, \ldots, \pi_2(c_{2k}), c_{3k}, \\ & \vdots \\ & \pi_{p-1}(c_{(p-2)k+1}), c_{(p-1)k+1}, \ldots, \pi_{p-1}(c_{(p-1)k}), c_{pk}, \\ & \pi_p(c_{(p-1)k+1}), \pi_p(c_{(p-1)k+2}), \ldots, \pi_p(c_{pk}) \big\rangle \ ,\end{aligned}$$

Note that $|\sigma| = |\mathcal{C}_\sigma| = 2pk$. As a slight abuse of notation, for a colour c, when c clearly is a colour at a specific index j in $\mathcal{C}_\sigma$, we will use $r(c)$ to indicate r_j in σ.

A colour is *new* on its first occurrence in $\mathcal{C}_\sigma$. For a phase $i \le p$, let $\mathcal{K}_i = \langle c_{(i-1)k+1}, \ldots, c_{ik} \rangle$ be a sequence the k colours which appear for the first time in phase $i-1$. Each permutation π_i is a permutation of $\mathcal{K}_i$.

The request sequence σ consists of $p+1$ phases. The initial phase, phase 0, consists of k requests, each to a distinct colour from $\mathcal{K}_1$. Each i-th phase, where $1 \le i \le p-1$, alternates the k items of $\mathcal{K}_i$ with k new items of $\mathcal{K}_{i+1}$. For an odd phase i, the k items of $\mathcal{K}_i$ are ordered according some permutation π_i. For an even phase i, the k items of $\mathcal{K}_i$ are ordered according to the identity permutation, i.e., they appear in the same order as in the previous phase. The final phase, phase p, consists of one request to each of the colours in $\mathcal{K}_p$.

The identity permutation π_i for the even phases i is used to ensure that, after reading the items from the first i phases into the buffer, for even i, the buffer of any tidy algorithm (defined below) will consist exactly of the elements with colours $\mathcal{K}_i$. This allows us to consider pairs of consecutive even and odd phases separately, when constructing the lower bound.

In the following lemma, we show that at any time when a current request exists and there are no items of the current service colour in the buffer, an optimal algorithm must switch to the colour of the current request, i.e., the items are served in exactly the same order as they appear in the permutations π_i.

Lemma 13 *Let $\sigma = \sigma^k_{p,\pi} \in \Psi^k$ be a request sequence, where $\pi = \{\pi_1, \pi_2, \ldots, \pi_p\}$. Let* OFF *be an optimal offline algorithm with a buffer of size k for σ. Then, for each $1 \le i \le p-1$ and $1 \le j \le k$, the $((i-1)k+j)$-th colour switch of* OFF *must be to the colour $\pi_i(c_{(i-1)k+j})$.*

Proof Let OFF be an algorithm with a buffer of size k for the sequence σ, such that for each $1 \le i \le p$ and $1 \le j \le k$, the $((i-1)k+j)$-th colour switch of OFF is to the colour $\pi_i(c_{(i-1)k+j})$. We will show that OFF will be able to process the whole input sequence, i.e., it will always have the elements of the needed colour in the buffer, and that after pk colour switches OFF will finish processing σ. As no algorithm can process σ with less than pk colour switches (as pk is the number of distinct colours), we will get that OFF is optimal. We will also show that during any of the first $(p-1)k$ colour switches, performing a colour switch to a different colour than described above results in an additional colour switch, yielding an sub-optimal algorithm.

Initially, OFF fills its buffer with the first k items which are of k distinct colours $\mathcal{K}_1$. By definition, the current request is now the second request of the colour $\pi_1(c_1)$, which is the colour switched to by OFF. OFF outputs both items of colour $\pi_1(c_1)$ after this colour switch, and an item with a new colour enters the buffer. The buffer of OFF consists again of k items with distinct colours. If any other colour c was switched to at this point, the algorithm would only output the first item of colour c, resulting in an additional colour switch to colour c later, and yielding a sub-optimal algorithm. Assume that for some $u < (p-1)k$, where $u = (i-1)k+j$ for some $1 \le i \le p-1$ and $1 \le j \le k$, the first $u-1$ colour switches have been performed as described above, and that after each colour switch the algorithm has output two items of the current service colour. We will now perform the inductive step from $u-1$ to u. The buffer of the algorithm consists of k distinct colours, and the current request is the request for $\pi_i(c_{(i-1)k+j})$. If OFF switches to the colour $\pi_i(c_{(i-1)k+j})$, it will be able to serve both items of this colour using one colour switch. Also, no other colour switch can serve two items of the same colour. This implies that an optimal algorithm must switch to $\pi_i(c_{(i-1)k+j})$ at this point.

We complete the correctness proof for OFF by noting that, after performing the first $(p-1)k$ colour switches, the buffer of OFF will contain the last k distinct colours. Therefore, k last colour switches will output the remaining $2k$ items of k distinct colours. □

7.2 Tidy Algorithms

In this section, we define a class of algorithms called tidy algorithms for serving the requests from Ψ^k. Informally, a tidy algorithm will always switch to a colour c for which it can output two items of colour c after one colour switch if such a colour c exists. For Ψ^k, that means switching to a colour c when two items of colour c are already in the buffer, or when the second request of colour c is sure to be served (i.e., when the current request is from an even phase i, where the permutation π_i is the identity permutation). Tidy algorithms are formally defined as follows.

Definition 8 For a request sequence $\sigma \in \Psi^k$, when performing a colour switch, a *tidy algorithm* $\text{ALG}^k_{\text{TIDY}}$ with a buffer of size k will switch (in order of priority) to a colour c in the buffer such that

1. $\text{ALG}^k_{\text{TIDY}}$ has the second item of colour c in the buffer,
2. $c = \pi_i(c')$ for some c', where $r(\pi_i(c'))$ is the current request and i is even,
3. c is any colour in the buffer of $\text{ALG}^k_{\text{TIDY}}$.

The following lemma shows that for $\sigma \in \Psi^k$ we can restrict our attention to *tidy* online algorithms.

Lemma 14 *Any online algorithm* ALG *can be converted to an online tidy algorithm* $\widehat{\text{ALG}}$ *such that, for any* $\sigma \in \Psi^k$, $\widehat{\text{ALG}}(\sigma) \leq \text{ALG}(\sigma)$.

Proof Without loss of generality, we assume that ALG is lazy. We define $\widehat{\text{ALG}}$ to be a lazy algorithm that will simulate ALG and maintain a queue of the colour switches performed by ALG. Whenever $\widehat{\text{ALG}}$ has to make a colour switch, it considers the colour d at the top of the queue. If switching to d does not violate the properties of a tidy algorithm, $\widehat{\text{ALG}}$ removes d from the queue and switches to colour d.

If switching to colour d violates one of the first two properties of a tidy algorithm, then switching to some colour c does not. The algorithm $\widehat{\text{ALG}}$ will switch to c. Note that, since the colour c satisfies one of the first two properties of a tidy algorithm, $\widehat{\text{ALG}}$ will serve both items of colour c on the current colour switch.

If switching to colour d violates the third property of a tidy algorithm, d corresponds to a colour that has already been fully served by $\widehat{\text{ALG}}$ (due to a colour switch that had to be performed by $\widehat{\text{ALG}}$, as otherwise $\widehat{\text{ALG}}$ would have violated one of the first two properties of a tidy algorithm). The colour d is popped from the queue and the next colour at the top of the queue is considered.

The algorithm $\widehat{\text{ALG}}$ will not perform more colour switches than ALG. □

Lemma 15 *Let* $\text{ALG}^k_{\text{TIDY}}$ *be any tidy algorithm with a buffer of size* k*, processing a request sequence* $\sigma \in \Psi^k$*. Let the current request be the first request of an odd phase* i*. Then the buffer of* $\text{ALG}^k_{\text{TIDY}}$ *contains exactly the* k *items* $\mathcal{K}_i$.

Proof Assume towards a contradiction that there is a colour $c \in \mathcal{K}_i = \{c_{(i-1)k+1}, \ldots, c_{ik}\}$ which has been already served by $\text{ALG}^k_{\text{TIDY}}$. Colour c appears for the first time in phase $i-1$. Consider time τ when $\text{ALG}^k_{\text{TIDY}}$ switched to colour c. When switching to colour c, $\text{ALG}^k_{\text{TIDY}}$ used the third rule. The buffer of $\text{ALG}^k_{\text{TIDY}}$ at time τ consisted of items of k distinct colours, such that each of the items is the first item of its colour, as otherwise $\text{ALG}^k_{\text{TIDY}}$ would use its first rule. The last item which has been read from the input must have some new colour c_j, as this item is in the buffer at time τ, and otherwise c_j would satisfy the first rule at time τ. That means that at time τ the current request is of the form $c' = \pi_{i-1}(c_\ell)$. As there are only k colours such that only one item of the colour has appeared so far in the input sequence, and the buffer of $\text{ALG}^k_{\text{TIDY}}$ contains k such colours, it means that an item of colour c' is in the buffer of $\text{ALG}^k_{\text{TIDY}}$ at time τ. Therefore, a tidy algorithm would switch to the colour c' (due to the second rule of tidy algorithms) instead of c, which contradicts the assumption that the algorithm considered is tidy. □

7.3 A Predetermined Tidy Algorithm

We will now define a class of predetermined tidy algorithms for serving the requests from Ψ^k. These will be tidy algorithms, which, before reading the first request of an odd phase i into the buffer, fix a permutation π^*_i of elements in $\mathcal{K}_i$, and, whenever a colour switch of the third priority is required, the new service colour is chosen in the order given by π^*_i. Note that from Lemma 15 the items in the buffer of the tidy algorithm at the beginning of the phase are exactly $\mathcal{K}_i$. The formal definition of a predetermined tidy algorithm is the following.

Definition 9 A *predetermined tidy* algorithm $\text{ALG}^k_{\text{PTA}}$ with a buffer of size k, processing a request sequence $\sigma \in \Psi^k$ fixes a permutation π^*_i of elements in $\mathcal{K}_i$ before reading into the buffer the first request of an odd phase i. When performing a colour switch, $\text{ALG}^k_{\text{PTA}}$ will switch (in order of priority) to a colour c in the buffer such that

1. the second item of colour c is already in the buffer;
2. $c = \pi_i(c')$ for some c', where $r(\pi_i(c'))$ is the current request and i is even;
3. $c = \pi^*_i(\ell)$, s.t. for all $j < \ell$, no items of colour $\pi^*_i(j)$ are in the buffer.

In the following, for an arbitrary request sequence $\sigma \in \Psi^k$, we will compare the number of odd indices i, such that the algorithms $\text{ALG}^k_{\text{TIDY}}$ and $\text{ALG}^k_{\text{PTA}}$ optimally serve all the colours from $\mathcal{K}_i \cup \mathcal{K}_{i+1}$ (i.e., when there is only one colour switch to each of these colours). For an algorithm ALG and a request sequence $\sigma \in \Psi^k$, let $m_{\text{ALG}}(\sigma)$ be the number of odd indices i such that the algorithm ALG performs more than $2k$ colour switches to the colours of $\mathcal{K}_i \cup \mathcal{K}_{i+1}$.

Fact 1 *For a request sequence* $\sigma \in \Psi^k$ *and any tidy algorithm* $\text{ALG}^k_{\text{TIDY}}$*, we have* $\text{ALG}^k_{\text{TIDY}}(\sigma) \geq kp + m_{\text{ALG}^k_{\text{TIDY}}}(\sigma)$.

We will now consider tidy and predetermined tidy algorithms using b bits of advice per request, where all the advice is given in advance.[1] We will show that a tidy algorithm cannot perform much better than some predetermined tidy algorithm.

Lemma 16 *Let* $\text{ALG}^k_{\text{TIDY}}$ *be an online tidy algorithm for* RBM*, using a buffer of size* k*, with* b *bits of advice per request, where all the advice is given in advance. There exists an online predetermined tidy algorithm* $\text{ALG}^k_{\text{PTA}}$ *for the same parameters, such that, for all request sequences* $\sigma \in \Psi^k$, $m_{\text{ALG}^k_{\text{PTA}}}(\sigma) \leq m_{\text{ALG}^k_{\text{TIDY}}}(\sigma)$.

Proof We will define a predetermined tidy algorithm, $\text{ALG}^k_{\text{PTA}}$, using b bits of advice per request that are received in advance, that will simulate the given algorithm $\text{ALG}^k_{\text{TIDY}}$ using the same advice bits.

The idea to determine π^*_i (i.e., the order in which $\text{ALG}^k_{\text{PTA}}$ attempts to serve the colours $\mathcal{K}_i$) for odd i is as follows. There can be only one permutation of the items in $\mathcal{K}_i$ in the request sequence, for which $\text{ALG}^k_{\text{TIDY}}$ serves the requests of the colours of $\mathcal{K}_i \cup \mathcal{K}_{i+1}$ with $2k$ colour switches. It is enough that $\text{ALG}^k_{\text{PTA}}$ finds it by simulating $\text{ALG}^k_{\text{TIDY}}$, and then sets π^*_i to behave in the same way as $\text{ALG}^k_{\text{TIDY}}$ for this particular request sequence.

More formally, for i odd, to determine every π^*_i, $\text{ALG}^k_{\text{PTA}}$ defines a σ'_i to be given as input to a simulation of $\text{ALG}^k_{\text{TIDY}}$ along with the advice received by $\text{ALG}^k_{\text{PTA}}$. The prefix of σ'_i is $\sigma[1], \ldots, \sigma[2k(i-1)+k]$, i.e., the requests seen by $\text{ALG}^k_{\text{PTA}}$ to this point. Since $\text{ALG}^k_{\text{TIDY}}$ is tidy, by Lemma 15, immediately after bringing $\sigma[2k(i-1)+k]$ into the buffer, the buffer of $\text{ALG}^k_{\text{TIDY}}$ consists of the k colours $\mathcal{K}_i$. For the next k subsequent colour switches of $\text{ALG}^k_{\text{TIDY}}$, if the colour of the j-th colour switch c_j is in $\mathcal{K}_i$, we append two request of colours c_j, c_{ik+j} to σ'_i, where c_{ik+j} is a new colour. We set $\pi^*_i(c_{(i-1)k+j}) = c_j$ and have $\text{ALG}^k_{\text{TIDY}}$ continue serving σ'_i. If c_j is not in $\mathcal{K}_i$ or $\text{ALG}^k_{\text{TIDY}}$ is unable to make a colour switch (e.g. due to an inconsistency between σ' and the advice [2]), then the simulation stops. In this case, the remaining items of π^*_i are defined in an arbitrary manner.

To show that $m_{\text{ALG}^k_{\text{PTA}}}(\sigma) \leq m_{\text{ALG}^k_{\text{TIDY}}}(\sigma)$, we need to show that for every odd index i such that $\text{ALG}^k_{\text{TIDY}}$ serves the requests of the colours of $\mathcal{K}_i \cup \mathcal{K}_{i+1}$ with $2k$ colour switches, so does $\text{ALG}^k_{\text{PTA}}$. By Lemma 13, if $\text{ALG}^k_{\text{TIDY}}$ serves the requests of the colours $\mathcal{K}_i \cup \mathcal{K}_{i+1}$ in σ with $2k$ colour switches, then the first k colour switches to serve the requests of the colours in $\mathcal{K}_i \cup \mathcal{K}_{i+1}$ must be in $\mathcal{K}_i$. In such a situation, it follows from the construction of π^*_i above that $\pi^*_i = \pi_i$ and, therefore, $\text{ALG}^k_{\text{PTA}}$ will serve the requests of the colours of $\mathcal{K}_i \cup \mathcal{K}_{i+1}$ with $2k$ colour switches. □

[1] As we are constructing a lower bound, this assumption only makes our results stronger.

[2] In such a lower bound construction, the advice string can be any sequence of bits. It is possible that appending a request of colour c_j to σ'_i may contradict an advice string. For instance, an advice string that indicates the colour of the current request to be something other than c_j. In such a case, the actions of $\text{ALG}^k_{\text{TIDY}}$ may be undefined for σ'_i and the given advice. Regardless, we know that $\text{ALG}^k_{\text{TIDY}}$ will serve the requests of the colours of π_i with more than k colour switches.

7.4 Reduction from the q-SGKH Problem

As already mentioned, for the lower bound we will assume that the algorithm receives all the advice in advance. We will use the lower bound framework as discussed in Section 2.2, i.e., we will perform a reduction from the q-SGKH problem to the reordering buffer management problem. Further, the request sequence σ_{RBM} obtained by the reduction will be in the family Ψ^k, as defined above, where k is the size of the buffer. Due to Lemma 14, we can assume that any RBM algorithm processing σ_{RBM} is a tidy algorithm, and we can use the predetermined tidy algorithm ALG_{PTA} defined in Lemma 16.

Lemma 17 *Suppose that there is a* ρ*-competitive algorithm for* RBM *with buffer of size* $k \geq 2$ *using* b *bits of advice per request, where* $1 \leq \rho \leq 1 + \frac{1}{2k}$*. Then, there exists an algorithm for* q-SGKH *using* $5kb$ *bits of advice per request, that is correct for at least* $(1-(\rho-1)2k)n$ *characters of the* n*-length string, where* $q = k!$.

Proof Let $\text{ALG}^k_{\text{RBM}}$ be a ρ-competitive algorithm for RBM using b bits of advice per request. We will design an algorithm $\text{ALG}_{q\text{-SGKH}}$ for q-SGKH that will generate a request sequence $\sigma_{\text{RBM}} \in \Psi^k$, use $\text{ALG}^k_{\text{RBM}}$ to process σ_{RBM}, and output a solution for q-SGKH based on this solution.

We will now present the construction of σ_{RBM}. The first k requests of σ_{RBM} are to k different colours. The remaining requests of σ_{RBM} are defined, as follows, in an online manner, to ensure that $\sigma_{\text{RBM}} \in \Psi^k$. Let Π be an enumeration of all the possible permutations of length k, and let $g : \Sigma \to \{1, \ldots, k!\}$ be a bijection between Σ, the alphabet of the q-SGKH problem (where $|\Sigma| = q = k!$), and an index of a k-length permutation in Π. Let n be the length of the input string $\sigma_{q\text{-SGKH}}$ for q-SGKH. Then σ_{RBM} will have $2n+1$ phases.

We now define how the algorithm generates the phases i and $i+1$, after receiving the ith request $\sigma_{q\text{-SGKH}}[i]$. Let $c_{(2i-1)k+1}, \ldots, c_{2ik+k}$ be $2k$ new colours (i.e., colours which have not appeared in σ_{RBM} yet). The algorithm $\text{ALG}_{q\text{-SGKH}}$ appends to σ_{RBM} requests to items

with the following colours

$$\big\langle \pi_{2i-1}(c_{(2i-2)k+1}), c_{(2i-1)k+1}, \ldots, \pi_{2i-1}(c_{(2i-1)k}), c_{2ik}, \\ \pi_{2i}(c_{(2i-1)k+1}), c_{2ik+1}, \ldots, \pi_{2i}(c_{2ik}), c_{2ik+k} \big\rangle \ ,$$

where π_{2i-1} is the permutation at $\Pi(g(\sigma_{q\text{-SGKH}}[i]))$ of the last k new colours of the requests already appended to $\sigma_{q\text{-SGKH}}$, i.e., colours $c_{(2i-2)k+1}, \ldots, c_{(2i-1)k}$, and π_{2i} is the identity permutation of $\big\langle c_{(2i-1)k+1}, \ldots, c_{2ik} \big\rangle$. These requests give us phases number $2i-1$ and $2i$ of the σ_{RBM} sequence.

The i-th action of $\text{ALG}_{q\text{-SGKH}}$ is based on the permutation π^*_{2i-1} of $\text{ALG}^k_{\text{PTA}}$. Recall from the definition of a predetermined tidy algorithm, that π^*_{2i-1} is determined before $\text{ALG}^k_{\text{PTA}}$ reads the first request of phase $2i-1$ into the buffer, i.e., it is determined based on the first $2i-2$ phases of σ_{RBM}, which are generated from $\sigma_{q\text{-SGKH}}[1], \ldots, \sigma_{q\text{-SGKH}}[i-1]$. The i-th action of $\text{ALG}_{q\text{-SGKH}}$ is defined as the inverse function of g on the index of π^*_{2i-1} in Π.

For each request in $\sigma_{q\text{-SGKH}}$, at most $5k$ requests are generated for σ_{RBM}, which for $\text{ALG}^k_{\text{RBM}}$ requires at most $5kb$ bits of advice. This is feasible, since $\text{ALG}_{q\text{-SGKH}}$ has $5kb$ bits of advice per request. The advice provided to $\text{ALG}^k_{\text{RBM}}$ is the same advice as provided to $\text{ALG}_{q\text{-SGKH}}$.

From the definition of σ_{RBM}, $\text{OPT}^k_{\text{RBM}}(\sigma_{\text{RBM}}) = 2kn$. Using this, the definition of $\text{ALG}_{q\text{-SGKH}}$, and Fact 1, we get that

$$\begin{aligned} &\text{ALG}^k_{\text{RBM}}(\sigma_{\text{RBM}}) \geq 2kn + m_{\text{ALG}^k_{\text{RBM}}}(\sigma_{\text{RBM}}) \iff \\ &m_{\text{ALG}^k_{\text{RBM}}}(\sigma_{\text{RBM}}) \leq \text{ALG}^k_{\text{RBM}}(\sigma_{\text{RBM}}) - 2kn \\ &\qquad = \text{ALG}^k_{\text{RBM}}(\sigma_{\text{RBM}}) - \text{OPT}^k_{\text{RBM}}(\sigma_{\text{RBM}}) \ . \qquad (2) \end{aligned}$$

Now, we can bound from above the number of incorrect characters in the output string (the number of misses) generated by $\text{ALG}_{q\text{-SGKH}}$ for $\sigma_{q\text{-SGKH}}$ as follows.

$$\begin{aligned} &\text{number of misses} \\ &\leq m_{\text{ALG}^k_{\text{RBM}}}(\sigma_{\text{RBM}}) \ \text{, from Lemma 16,} \\ &\leq \text{ALG}^k_{\text{RBM}}(\sigma_{\text{RBM}}) - \text{OPT}^k_{\text{RBM}}(\sigma_{\text{RBM}}) \ \text{, from (2),} \\ &\leq (\rho-1)\text{OPT}^k_{\text{RBM}}(\sigma_{\text{RBM}}) \ \text{, as } \text{ALG}^k_{\text{RBM}} \text{ is } \rho\text{-comp.,} \\ &= (\rho-1)2kn. \end{aligned}$$

□

We are now ready to prove the main lower bound theorem.

Theorem 6 *Any deterministic online algorithm with advice for the reordering buffer management problem with buffer of size $k \geq 2$ requires at least $\frac{1}{10} \cdot ((1-H_{k!}(1-\alpha)) \log k)$ bits of advice per request to be ρ-competitive for $1 < \rho \leq 1 + \frac{1}{2k} - \frac{1}{2k \cdot k!}$, where H_q is the q-ary entropy function and $\alpha = 1 - (\rho-1)2k$.*

Proof For $1 < \rho \leq 1 + \frac{1}{2k} - \frac{1}{2k \cdot k!}$, let $\text{ALG}^k_{\text{RBM}}$ be a ρ-competitive deterministic algorithm for the reordering buffer management problem with a buffer of size k, using b bits of advice per request. Define $q = k!$. By Lemma 17, there exists an algorithm for the q-SGKH problem with $5kb$ bits of advice per request that is correct for at least $(1-(\rho-1)2k)n$ characters of the n length string. The bounds on ρ and k imply that $q \geq 2$ and $1/q \leq \alpha < 1$. Theorem 1 implies[3] that $5kb \geq (1 - H_q(1-\alpha)) \log q$ and, therefore,

$$\begin{aligned} b &\geq \frac{(1 - H_q(1-\alpha))}{5k} \log q \\ &\geq \frac{(1 - H_{k!}(1-\alpha))}{10} \log k \ \text{, as } q = k! \geq k^{k/2}. \end{aligned}$$

□

From Theorem 6, we get the following corollary.

Corollary 1 *Any deterministic online algorithm with advice for* RBM *with a buffer of size $k \geq 2$ requires $\Omega(\log k)$ bits of advice per request to be 1-competitive.*

Proof Let ALG be a ρ-competitive deterministic online algorithm, using b bits of advice per request, for the reordering buffer management problem with buffer of size $k \geq 2$, where $1 < \rho \leq 1 + \frac{1}{2k} - \frac{1}{2k \cdot k!}$. Let $\alpha = 1 - (\rho-1)2k$. From Theorem 6, we have

$$b \geq \frac{(1 - H_{k!}(1-\alpha))}{10} \log k \ . \qquad (3)$$

As ρ approaches 1 from above, α approaches 1 from below and $H_{k!}(1-\alpha)$ approaches 0. Hence, (3) approaches $\frac{\log k}{10}$ as ρ approaches 1. □

8 Conclusion

We presented a $(1+\varepsilon)$-competitive algorithm with advice for the reordering buffer management problem, using $O(\log(1/\varepsilon))$ bits of advice per request. We complemented this result by providing a lower bound of $\Omega(\log k)$ on the number of bits of advice per request for an online algorithm to achieve a competitive ratio of 1.

A natural question is whether our approach, in particular the structure of the near-optimal solution described by the advice sequence, can be used to develop a PTAS for the offline RBM problem. Note that our algorithm can be used as an offline approximation algorithm for RBM, with an exponential running time.

[3] As noted previously, Theorem 1 holds even if the advice is received in advance.

Specifically, the algorithm can try all $2^{O(n \log 1/\varepsilon)}$ possible advice strings, for each of them simulate our algorithm with advice, and output the best solution. That gives an offline algorithm for RBM, with an approximation ratio of $(1 + \varepsilon)$, with a total running time of $2^{O(n \log 1/\varepsilon)}$, using $O(n)$ space. The naïve offline algorithm that considers all k possibilities at each request, and outputs an optimal solution, has a running time of $k^{O(n)}$.

From the online perspective, it would be interesting to obtain matching results for a more general version of the RBM problem, when the cost of switching from a colour c to a colour c' depends on the colours c and c'.

Acknowledgments We would like to thank the reviewers for their thorough reading of the paper and their helpful comments, which helped us to improve the presentation of the paper.

References

1. Standard for information technology portable operating system interface (posix(r)) base specifications, issue 7. IEEE Std 1003.1, 2013 Edition (incorporates IEEE Std 1003.1-2008, and IEEE Std 1003.1-2008/Cor 1-2013) pp. 1–3906 (2013). DOI 10.1109/IEEESTD.2013.6506091
2. Adamaszek, A., Czumaj, A., Englert, M., Räcke, H.: Almost tight bounds for reordering buffer management. In: L. Fortnow, S.P. Vadhan (eds.) STOC, pp. 607–616. ACM (2011)
3. Albers, S., Hellwig, M.: Online makespan minimization with parallel schedules. In: R. Ravi, I.L. Gørtz (eds.) Algorithm Theory - SWAT 2014 - 14th Scandinavian Symposium and Workshops, Copenhagen, Denmark, July 2-4, 2014. Proceedings, *Lecture Notes in Computer Science*, vol. 8503, pp. 13–25. Springer (2014). DOI 10.1007/978-3-319-08404-6_2. URL http://dx.doi.org/10.1007/978-3-319-08404-6_2
4. Asahiro, Y., Kawahara, K., Miyano, E.: Np-hardness of the sorting buffer problem on the uniform metric. Discrete Applied Mathematics **160**(10-11), 1453–1464 (2012)
5. Avigdor-Elgrabli, N., Rabani, Y.: An improved competitive algorithm for reordering buffer management. In: M. Charikar (ed.) SODA, pp. 13–21. SIAM (2010)
6. Avigdor-Elgrabli, N., Rabani, Y.: A constant factor approximation algorithm for reordering buffer management. In: S. Khanna (ed.) SODA, pp. 973–984. SIAM (2013)
7. Avigdor-Elgrabli, N., Rabani, Y.: An optimal randomized online algorithm for reordering buffer management. In: 54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA, pp. 1–10 (2013). DOI 10.1109/FOCS.2013.9
8. Blandford, D.K., Blelloch, G.E.: Index compression through document reordering. In: DCC, pp. 342–351. IEEE Computer Society (2002)
9. Böckenhauer, H., Hromkovic, J., Komm, D., Krug, S., Smula, J., Sprock, A.: The string guessing problem as a method to prove lower bounds on the advice complexity. Theor. Comput. Sci. **554**, 95–108 (2014). DOI 10.1016/j.tcs.2014.06.006. URL http://dx.doi.org/10.1016/j.tcs.2014.06.006
10. Böckenhauer, H.J., Komm, D., Královic, R., Královic, R.: On the advice complexity of the k-server problem. In: L. Aceto, M. Henzinger, J. Sgall (eds.) ICALP (1), *Lecture Notes in Computer Science*, vol. 6755, pp. 207–218. Springer (2011). Also as technical report at ftp://ftp.inf.ethz.ch/pub/publications/techreports/7xx/703.pdf
11. Böckenhauer, H.J., Komm, D., Královic, R., Královic, R., Mömke, T.: On the advice complexity of online problems. In: Y. Dong, D.Z. Du, O.H. Ibarra (eds.) ISAAC, *Lecture Notes in Computer Science*, vol. 5878, pp. 331–340. Springer (2009)
12. Chan, H.L., Megow, N., Sitters, R., van Stee, R.: A note on sorting buffers offline. Theor. Comput. Sci. **423**, 11–18 (2012)
13. Dobrev, S., Královič, R., Pardubská, D.: How much information about the future is needed? In: SOFSEM'08: Proceedings of the 34th conference on Current trends in theory and practice of computer science, pp. 247–258. Springer-Verlag, Berlin, Heidelberg (2008)
14. Dohrau, J.: Online makespan scheduling with sublinear advice. In: G.F. Italiano, T. Margaria-Steffen, J. Pokorný, J. Quisquater, R. Wattenhofer (eds.) SOFSEM 2015: Theory and Practice of Computer Science - 41st International Conference on Current Trends in Theory and Practice of Computer Science, Pec pod Sněžkou, Czech Republic, January 24-29, 2015. Proceedings, *Lecture Notes in Computer Science*, vol. 8939, pp. 177–188. Springer (2015). DOI 10.1007/978-3-662-46078-8_15. URL http://dx.doi.org/10.1007/978-3-662-46078-8_15
15. Dorrigiv, R., He, M., Zeh, N.: On the advice complexity of buffer management. In: K.M. Chao, T. sheng Hsu, D.T. Lee (eds.) ISAAC, *Lecture Notes in Computer Science*, vol. 7676, pp. 136–145. Springer (2012)
16. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. Theor. Comput. Sci. **412**(24), 2642–2656 (2011)
17. Englert, M., Räcke, H., Westermann, M.: Reordering buffers for general metric spaces. Theory of Computing **6**(1), 27–46 (2010)
18. Englert, M., Westermann, M.: Reordering buffer management for non-uniform cost models. In: L. Caires, G.F. Italiano, L. Monteiro, C. Palamidessi, M. Yung (eds.) ICALP, *Lecture Notes in Computer Science*, vol. 3580, pp. 627–638. Springer (2005)
19. Gutenschwager, K., Spiekermann, S., Voß, S.: A sequential ordering problem in automotive paint shops. Internat. J. Production Research **42**(9), 1865–1878 (2004). DOI 10.1080/00207540310001646821
20. Hromkovic, J., Královic, R., Královic, R.: Information complexity of online problems. In: P. Hlinený, A. Kucera (eds.) MFCS, *Lecture Notes in Computer Science*, vol. 6281, pp. 24–36. Springer (2010)
21. Komm, D., Královic, R.: Advice complexity and barely random algorithms. RAIRO - Theor. Inf. and Applic. **45**(2), 249–267 (2011)
22. Krokowski, J., Räcke, H., Sohler, C., Westermann, M.: Reducing state changes with a pipeline buffer. In: B. Girod, M.A. Magnor, H.P. Seidel (eds.) VMV, p. 217. Aka GmbH (2004)
23. Lewis, H.R., Papadimitriou, C.H.: Elements of the Theory of Computation, 2nd edn. Prentice Hall PTR, Upper Saddle River, NJ, USA (1997)
24. Räcke, H., Sohler, C., Westermann, M.: Online scheduling for sorting buffers. In: R.H. Möhring, R. Raman (eds.)

ESA, *Lecture Notes in Computer Science*, vol. 2461, pp. 820–832. Springer (2002)
25. Renault, M.P., Rosén, A.: On online algorithms with advice for the k-server problem. Theory of Computing Systems pp. 1–19 (2012). DOI 10.1007/s00224-012-9434-z
26. Renault, M.P., Rosén, A., van Stee, R.: Online algorithms with advice for bin packing and scheduling problems. Theor. Comput. Sci. **600**, 155–170 (2015). DOI 10.1016/j.tcs.2015.07.050. URL http://dx.doi.org/10.1016/j.tcs.2015.07.050