

# On the List Update Problem with Advice

Joan Boyar<sup>a</sup>, Shahin Kamali<sup>b</sup>, Kim S. Larsen<sup>a</sup>, Alejandro López-Ortiz<sup>c</sup>

<sup>a</sup>*University of Southern Denmark, Department of Mathematics and Computer Science,  
Campusvej 55, 5230 Odense M, Denmark,  
{joan,kslarsen}@imada.sdu.dk*

<sup>b</sup>*Massachusetts Institute of Technology, Computer Science and Artificial Intelligence  
Laboratory, 32 Vassar Street, Cambridge, MA 02139, U.S.A.,  
skamali@mit.edu*

<sup>c</sup>*University of Waterloo, School of Computer Science, 200 University Avenue West,  
Waterloo, ON N2L 3G1, Canada,  
alopez-o@cs.uwaterloo.ca*

---

## Abstract

We study the online list update problem under the advice model of computation. Under this model, an online algorithm receives partial information about the unknown parts of the input in the form of some bits of advice generated by a benevolent offline oracle. We show that advice of linear size is required and sufficient for a deterministic algorithm to achieve an optimal solution or even a competitive ratio better than  $15/14$ . On the other hand, we show that surprisingly two bits of advice are sufficient to break the lower bound of 2 on the competitive ratio of deterministic online algorithms and achieve a deterministic algorithm with a competitive ratio of  $1.\bar{6}$ . In this upper-bound argument, the bits of advice determine the algorithm with smaller cost among three classical online algorithms, `TIMESTAMP` and two members of the `MTF2` family of algorithms. We also show that `MTF2` algorithms are 2.5-competitive.

*Keywords:*

List Update, Advice Complexity, Competitive Analysis, Online Algorithms

---

## 1. Introduction

List update is a well-studied problem in the context of online algorithms. The input is a sequence of requests to items of a list; the requests appear in a sequential and online manner, i.e., while serving a request an algorithm

cannot look at the incoming requests. A request involves accessing an item in the list.<sup>1</sup> To access an item, an algorithm should linearly probe the list; each probe has a cost of 1, and accessing an item in the  $i$ th position results in a cost of  $i$ . The goal is to maintain the list in a way to minimize the total cost. An algorithm can make a *free exchange* to move an accessed item somewhere closer to the front of the list. Further, it can make any number of *paid exchanges*, each having a cost of 1, to swap the positions of any two consecutive items in the list.

Similar to other online problems, the standard method for comparing online list update algorithms is competitive analysis. The competitive ratio of an online algorithm  $A$  is the maximum ratio between the cost of  $A$  for serving any sequence and the cost of  $\text{OPT}$  for serving the same sequence. Here,  $\text{OPT}$  is an optimal offline algorithm. It is known that, for a list of length  $l$ , no deterministic online algorithm can achieve a competitive ratio better than  $2l/(l+1)$  (reported in [22]); this converges to 2 for large lists. There are 2-competitive algorithms (hence best possible online algorithms) for the problem; these include Move-To-Front (MTF) [30] and TIMESTAMP [2].

Although competitive analysis has been accepted as the standard tool for comparing online algorithms, there are objections to it. One relevant objection is that assuming a total lack of information about the future is unrealistic in many applications. This is particularly the case for the list update problem when it is used as a method for compression [9]. In this application, each character of a text is treated as an item in the list, and the text as the input sequence which is parsed (revealed) in a sequential manner. A compression algorithm can be devised from a list update algorithm  $A$  by writing the access cost of  $A$  for serving each character in unary.<sup>2</sup> Hence, the size of the compressed file is roughly equal to the access cost of the list update algorithm. In this application, it is possible to include some partial information about the structure of the sequence (text) in the compressed file, for example, which of three algorithms was used to do the compression. This partial information could potentially be stored using very little space com-

---

<sup>1</sup>Similar to other works, we consider the *static* list update problem in which there is no insertion or deletion.

<sup>2</sup>Encodings other than unary correspond to other cost models for list update, and, naturally, encoding positions in binary would improve the compression [9]. The choice of algorithm is also important and tests indicate that TIMESTAMP may be a better algorithm for this than MTF [3]

pared to the subsequent savings in the size of the compressed file compared with the original file, due to the availability of the partial information [24].

Advice complexity provides an alternative for the analysis of online problems. Under the advice model, the online algorithm is provided with some bits of advice, generated by a benevolent offline oracle with infinite computational power. This reduces the power of the adversary relative to the online algorithm. Variant models are proposed and studied for the advice complexity model [16, 17, 13, 12]. Here, we use a natural model from [13, 12] that assumes advice bits are written once on a tape before the algorithm starts, and the online algorithm can access the tape sequentially from the beginning at any time. The advice complexity of an algorithm is then the worst case number of bits read from the tape, as a function of the length of the input. Since its introduction, many online problems have been studied under the advice model. These include classical online problems such as paging [13, 21, 25],  $k$ -server [17, 12, 28, 20], bin packing [15, 7], and various coloring problems [10, 18, 29].

### 1.1. Contribution

When studying an online problem under the advice model, the first question to answer is how many bits of advice are required to achieve an optimal solution. We show that advice of size  $\text{OPT}(\sigma)$  is sufficient to optimally serve a sequence  $\sigma$ , where  $\text{OPT}(\sigma)$  is the cost of an optimal offline algorithm for serving  $\sigma$ , and it is linear in the length of the sequence, assuming that the length of the list is a constant. We further show that advice of linear size is required to achieve a deterministic algorithm with a competitive ratio better than  $15/14$ .

Another important question is how many bits of advice are required to break the lower bound on the competitive ratio of any deterministic algorithm. We answer this question by introducing a deterministic algorithm that receives two bits of advice and achieves a competitive ratio of at most  $1.\bar{6}$ . The advice bit for a sequence  $\sigma$  simply indicates the best option between three online algorithms for serving  $\sigma$ . These three algorithms are **TIMESTAMP**, **MTF-Odd (MTFO)** and **MTF-Even (MTFE)**. **TIMESTAMP** inserts an accessed item  $x$  in front of the first item  $y$  (from the front of the list) that precedes  $x$  in the list and was accessed at most once since the last access to  $x$ . If there is no such item  $y$  or  $x$  is accessed for the first time, no items are moved. **MTFO** (resp. **MTFE**) moves a requested item  $x$  to the front on every odd (resp. even) request to  $x$ .

Our results indicate that if we dismiss `TIMESTAMP` and take the better algorithm between `MTFO` and `MTFE`, the competitive ratio of the resulting algorithm is no better than 1.75. We also study the competitiveness of `MTFE` and `MTFO`, and more generally any algorithm that belongs to the family of Move-To-Front-Every-Other-Access (also known as `MTF2` algorithms). We show that these algorithms have competitive ratios of 2.5.

## 2. Optimal solution

In this section, we provide upper and lower bounds on the number of advice bits required to optimally serve a sequence. We start with an upper bound:

**Theorem 2.1.** *Under the advice model,  $\text{OPT}(\sigma) - n$  bits of advice are sufficient to achieve an optimal solution for any sequence  $\sigma$  of length  $n$ , where  $\text{OPT}(\sigma)$  is the cost of an optimal algorithm for serving  $\sigma$ .*

*Proof.* It is known that there is an optimal algorithm that moves items using only a family of paid exchanges called *subset transfer* [26]. In a subset transfer, before serving a request to an item  $x$ , a subset  $S$  of items preceding  $x$  in the list is moved (using paid exchanges) to just after  $x$  in the list, so that the relative order of items in  $S$  among themselves remains unchanged. Consider an optimal algorithm `OPT` which only moves items via subset transfer. Before a request to  $x$  at index  $i$ , an online algorithm can read  $i - 1$  bits from the advice tape, indicating (bit vector style) the subset which should be moved behind  $x$ . Provided with this, the algorithm can always maintain the same list as `OPT`. The total number of bits read by the algorithm will be at most  $\text{OPT}(\sigma) - n$ .  $\square$

The above theorem implies that for lists of constant size, advice of linear size is sufficient to optimally serve a sequence. We show that advice of linear size is also required to achieve any competitive ratio smaller than 15/14.

In order to prove this lower bound, we first define a large set of possible sequences, defined from bit strings, where an online algorithm with a good competitive ratio must make the right decision many times. This ends up essentially being a ‘guess’, for every fifth request, as to what the next request will be. Thus, a lookahead of 1 (being able to see the next request before making a decision about the current one, the ‘weak lookahead’ as defined in [1]), would be sufficient to perform optimally on these particular sequences.

Less than linear advice leaves some sequences where the algorithm does not ‘guess’ well enough.

Consider instances of the list update problem on a list of two items  $x$  and  $y$  which are defined as follows. Assume the list is ordered as  $[x, y]$  before the first request. Also, to make our explanation easier, assume that the length of the sequence,  $n$ , is divisible by 5. Consider an arbitrary bitstring  $B$ , of size  $n/5$ , which we refer to as the *defining bitstring*. Let  $\sigma$  denote the list update sequence defined from  $B$  in the following manner: For each bit in  $B$ , there are five requests in  $\sigma$ , which we refer to as a *round*. We say that a round in  $\sigma$  is of type 0 (resp. 1) if the bit associated with it in  $B$  is 0 (resp. 1). For a round of type 0,  $\sigma$  will contain the requests  $yyyx$ , and for a round of type 1, the requests  $xxxx$ . For example, if  $B = 011\dots$ , we will have  $\sigma = \langle yyyx, xxxx, xxxx, \dots \rangle$ .

Since the last two requests in a round are to the same item  $x$ , it makes sense for an online algorithm to move  $x$  to the front after the first access. This is formalized in the following lemma.

**Lemma 2.2.** *For any online list update algorithm  $\mathbb{A}$  serving a sequence  $\sigma$  created from a defining bitstring, there is another algorithm whose cost is not more than  $\mathbb{A}$ ’s cost for serving  $\sigma$  and that ends each round with the list in the order  $[x, y]$ .*

*Proof.* Let  $R_t$  denote the first round such that the ordering of the list maintained by  $\mathbb{A}$  is  $[y, x]$  at the end of the round. So,  $\mathbb{A}$  incurs a cost of 4 for the last two requests of the round (which are both to  $x$ ) and a cost of 1 for the first request of the next round (which is to  $y$ ). This sums to a cost of 5 for these three requests. Consider an alternative algorithm  $\mathbb{A}'$  which moves  $x$  to the front after the first access to  $x$  in  $R_t$ . The cost of  $\mathbb{A}'$  for the last two requests of  $R_t$  is 3. Also,  $\mathbb{A}'$  incurs a cost of 2 to access the first request of the next round. Hence,  $\mathbb{A}'$  incurs a cost of at most 5, equal to the cost of  $\mathbb{A}$  for these three requests. After the access to  $y$  in the second position of the list,  $\mathbb{A}'$  can reestablish the same ordering  $\mathbb{A}$  uses from that point (using at most one free exchange). Consequently, the cost of  $\mathbb{A}'$  is not more than  $\mathbb{A}$ . Repeating this argument for all rounds completes the proof.  $\square$

Provided with the above lemma, we can restrict our attention to algorithms that maintain the ordering  $[x, y]$  at the end of each round. In what follows, by an ‘online algorithm’ we mean an online algorithm with this property.

**Lemma 2.3.** *The cost of an optimal algorithm for serving a sequence of length  $n$ , where the sequence is created from a defining bitstring, is at most  $7n/5$ .*

*Proof.* Since there are  $n/5$  rounds, it is sufficient to show that there is an algorithm which incurs a cost of at most 7 for each round. Consider an algorithm that works as follows: For a round of type 0, the algorithm moves  $y$  to the front after the first access to  $y$ . It also moves  $x$  to the front after the first access to  $x$ . Hence, it incurs a cost  $2+1+1+2+1 = 7$ . For a round of type 1, the algorithm does not move any item and incurs a cost of  $2+1+1+1+1 = 6$ . In both cases, the list ordering is  $[x, y]$  at the end of the round and the same argument can be repeated for the next rounds.  $\square$

For a round of type 0 (with requests to  $yyyxx$ ), if an online algorithm  $\mathbb{A}$  moves each of  $x$  and  $y$  to the front after the first accesses, it has cost 7. If it does not move  $y$  immediately, it has cost at least 8. For a round of type 1 (i.e., a round of requests to  $yxxx$ ), if an algorithm does no rearrangement, its cost will be 6; otherwise its cost is at least 7. To summarize, an online algorithm should ‘guess’ the type of each round and act accordingly after accessing the first request of the round. If the algorithm makes a wrong guess, it incurs a ‘penalty’ of at least 1 unit. This relates our problem to the binary string guessing problem, defined in [17, 11].

**Definition 2.4** ([11]). *The Binary String Guessing Problem with known history (2-SGKH) is the following online problem. The input is a bitstring of length  $m$ , and the bits are revealed one by one. For each bit  $b_t$ , the online algorithm  $\mathbb{A}$  must guess if it is a 0 or a 1. After the algorithm has made a guess, the value of  $b_t$  is revealed to the algorithm.*

**Lemma 2.5** ([11]). *On an input of length  $m$ , any deterministic algorithm for 2-SGKH that is guaranteed to guess correctly on more than  $\alpha m$  bits, for  $1/2 \leq \alpha < 1$ , needs to read at least  $(1 + (1 - \alpha) \log(1 - \alpha) + \alpha \log \alpha)m$  bits of advice.<sup>3</sup>*

We reduce the 2-SGKH problem to the list update problem:

**Theorem 2.6.** *On an input of size  $n$ , any algorithm for the list update problem which achieves a competitive ratio of  $\gamma$  ( $1 < \gamma \leq 15/14$ ) needs to*

---

<sup>3</sup>In this paper we use  $\log n$  to denote  $\log_2(n)$ .

read at least  $(1 + (7\gamma - 7)\log(7\gamma - 7) + (8 - 7\gamma)\log(8 - 7\gamma))/5 \cdot n$  bits of advice.

*Proof.* Consider the 2-SGKH problem for an arbitrary bitstring  $B$ . Given an online algorithm  $\mathbb{A}$  for the list update problem, define an algorithm for 2-SGKH as follows: Consider an instance  $\sigma$  of the list update problem on a list of length 2 where  $\sigma$  has  $B$  as its defining bitstring, and run  $\mathbb{A}$  to serve  $\sigma$ . For the first request  $y$  in each round in  $\sigma$ ,  $\mathbb{A}$  should decide whether to move it to the front or not. The algorithm for the 2-SGKH problem guesses a bit as being 0 (resp. 1) if, after accessing the first item requested in the round associated with the bit in  $B$ ,  $\mathbb{A}$  moves it to front (resp. keeps it at its position). As mentioned earlier, for each incorrect guess  $\mathbb{A}$  incurs a penalty of at least 1 unit, i.e.,  $\mathbb{A} \geq \text{OPT} + w$ , where  $w$  is the number of wrong guesses for critical requests. Since  $\mathbb{A}$  has a competitive ratio of  $\gamma$ , we have  $\mathbb{A} \leq \gamma \text{OPT}$ . Consequently, we have  $w \leq (\gamma - 1) \text{OPT}(\sigma)$  and by Lemma 2.3,  $w \leq 7(\gamma - 1)/5 \cdot n$ . This implies that if  $\mathbb{A}$  has a competitive ratio of  $\gamma$ , the 2-SGKH algorithm makes at most  $7(\gamma - 1)/5 \cdot n$  mistakes for an input bitstring  $B$  of size  $n/5$ , i.e., at least  $n/5 - 7(\gamma - 1)/5 \cdot n = (8 - 7\gamma) \cdot n/5$  correct guesses. Define  $\alpha = 8 - 7\gamma$ , and note that  $\alpha$  is in the range  $[1/2, 1)$  when  $\gamma$  is in the range stated in the lemma. By Lemma 2.5, at least  $(1 + (1 - \alpha)\log(1 - \alpha) + \alpha \log \alpha)n/5$  bits of advice are required by such a 2-SGKH algorithm. Replacing  $\alpha$  with  $8 - 7\gamma$  completes the proof.  $\square$

Thus, to obtain a competitive ratio better than  $15/14$ , a linear number of bits of advice is required. For example, to achieve a competitive ratio of 1.01, at least  $0.12n$  bits of advice are required. Theorems 2.1 and 2.6 imply the following corollary.

**Corollary 2.7.** *For any list of fixed length  $n$ ,  $\Theta(n)$  bits of advice are required and sufficient to achieve an optimal solution for the list update problem. Also,  $\Theta(n)$  bits of advice are required and sufficient to achieve a 1-competitive algorithm.*

### 3. An algorithm with two bits of advice

In this section we show that two bits of advice are sufficient to break the lower bound of 2 on the competitive ratio of deterministic algorithms and achieve a deterministic online algorithm with a competitive ratio of  $1.\bar{6}$ . The two bits of advice for a sequence  $\sigma$  indicate which of the three algorithms

TIMESTAMP, MTF-Odd (MTFO) and MTF-Even (MTFE), have the lower cost for serving  $\sigma$ . Recall that MTFO (resp. MTFE) moves a requested item  $x$  to the front on every odd (resp. even) request to  $x$ . We prove the following theorem:

**Theorem 3.1.** *For any sequence  $\sigma$ , we have either  $\text{TIMESTAMP}(\sigma) \leq 1.6 \text{OPT}(\sigma)$ ,  $\text{MTFO}(\sigma) \leq 1.6 \text{OPT}(\sigma)$ , or  $\text{MTFE}(\sigma) \leq 1.6 \text{OPT}(\sigma)$ .*

To prove the theorem, we show that for any sequence  $\sigma$ ,  $\text{TIMESTAMP}(\sigma) + \text{MTFO}(\sigma) + \text{MTFE}(\sigma) \leq 5 \text{OPT}(\sigma)$ . We note that all three algorithms have the *projective property*, meaning that the relative order of any two items only depends on the requests to those items and their initial order in the list (and not on the requests to other items). MTFO (resp. MTFE) is projective since in its list an item  $y$  precedes  $x$  if and only if the last odd (resp. even) access to  $y$  is more recent than the last odd (resp. even) access to  $x$ . In the lists maintained by TIMESTAMP, item  $y$  precedes item  $x$  if and only if in the projected sequence on  $x$  and  $y$ ,  $y$  was requested twice after the second to last request to  $x$  or the most recent request was to  $y$  and  $x$  has been requested at most once. Hence, TIMESTAMP also has the projective property.

Similar to most other work for the analysis of projective algorithms,<sup>4</sup> we consider the *partial cost model*, in which accessing an item in position  $i$  is defined to have cost  $i - 1$ . We say an algorithm is *cost-independent* if its decisions are independent of the cost it has paid for previous requests. The cost of any cost-independent algorithm for serving a sequence of length  $n$  decreases  $n$  units under the partial cost model when compared to the *full* cost model. Hence, any upper bound for the competitive ratio of a cost-independent algorithm under the partial cost model can be extended to the full cost model.

To prove an upper bound on the competitive ratio of a projective algorithm under the partial cost model, it is sufficient to prove that the claim holds for lists of size 2. The reduction to lists of size two is done by applying a *factoring lemma*, which holds for algorithms not using paid exchanges, ensuring that the total cost of a projective algorithm  $\mathbb{A}$  for serving a sequence  $\sigma$  can be formulated as the sum of the costs of  $\mathbb{A}$  for serving projected sequences of two items. A projected sequence of  $\sigma$  on two items  $x$  and  $y$  is a

---

<sup>4</sup>Almost all existing algorithms for the list update problem are projective; the only exceptions are TRANSPOSE, Move-Fraction [30], and SPLIT [22]; see [23] for a survey.



copy of  $\sigma$  in which all items except  $x$  and  $y$  are removed. We refer the reader to [14, p. 16] for details on the factoring lemma. Since MTFO, MTFE, and TIMESTAMP do not use paid exchanges and since they are projective and cost-independent, to prove Theorem 3.1, it suffices to prove the following lemma:

**Lemma 3.2.** *Under the partial cost model, for any sequence  $\sigma_{xy}$  of two items, we have  $\text{MTFO}(\sigma_{xy}) + \text{MTFE}(\sigma_{xy}) + \text{TIMESTAMP}(\sigma_{xy}) \leq 5 \cdot \text{OPT}(\sigma_{xy})$ .*

Before proving the above lemma, we study the aggregated cost of MTFO and MTFE on certain subsequences of two items. One way to think of these algorithms is to imagine they maintain a bit for each item. On each request, the bit of the item is flipped; if it becomes ‘0’, the item is moved to the front. Note that the bits of MTFO and MTFE are complements of each other. Thus, we can think of them as one algorithm started on complementary bit sequences. We say a list is in state  $[ab]_{(i,j)}$  if item  $a$  precedes  $b$  in the list and the bits maintained for  $a$  and  $b$  are  $i$  and  $j$  ( $i, j \in \{0, 1\}$ ), respectively. To study the value of  $\text{OPT}(\sigma_{xy})$ , we consider an offline algorithm which uses a free exchange to move an accessed item from the second position to the front of the list if and only if the next request is to the same item. It is known that this algorithm is optimal for lists of two items [26].

**Lemma 3.3.** *Consider a subsequence of two items  $a$  and  $b$  of the form  $\langle (ba)^{2i} \rangle$ , i.e.,  $i$  repetitions of  $\langle baba \rangle$ . Assume the initial ordering is  $[ab]$ . The cost of each of MTFO and MTFE for serving the subsequence is  $3i$  (under the partial cost model). Moreover, at the end of serving the subsequence, the ordering of items in the list maintained by at least one of the algorithms is  $[ab]$ .*

*Proof.* We refer to a repetition of  $baba$  as a *round*. We show that MTFO and MTFE have a cost of 3 for serving each round. Assume the bits associated with both items are ‘0’ before serving  $baba$ . The first request has a cost of 1 and  $b$  remains in the second position, the second request has cost 0, and the remaining requests each have a cost of 1. In total, the cost of the algorithm is 3. The other cases (when items have different bits) are handled similarly. Table 1 includes a summary of all cases. As illustrated in the table, if the bits maintained for  $a$  and  $b$  before serving  $baba$  are  $(0, 0)$ ,  $(0, 1)$ , or  $(1, 1)$ , the list order will be  $[ab]$  after serving the round. Since both  $a$  and  $b$  are requested twice, the bits will be also the same after serving  $baba$ . Hence, in these three

Bits for $(a, b)$	Cost for $\langle baba \rangle$	Orders before accessing items	Final order
(0, 0)	$1 + 0 + 1 + 1 = 3$	$[a\downarrow b] [a\downarrow b] [\overleftarrow{a}b] [\overleftarrow{b}a]$	$[ab]$
(0, 1)	$1 + 1 + 0 + 1 = 3$	$[\overleftarrow{a}b] [b\downarrow a] [b\downarrow a] [\overleftarrow{b}a]$	$[ab]$
(1, 0)	$1 + 0 + 1 + 1 = 3$	$[a\downarrow b] [a\downarrow b] [\overleftarrow{a}b] [b\downarrow a]$	$[ba]$
(1, 1)	$1 + 1 + 1 + 0 = 3$	$[\overleftarrow{a}b] [\overleftarrow{b}a] [a\downarrow b] [a\downarrow b]$	$[ab]$

Table 1: Assuming the initial ordering of items is  $[ab]$ , the cost of a both MTFO and MTFE for serving subsequence  $\langle baba \rangle$  is 3 (under the partial cost model). The final ordering of the items will be  $[ab]$  in three of the cases.

cases, the same argument can be repeated to conclude that the list order will be  $[ab]$  at the end of serving  $(ba)^{2i}$ . Since the bits maintained for the items are complements in MTFE and MTFO, at least one of them starts with bits (0, 0), (0, 1), or (1, 1) for  $a$  and  $b$ ; consequently, at least one algorithm ends up with state  $[ab]$  at the end.  $\square$

**Lemma 3.4.** *Consider a subsequence of two items  $a$  and  $b$  which has the form  $\langle baa \rangle$ . The total cost that MTFE and MTFO incur together for serving this subsequence is at most 4 (under the partial cost model).*

*Proof.* If the initial order of  $a$  and  $b$  is  $[ba]$ , the first request has no cost, and each algorithm incurs a total cost of at most 2 for the other two requests of the sequence. Hence, the aggregated cost of the two algorithms is 4. Next, assume the initial order is  $[ab]$ . Assume the bits maintained by one of the algorithms for  $a$  and  $b$  are (1, 0), respectively. As illustrated in Table 2, this algorithm incurs a cost of 1 for serving  $baa$ ; the other algorithm incurs a cost of 3. In total, the algorithms incur a cost of 4. In the other case, when bits maintained for  $a$  and  $b$  are both ‘0’ in one algorithm (consequently, both are ‘1’ in the other algorithm), the total cost of the algorithms for serving  $\langle baa \rangle$  is 3.  $\square$

Using Lemmas 3.3 and 3.4, we are ready to prove Lemma 3.2:

*Proof of Lemma 3.2, and consequently Theorem 3.1.* Consider a sequence  $\sigma_{xy}$  of two items  $x$  and  $y$ . We use the *phase partitioning technique* as introduced in [4], even using the same partitioning, though later considering more sub-cases. We partition  $\sigma_{xy}$  into *phases* which are defined inductively as follows. Assume we have defined phases up until, but not including, the  $t$ th request

Initial order	Bits for $(a, b)$	Cost for $\langle baa \rangle$	Orders before accessing items	Bits and Costs (other algorithm)	Total cost (both algs.)
$[ab]$	(0,0)	$1 + 0 + 0 = 1$	$\begin{matrix} \downarrow \\ [ab] \end{matrix} \begin{matrix} \downarrow \\ [a\ b] \end{matrix} \begin{matrix} \downarrow \\ [a\ b] \end{matrix}$	$(1, 1) \rightarrow 2$	$1 + 2 = 3$
$[ab]$	(0,1)	$1 + 1 + 1 = 3$	$\begin{matrix} \leftarrow \\ [ab] \end{matrix} \begin{matrix} \downarrow \\ [ba] \end{matrix} \begin{matrix} \leftarrow \\ [ba] \end{matrix}$	$(1, 0) \rightarrow 1$	$3 + 1 = 4$
$[ab]$	(1,0)	$1 + 0 + 0 = 1$	$\begin{matrix} \downarrow \\ [ab] \end{matrix} \begin{matrix} \downarrow \\ [a\ b] \end{matrix} \begin{matrix} \downarrow \\ [a\ b] \end{matrix}$	$(0, 1) \rightarrow 3$	$1 + 3 = 4$
$[ab]$	(1,1)	$1 + 1 + 0 = 2$	$\begin{matrix} \leftarrow \\ [ab] \end{matrix} \begin{matrix} \leftarrow \\ [ba] \end{matrix} \begin{matrix} \downarrow \\ [a\ b] \end{matrix}$	$(0, 0) \rightarrow 1$	$2 + 1 = 3$
$[ba]$	(0,0) (0,1) (1,0) (1,1)	$\leq 0 + 1 + 1 = 2$	-	$\leq 2$	$2 + 2 = 4$

Table 2: The total cost of MTF0 and MTFE for serving a sequence  $\langle baa \rangle$  is at most 4 (under the partial cost model). Note that the bits maintained by these algorithms for each item are complements of each other.

( $t \geq 1$ ) and the relative order of the two items is  $[xy]$  before the  $t$ th request. Then the next phase is of *type 1* and is of one of the following forms ( $j \geq 0$  and  $k \geq 1$ ):

$$(a) x^j yy \quad (b) x^j (yx)^k yy \quad (c) x^j (yx)^k x$$

In case the relative order of the items is  $[yx]$  before the  $t$ th request, the phase has type 2 and its form is exactly the same as above with  $x$  and  $y$  interchanged. Note that, after two consecutive requests to an item, TIMESTAMP, MTF0 and MTFE all have that item in the front of the list. So, after serving each phase, the relative order of items is the same for all three algorithms. This implies that  $\sigma_{xy}$  is partitioned in the same way for all three algorithms. To prove the lemma, we show that its statement holds for every phase.

Table 3 shows the costs incurred by all three algorithms as well as OPT for each phase. Note that phases of the form (b) and (c) are divided into two cases, depending on whether  $k$  is even or odd. We discuss the different phases of type 1 separately. Similar analyses, with  $x$  and  $y$  interchanged, apply to the phases of type 2. Note that before serving a phase of type 1, the list is ordered as  $[xy]$  and the first  $j$  requests to  $x$  have no cost.

Consider phases of form (a),  $x^j yy$ . MTF0 and MTFE incur a total cost of 3 for serving  $yy$  (one of them moves  $y$  to the front after the first request, while the other keeps it in the second position). TIMESTAMP incurs a cost of 2 for serving  $yy$  (it does not move it to the front after the first request). So, in total, the three algorithms incur an aggregated cost of 5. On the other hand, OPT incurs a cost of 1 for the phase. So, the ratio between the sum of the costs of the algorithms and the cost of OPT is 5.

Next, consider phases of the form (b). TIMESTAMP incurs a cost of  $2k$

Phase	ALGMIN	ALGMAX	TIMESTAMP	Sum (ALGMIN + ALGMAX + TIMESTAMP)	OPT'	$\frac{\text{Sum}}{\text{OPT}'}$
$x^j yy$	1	2	2	5	1	5
$x^j (yx)^{2i} yy$	$\leq 3i + 1$	$\leq 3i + 2$	$2 \cdot 2i = 4i$	$\leq 10i + 3$	$2i + 1$	$< 5$
$x^j (yx)^{2i-2} yxyy$	$\leq 3(i-1) + 1$ + ALGMIN( $\langle xy \rangle$ )	$\leq 3(i-1) + 1$ + ALGMAX( $\langle xy \rangle$ )	$2(2i-1)$ $= 4i - 2$	$\leq 6(i-1) + 2 + 4$ $+ (4i-2) = 10i - 2$	$2i$	$< 5$
$x^j (yx)^{2i} x$	$\leq 3i$	$\leq 3i + 1$	$2 \cdot 2i - 1$ $= 4i - 1$	$\leq (6i + 1) + (4i - 1)$ $= 10i$	$2i$	$\leq 5$
$x^j (yx)^{2i-2} yxx$	$\leq 3(i-1)$ + ALGMIN( $\langle yxx \rangle$ )	$\leq 3(i-1)$ + ALGMAX( $\langle yxx \rangle$ )	$2 \cdot (2i-1) - 1$ $= 4i - 3$	$\leq 6(i-1) + 4$ $+ (4i-3) = 10i - 5$	$2i - 1$	$\leq 5$

Table 3: For use in the proof of Lemma 3.2, we list the costs of MTFO, MTFE, and TIMESTAMP for a phase of type 1 (the phase has type 1, i.e., the initial ordering of items is  $xy$ ). The ratio between the aggregated cost of algorithms and the cost of OPT for each phase is at most 5. ALGMIN (resp. ALGMAX) is the algorithm among MTFO and MTFE, which incurs less (resp. more) cost for the phase. Note that the costs are under the partial cost model.

for serving the phase; it incurs a cost of 1 for all requests in  $(yx)^{2i}$  except the very first request to  $x$ , and a cost of 1 for serving the second to last request to  $y$ . Assume  $k$  is even and we have  $k = 2i$  for some  $i \geq 1$ , so the phase looks like  $x^j (yx)^{2i} yy$ . By Lemma 3.3, the cost incurred by MTFO and MTFE is  $3i$  for serving  $(yx)^{2i}$ . We show that for the remaining two requests to  $y$ , MTFO and MTFE incur an aggregated cost of at most 3. If the list maintained by any of the algorithms is ordered as  $[yx]$  before serving  $yy$ , that algorithm incurs a cost of 0 while the other algorithm incurs a cost of at most 2 for these requests; in total, the cost of both algorithms for serving  $yy$  will be at most 2. If the lists of both algorithms are ordered as  $[xy]$ , one of the algorithms incurs a cost of 1 and the other incurs a cost of 2 (depending on the bit they keep for  $y$ ). In conclusion, MTFO and MTFE incur a total cost of at most  $6i + 3$ . TIMESTAMP incurs a cost of  $2k = 4i$ , while OPT incurs a cost of  $2i + 1$  for the phase. To conclude, the aggregated cost of all algorithms is at most  $10i + 3$  compared to  $2i + 1$  for OPT, and the ratio between them is less than 5.

Next, assume  $k$  is odd and we have  $k = 2i - 1$ , i.e., the phase has the form  $x^j (yx)^{2i-2} yxyy$ . The total cost of MTFO and MTFE for  $(yx)^{2i-2}$  is  $2(3(i-1))$  (Lemma 3.3), the total cost for the next request to  $y$  is at most 2, and the total cost for subsequent  $yxyy$  is at most 4 (Lemma 3.4). In total, MTFO and MTFE incur a cost of at most  $6i$  for the phase. On the other hand, TIMESTAMP incurs a cost of  $4i - 2$  for the phase. The aggregated cost of the three algorithms is at most  $10i - 2$  for the phase, while OPT incurs a cost of  $2i$ . So, the ratio between sum of the costs of the algorithms and OPT

is less than 5.

Next, consider phases of type 1 and form (c). **TIMESTAMP** incurs a cost of  $2k - 1$  in this case. Assume  $k$  is even, i.e., the phase has the form  $x^j(yx)^{2i}x$ . By Lemma 3.3, **MTFO** and **MTFE** each incur a total cost of  $3i$  for  $(yx)^{2i}$ . Moreover, after this, the list maintained for at least one of the algorithms is ordered as  $[xy]$ . Hence, the aggregated cost of algorithms for the next request to  $x$  is at most 1. Consequently, the total cost of **MTFE** and **MTFO** is at most  $6i + 1$  for the round. Adding the cost  $2k - 1 = 4i - 1$  of **TIMESTAMP**, the total cost of all three algorithms is at most  $10i$ . On the other hand, **OPT** incurs a cost of  $2i$  for the phase. So, the ratio between the aggregated cost of all three algorithms and the cost of **OPT** is at most 5. Finally, assume  $k$  is odd, i.e., the phase has form  $x^j(yx)^{2i-2}yxx$ . By Lemma 3.3, **MTFO** and **MTFE** together incur a total cost of  $2(3(i-1))$  for  $x^j(yx)^{2i-2}$ . By Lemma 3.4, they incur a total cost of at most 4 for  $yxx$ . In total, they incur a cost of at most  $6(i-1) + 4$  for the phase. **TIMESTAMP** incurs a cost of  $4i - 3$ ; this sums up to  $10i - 5$  for all three algorithms. In this case, **OPT** incurs a cost of  $2i - 1$ . Hence, the ratio between the sum of the costs of all three algorithms and **OPT** is at most 5.  $\square$

In fact, the upper bound provided in Theorem 3 for the competitive ratio of the best algorithm among **TIMESTAMP**, **MTFO** and **MTFE** is tight under the partial cost model. To show this, we make use of the following lemma.

**Lemma 3.5.** *Consider a sequence  $\sigma_\alpha = \langle x(yxxx\ yxxx)^k \rangle$ , i.e., a single request to  $x$ , followed by  $k$  repetitions of  $(yxxx\ yxxx)$ . Assume the list is initially ordered as  $[xy]$ . We have  $\text{MTFO}(\sigma) = \text{MTFE}(\sigma) = 4k$  while  $\text{OPT}(\sigma) = 2k$  (under the partial cost model).*

*Proof.* We refer to each repetition of  $(yxxx\ yxxx)$  as a round. Initially, the bits maintained by **MTFO** (resp. **MTFE**) for  $x, y$  are  $(1, 1)$  (resp.  $(0, 0)$ ). After the first request to  $x$ , the bits of **MTFO** (resp. **MTFE**) change to  $(0, 1)$  (resp.  $(1, 0)$ ) for  $x, y$ . **MTFO** incurs a cost of 3 for the first half of each round; it incurs a cost of 1 for all requests except the last request to  $x$ . **MTFE** incurs a cost of 1 for serving the first half of a round; it only incurs a cost of 1 on the first request  $y$ . After serving the first half, the list for each algorithm will be ordered as  $[xy]$  and the bits maintained by **MTFO** (resp. **MTFE**) for  $x, y$  will be  $(1, 0)$  (resp.  $(0, 1)$ ). Using a symmetric argument, the costs of **MTFO** and **MTFE** for the second half of a round are respectively 1 and 3. In total, both **MTFO** and **MTFE** incur a cost of 4 for each round. After serving the round,

the list maintained by both algorithms will be ordered as  $[xy]$  and the bits associated with the items will be the same as at the start of the first round. Thus, MTFO and MTFE each have a total cost of  $4k$  on  $\sigma_\alpha$ . A summary of actions and costs of MTFO and MTFE can be stated as follows (the numbers below the arrows indicate the costs of requests on top, and the numbers on top of  $x$  and  $y$  indicate their bits):

$$\begin{array}{cccccccccccccccc} \begin{array}{c} 01 \\ xy \end{array} & \xrightarrow[1]{y} & \begin{array}{c} 00 \\ yx \end{array} & \xrightarrow[1]{x} & \begin{array}{c} 01 \\ yx \end{array} & \xrightarrow[1]{x} & \begin{array}{c} 00 \\ xy \end{array} & \xrightarrow[0]{x} & \begin{array}{c} 10 \\ xy \end{array} & \xrightarrow[1]{y} & \begin{array}{c} 11 \\ xy \end{array} & \xrightarrow[0]{x} & \begin{array}{c} 01 \\ xy \end{array} & \xrightarrow[0]{x} & \begin{array}{c} 11 \\ xy \end{array} & \xrightarrow[0]{x} & \begin{array}{c} 01 \\ xy \end{array} \\ \begin{array}{c} 10 \\ xy \end{array} & \xrightarrow[1]{y} & \begin{array}{c} 11 \\ xy \end{array} & \xrightarrow[0]{x} & \begin{array}{c} 01 \\ xy \end{array} & \xrightarrow[0]{x} & \begin{array}{c} 11 \\ xy \end{array} & \xrightarrow[0]{x} & \begin{array}{c} 01 \\ xy \end{array} & \xrightarrow[1]{y} & \begin{array}{c} 00 \\ yx \end{array} & \xrightarrow[1]{x} & \begin{array}{c} 01 \\ yx \end{array} & \xrightarrow[1]{x} & \begin{array}{c} 00 \\ xy \end{array} & \xrightarrow[0]{x} & \begin{array}{c} 10 \\ xy \end{array} \end{array}$$

An optimal algorithm OPT never changes the ordering of the list and has a cost of 2 for the whole round, giving a cost of  $2k$  for  $\sigma_\alpha$ .  $\square$

**Theorem 3.6.** *There are sequences for which the costs of all of TIMESTAMP, MTFE, and MTFO are  $1.\bar{6}$  times that of OPT (under the partial cost model).*

*Proof.* Consider a sequence  $\sigma = \sigma_\alpha\sigma_\beta$  where  $\sigma_\alpha = x(yxxx\ yxxx)^{k_\alpha}$  and  $\sigma_\beta = (yyxx)^{k_\beta}$ . Here,  $k_\alpha$  is an arbitrary large integer and  $k_\beta = 2k_\alpha$ . By Lemma 3.5, we have  $\text{MTFO}(\sigma_\alpha) = \text{MTFE}(\sigma_\alpha) = 4k_\alpha$  while  $\text{OPT}(\sigma_\alpha) = 2k_\alpha$ . We have  $\text{TIMESTAMP}(\sigma_\alpha) = 2k_\alpha$ , because it does not move  $y$  from the second position.

Next, we study the cost of MTFO and MTFE for serving  $\sigma_\beta$ . Note that after serving  $\sigma_\alpha$ , the lists maintained by these algorithms is ordered as  $[xy]$  and the bits associated with  $x$  and  $y$  are respectively  $(0, 1)$  for MTFO and  $(1, 0)$  for MTFE (see the proof of Lemma 3.5). We show that for each round  $yyxx$  of  $\sigma_\beta$ , the cost of each of these two algorithms is 3. On the first request to  $y$ , MTFO moves it to the front (since the bit maintained for  $y$  is 1); so it incurs a cost of 1 for the first requests to  $y$ . On the first request to  $x$ , MTFO keeps  $x$  in the second position; hence it incurs a cost of 2 for the requests to  $x$ . In total, it has a cost of 3 for the round. With a similar argument, MTFE incurs a cost of 2 for the requests to  $y$  and a cost of 1 for the requests to  $x$  and a total cost of 3. The list order and bits maintained for the items will be the same at the end of the round as at the start. Hence, the same argument can be extended to other rounds to conclude that the cost of both MTFE and MTFO for serving  $\sigma_\beta$  is  $3k_\beta$ . On the other hand, TIMESTAMP incurs a cost of 4 on each round as it moves items to the front on the second consecutive request to them; hence, the cost of TIMESTAMP for serving  $\sigma_\beta$  is

$4k_\beta$ . An algorithm that moves items in front on the first of two consecutive requests to them will incur a cost of 2 on each round; hence the cost of OPT for serving  $\sigma_\beta$  is at most  $2k_\beta$ .

To summarize, the cost of each of MTFO and MTFE for serving  $\sigma$  is  $4k_\alpha + 3k_\beta = 10k_\alpha$  while the cost of TIMESTAMP is  $2k_\alpha + 4k_\beta = 10k_\alpha$ , and the cost of OPT is  $2k_\alpha + 2k_\beta = 6k_\alpha$ . As a consequence, all three algorithms have a cost which is  $10/6 = 1.\bar{6}$  times that of OPT.  $\square$

Before continuing with lower bounds, we compare the the results of Theorem 3.1 with the randomized algorithm COMB [4], which chooses to use BIT [27] with probability  $4/5$  and TIMESTAMP with probability  $1/5$ . BIT is the randomized algorithm which, for each item in the list, initially chooses randomly and independently with probability  $1/2$  whether that item should be moved to the front on odd or even accesses to it. BIT is 1.75-competitive [27]. COMB achieves a competitive ratio of 1.6, so for any request sequence  $I$ , either TIMESTAMP must achieve a performance ratio of 1.6 compared to OPT, or there must be some setting of the randomized bits for BIT which achieves a ratio of 1.6. This immediately gives an online algorithm with advice achieving the ratio 1.6 and using  $\ell + 1$  bits of advice, one bit to specify TIMESTAMP or BIT and  $\ell$  bits for BIT.

The phase partitioning in the proof of Theorem 3.1 uses more subcases than the proof for COMB, since it cannot assume independence of the times when  $x$  and  $y$  are moved to the front. It might be tempting to try to obtain an online algorithm with advice which achieves the ratio 1.6, as COMB does. The idea would be to create a randomized algorithm which uses MTFE and MTFO each with probability  $2/5$  and TIMESTAMP with probability  $1/5$ , and then change that to an algorithm using advice instead. The proof of Theorem 3.6 shows that this does not work, at least in the partial cost model. Consider the sequence  $\sigma_\alpha$ . Since both MTFE and MTFO have cost  $4k_\alpha$ , and TIMESTAMP and OPT both have cost  $2k_\alpha$ , the performance ratio one achieves with this weighting is 1.8.

On the other hand, note that MTFE and MTFO are equivalent to BIT in the cases where all the random bits are identical, either all zero or all one. Thus, using Theorem 3.1, one obtains a randomized algorithm with less randomness than COMB (choosing with equal probabilities between MTFE, MTFO, and TIMESTAMP) with a competitive ratio of  $1.\bar{6}$ . In addition, by Theorem 3.11 below, if only one bit of randomness is used for BIT, to decide whether MTFE or MTFO is used, the resulting algorithm is 2-competitive.

The lower bound proven on the competitiveness of this algorithm using only one bit of advice is 1.75.

It is clear in general that a  $c$ -competitive randomized algorithm using  $b(n)$  random bits for sequences of length  $n$  automatically gives a  $c$ -competitive algorithm with advice using  $b(n)$  bits of advice. Advice can be more powerful, though. It was shown in [12] that for minimization problems, for all  $\epsilon > 0$ , the existence of a  $c$ -competitive randomized algorithm implies the existence of a  $(1 + \epsilon)c$ -competitive algorithm using at most  $\lceil \log n \rceil + 2\lceil \log \lceil \log n \rceil \rceil + \log \left( \left\lfloor \frac{\log(m(n))}{\log(1+\epsilon)} \right\rfloor \right) + 1$  bits of advice, where  $m(n)$  is the number of possible inputs of length  $n$ .

The lower bound from Theorem 3.6 cannot easily be extended to the full cost model. In what follows, we provide, for the full cost model, a lower bound of 1.6 for the competitive ratio of the best algorithm among **TIMESTAMP**, **MTFE**, and **MTFO**. We start with the following lemma:

**Lemma 3.7.** *Consider a list of  $l$  items which is initially ordered as  $[a_1, a_2, \dots, a_l]$ . Consider the following sequence of requests with an  $m$ -fold repetition:*

$$\sigma_\beta = \langle (a_1, a_2, \dots, a_l, a_1^2, a_2^2, \dots, a_l^2, a_l, a_{l-1}, \dots, a_1, a_l^2, a_{l-1}^2, \dots, a_1^2)^m \rangle.$$

*Then for large  $l$ , we have  $\text{MTFO}(\sigma_\beta) = \text{MTFE}(\sigma_\beta) = m \cdot (3.5l^2 + o(l^2))$  while  $\text{TIMESTAMP}(\sigma_\beta) = m \cdot (2l^2 + o(l^2))$  (under the full cost model).*

*Proof.* Define a phase to be a subsequence of requests which forms one of the  $m$  repetitions in  $\sigma_\beta$ . We calculate the costs of the algorithms for each phase. Note that each phase contains an even number of requests to each item. Also, if  $i < j$ , so item  $a_i$  precedes item  $a_j$  in the initial ordering of the list, then, in each phase,  $a_i$  is requested twice after the last request to  $a_j$ . Each algorithm moves  $a_i$  in front of  $a_j$  on the first or second of these requests. Thus, the state of the list maintained by all algorithms is the same as with the initial ordering after serving a phase.

Each of the three algorithms incurs a cost of  $l(l+1)/2$  for serving  $a_1, a_2, \dots, a_l$  at the beginning of a phase. **MTFO** moves items to the front, reversing the list, but **MTFE** and **TIMESTAMP** do not move the items. For serving the subsequent requests to  $a_1^2, a_2^2, \dots, a_l^2$ , **MTFO** incurs a cost of  $2l^2$  since it does not move items to the front on the first of the two consecutive requests to an item, but on the second request. **MTFE** and **TIMESTAMP** move to the front at the first of the consecutive requests and incur a cost of  $l(l+1)/2 + l$  (the second request is to front of the list). At this point, for all three algorithms,



the list is in the reverse of the initial ordering since for  $i < j$  there have been two consecutive requests to  $a_j$  after the last request to  $a_i$ . Also, the bits maintained by MTFE and MTFO are flipped compared to the beginning of the phase (since there have been three requests to each item). Thus, for the second half of the list, MTFE and MTFO reverse roles. For the next requests to  $a_l, a_{l-1}, \dots, a_1$ , only MTFE reverses the list, and each of the three algorithms incurs a cost of  $l(l+1)/2$ . Consequently, for the remaining requests to  $a_l^2, a_{l-1}^2, \dots, a_1^2$ , MTFE incurs a cost of  $2l^2$ , while MTFO and TIMESTAMP each incur a cost of  $l(l+1)/2 + 2l$ .

To summarize, the costs of both MTFO and MTFE for each phase is  $3.5l^2 + o(l^2)$ , while the cost of TIMESTAMP is  $2l^2 + o(l^2)$ . The actions and costs of the algorithms can be summarized as following (as before, the numbers below arrows indicate the cost for serving the sequence on top, and the numbers on top of items indicate the bits maintained by MTFO and MTFE). The three lines correspond to MTFE, MTFO, and TIMESTAMP, respectively.

$$\begin{array}{l}
[a_1^0 \dots a_l^0] \xrightarrow[l^2/2+o(l^2)]{a_1 \dots a_l} [a_1^1 \dots a_l^1] \xrightarrow[l^2/2+o(l^2)]{a_1^2 \dots a_l^2} [a_1^1 \dots a_l^1] \xrightarrow[l^2/2+o(l^2)]{a_1 \dots a_1} [a_1^0 \dots a_l^0] \xrightarrow[2l^2+o(l^2)]{a_l^2 \dots a_1^2} [a_1^0 \dots a_l^0] \\
[a_1^1 \dots a_l^1] \xrightarrow[l^2/2+o(l^2)]{a_1 \dots a_l} [a_l^0 \dots a_1^0] \xrightarrow[2l^2+o(l^2)]{a_1^2 \dots a_l^2} [a_l^0 \dots a_1^0] \xrightarrow[l^2/2+o(l^2)]{a_l \dots a_1} [a_l^1 \dots a_1^1] \xrightarrow[l^2/2+o(l^2)]{a_l^2 \dots a_1^2} [a_1^1 \dots a_l^1] \\
[a_1 \dots a_l] \xrightarrow[l^2/2+o(l^2)]{a_1 \dots a_l} [a_1 \dots a_l] \xrightarrow[l^2/2+o(l^2)]{a_1^2 \dots a_l^2} [a_l \dots a_1] \xrightarrow[l^2/2+o(l^2)]{a_l \dots a_1} [a_l \dots a_1] \xrightarrow[l^2/2+o(l^2)]{a_l^2 \dots a_1^2} [a_1 \dots a_l]
\end{array}$$

□

The sequence  $\sigma_\beta$  of the above lemma shows that using one bit of advice to decide between using MTFE and MTFO gives a competitive ratio of at least 1.75, but TIMESTAMP serves  $\sigma_\beta$  optimally. Next, we introduce sequences for which TIMESTAMP performs significantly worse than both MTFO and MTFE.

**Lemma 3.8.** *Consider a list of  $l$  items which is initially ordered as  $[a_1, a_2, \dots, a_l]$ . Consider the following sequence of requests:*

$$\sigma_\gamma = \langle (a_l^3, a_2^3, \dots, a_1^3)^{2s} \rangle.$$

*Assuming that  $l$  is sufficiently large, we have  $\text{MTFO}(\sigma_\gamma) = \text{MTFE}(\sigma_\gamma) = s(3l^2 + o(l^2))$ , while  $\text{TIMESTAMP}(\sigma_\gamma) = s(4l^2 + o(l^2))$  and  $\text{OPT}(\sigma_\gamma) = s(2l^2 + o(l^2))$  (under the full cost model).*

*Proof.* Define a phase to be two consecutive repetitions of the subsequence in parentheses. We calculate the costs of the algorithms for each phase. Note that there are an even number of requests in each phase, and for  $i < j$ , there are (actually more than) two consecutive requests to  $a_i$  after the last request to  $a_j$ . So the list orderings and bits maintained by MTFO and MTFE are the same for each algorithm before and after serving each phase. Similarly, after serving the first half of a phase (the subsequence in parentheses), the lists of all three algorithms are the same as the initial ordering.

An optimal algorithm applies the MTF strategy and incurs a cost of  $2l^2 + 4l$ . More precisely, for serving each half of the phase, it incurs a cost of  $l^2$  for the first of three consecutive requests to each item, and a total cost of  $2l$  for the second and third requests. TIMESTAMP moves items to the front on the second of three consecutive requests. In each half of a phase, it incurs a total cost of  $2l^2$  for the first two requests to items and a cost of  $l$  for the third requests. In total, it incurs a cost of  $4l^2 + 2l$  for each phase. For the first half of the phase, MTFO moves items to front on the first request to each item, while MTFE does so on the second requests. Hence, MTFO and MTFE respectively incur a cost of  $l^2 + 2l$  and  $2l^2 + l$  for the first half. For the second half, the bits maintained by the algorithms are flipped, while the list ordering is the same as the initial ordering. Hence, MTFO and MTFE respectively incur a cost of  $2l^2 + l$  and  $l^2 + 2l$  for the second half. In total the costs of each of MTFO and MTFE for each phase is  $3l^2 + 3l$ . Since the cost of all algorithms are the same for all phases, the statement of the lemma follows. The actions and costs of the algorithms for each phase can be summarized as follows:

$$\begin{aligned}
\text{MTFE} &: [a_1^0 \dots a_l^0] \xrightarrow{a_l^3 \dots a_1^3} [a_1^1 \dots a_l^1] \xrightarrow{a_l^3 \dots a_1^3} [a_1^0 \dots a_l^0] \\
\text{MTFO} &: [a_1^1 \dots a_l^1] \xrightarrow{a_l^3 \dots a_1^3} [a_1^0 \dots a_l^0] \xrightarrow{a_l^3 \dots a_1^3} [a_1^1 \dots a_l^1] \\
\text{TIMESTAMP} &: [a_1 \dots a_l] \xrightarrow{a_l^3 \dots a_1^3} [a_1 \dots a_l] \xrightarrow{a_l^3 \dots a_1^3} [a_1 \dots a_l] \\
\text{OPT} &: [a_1 \dots a_l] \xrightarrow{a_l^3 \dots a_1^3} [a_1 \dots a_l] \xrightarrow{a_l^3 \dots a_1^3} [a_1 \dots a_l]
\end{aligned}$$

□

We use the above two lemmas to prove the following theorem. We remark

that for the partial cost model, a stronger result is proven in [6], which establishes that 1.6 is a lower bound for any projective algorithm.

**Theorem 3.9.** *The competitive ratio of the best algorithm among MTFE, MTFO, and TIMESTAMP is at least 1.6 under the full cost model.*

*Proof.* Consider the sequence  $\sigma = \sigma_\beta \sigma_\gamma$ , i.e., the concatenation of the sequences  $\sigma_\beta$  and  $\sigma_\gamma$  as defined in Lemmas 3.7 and 3.8. Recall that these sequences consist of  $m$  and  $s$  phases, respectively. In defining  $\sigma$ , consider values of  $s$  which are multiples of 3, and let  $m = 2s/3$ .

Assume the initial ordering is also the same as the one stated in the lemmas, and recall that the state of the algorithms (list ordering and bits of MTFE and MTFO) are the same at the end of serving  $\sigma_\beta$ . The costs of each of MTFE and MTFO for serving  $\sigma_\beta$  is  $3.5l^2m + o(l^2m) = \frac{7}{3}l^2s + o(l^2s)$  (by Lemma 3.7), while they incur a cost of  $3l^2s + o(l^2s)$  for  $\sigma_\gamma$  (by Lemma 3.8). In total, each of these two algorithms incurs a cost of  $\frac{16}{3}l^2s + o(l^2s)$  for  $\sigma$ . On the other hand, TIMESTAMP incurs a cost of  $2l^2m + o(l^2m) = \frac{4}{3}l^2s + o(l^2s)$  for  $\sigma_\beta$  and a cost of  $4l^2s + o(l^2s)$  for  $\sigma_\gamma$ . In total, its cost for  $\sigma$  is  $\frac{16}{3}l^2s + o(l^2s)$ . Note that all three algorithms have the same costs for serving  $\sigma$ . The cost of OPT for serving  $\sigma_\beta$  is at most  $2l^2m + o(l^2m) = \frac{4}{3}l^2s + o(l^2s)$  (by Lemma 3.7), while it has a cost of  $2l^2s + o(l^2s)$  for serving  $\sigma_\gamma$ . In total, the cost of OPT is  $\frac{10}{3}l^2s + o(l^2s)$ . Comparing this with the cost of  $\frac{16}{3}l^2s + o(l^2s)$  of the three algorithms, we conclude that the minimum competitive ratio is at least 1.6.  $\square$

Thus, the competitive ratio of the best of the three algorithms, MTFO, MTFE, and TIMESTAMP, is at least 1.6 and at most  $1.\bar{6}$ . We concluded after Lemma 3.7 that the competitive ratio of the better of MTFO and MTFE is at least 1.75. Here, we show that the competitive ratio of the better algorithm among MTFO and MTFE is at most 2, using the potential function method.

**Lemma 3.10.** *For any sequence  $\sigma$  of length  $n$ , we have*

$$\text{MTFO}(\sigma) + \text{MTFE}(\sigma) \leq 4 \text{OPT}(\sigma).$$

*Proof.* Consider an algorithm  $A \in \{\text{MTFO}, \text{MTFE}\}$ . At any time  $t$  (i.e., before serving the  $t$ th request), we say a pair  $(a, b)$  of items forms an *inversion* if  $a$  appears before  $b$  in the list maintained by  $A$  while  $b$  appears before  $a$  in the list maintained by OPT. We define the *weight* of an inversion  $(a, b)$  to be 1, if the bit maintained by  $A$  for  $b$  is 1, and 2 otherwise. Intuitively, the

weight of an inversion is the number of accesses to the latter of the two items in  $A$ 's list before the item is moved to the front and the inversion disappears.

We define the potential,  $\Phi_t$ , at each time  $t$  to be the total weight of the inversions in the list maintained by MTFO plus the total weight of the inversions in the list maintained by MTFE.

We consider the events that involve costs and change the potential function. An *online* event is the processing of a request by both MTFO and MTFE. An *offline* event is OPT making a paid exchange. The latter is not directly associated with a request, and we define the cost of MTFO and MTFE in connection with this event to be zero, but there may be a change in the potential function.

For an event at time  $t$ , we define the amortized cost  $a_t$  to be the total cost paid by MTFO and MTFE together for processing the request (if any), plus the increase in potential due to that processing, i.e.,  $a_t = \text{MTFO}_t + \text{MTFE}_t + \Phi_t - \Phi_{t-1}$ . So the total cost of MTFO and MTFE for serving a sequence  $\sigma$  is  $\sum_t a_t - (\Phi_{last} - \Phi_0)$ . The maximum possible value of  $\Phi_{last} - \Phi_0$  is independent of the length of the sequence. Hence, to prove the competitiveness of MTFO and MTFE together, it is enough to bound the amortized cost relative to OPT's cost. Let  $\text{OPT}_t$  be the cost paid by OPT at event  $t$ . To prove the lemma, it suffices to show that for each event, we have  $a_t \leq 4 \text{OPT}_t$ .

Note that one may assume that OPT only does paid exchanges, no free ones [26]. Consider MTFO and MTFE for an *online event*. Let  $A$  be the algorithm that moves  $y$  to the front, while  $A'$  is the other algorithm, i.e., the one that keeps it at its current position. Assume  $A$  accesses  $y$  at index  $k$  while  $A'$  finds it at index  $k'$ . Also, let  $j$  denote the index of  $y$  in the list maintained by OPT.

We first show that the contribution by  $A$  to the amortized cost is at most  $k - (k - j) + 2j = 3j$ . The first term ( $k$ ) is the access cost for  $A$ . Before moving  $y$  to front, there are at least  $k - j$  inversions with  $y$  for  $A$  involving items which occur before  $y$ , each having a weight of 1 (since the bit of  $y$  in  $A$  has been 1 as it moves  $y$  to front). All these inversions are removed after moving  $y$  to front. This gives the second term in the amortized cost, i.e.,  $-(k - j)$ . Moving  $y$  to the front creates at most  $j$  new inversions, each having a weight of at most 2, which results in a total increase of  $2j$  in the potential. Next, we show that the contribution by  $A'$  to the amortized cost is at most  $k' - (k' - j) = j$ . This is because, after accessing  $y$  at index  $k'$ , there are at least  $k' - j$  inversions with  $y$  for  $A'$  involving items which occur before

$y$ . Since  $A'$  does not move  $y$  to the front, the bit of  $y$  was 0, i.e., all these inversions had weight 2. After the access, the bit of  $y$  becomes 1 and the weights of these inversions decreases 1 unit. To summarize, the amortized cost  $a_t$  is at most  $3j + j = 4j$ . Since  $\text{OPT}$  accesses  $y$  at index  $j$ , we have  $\text{OPT}_j = j$  and consequently  $a_t \leq 4 \text{OPT}_t$ .

Next, consider an *offline event* where  $\text{OPT}$  makes a paid exchange. In doing so, it incurs a cost of 1 and  $\text{OPT}_t = 1$ . This single exchange might create an inversion in the list of MTFO and an inversion in the list of MTFE. Each of these inversions have a weight of at most 2. So, the total increase in the potential is at most 4, i.e.,  $a_t \leq 4$ . Consequently,  $a_t \leq 4 \text{OPT}_t$ .  $\square$

The above lemma implies that the better algorithm between MTFO and MTFE has a competitive ratio of at most 2. By Lemma 3.7, such an algorithm has a competitive ratio of at least 1.75.

**Theorem 3.11.** *The competitive ratio of the better algorithm between MTFO and MTFE is at least 1.75 and at most 2.*

#### 4. Analysis of Move-To-Front-Every-Other-Access

In the previous sections, we have used MTFE and MTFO to devise algorithms with better competitive ratios. These algorithms are Move-To-Front-Every-Other-Access algorithms (also called MTF2 algorithms). In this section, we study the competitive ratio of these algorithms. In [14, Exercise 1.5], it is stated that MTF2 is 2-competitive (throughout, by ‘MTF2’, we mean ‘an algorithm that belongs to the family of MTF2 algorithms’). The same statement is repeated in [8, 23]. It was first observed in [19] that MTF2 is in fact *not* 2-competitive. There, the author proves a lower bound of  $7/3$  for the competitive ratio of MTF2, and claims that an upper bound of 2.5 can be achieved. Here, we show that the competitive ratio of MTF2 is 2.5, and it is tight.

**Lemma 4.1.** *The competitive ratio of Move-To-Front-Every-Other-Access algorithms is at least 2.5 (under both partial and full cost models).*

*Proof.* We prove the lemma for MTFO and later extend it to other Move-To-Front-Every-Other-Access algorithms.

Consider a list of  $l$  items, initially ordered as  $[a_1, a_2, \dots, a_l]$ . Consider the following sequence of requests:

$$\sigma_\delta = \langle (a_1, a_2, \dots, a_l, a_1^3, a_2^3, \dots, a_l^3, a_l, a_{l-1}, \dots, a_1, a_l^3, a_{l-1}^3, \dots, a_1^3)^m \rangle.$$

We show that asymptotically, the cost of MTFO is 2.5 times the cost of OPT. Similar to our other lower bound proofs, we define a phase as a subsequence of requests which forms one of the  $m$  repetitions in  $\sigma_\delta$ . Note that each phase contains an even number of requests to each item. Also, if  $i < j$ , meaning that item  $a_i$  precedes item  $a_j$  in the initial ordering of the list, then, in each phase,  $a_i$  is requested three times after the last request to  $a_j$ . MTFO moves  $a_i$  in front of  $a_j$  due to these requests. Thus, the state of the list maintained by the algorithm is the same as the initial ordering after serving a phase.

Both MTFO and OPT incur a cost of  $l(l+1)/2$  for serving  $a_1, a_2, \dots, a_l$  at the beginning of a phase. MTFO moves items to the front and reverses the list, but OPT does not move the items. For serving the subsequent requests to  $a_1^3, a_2^3, \dots, a_l^3$ , MTFO incurs a cost of  $2l^2 + l$  since it moves items to the front on the second of the three consecutive requests to an item. OPT moves to the front at the first of the consecutive requests and incurs a cost of  $l(l+1)/2 + 2l$  (the second and third requests are to the front of the list). At this point, for both algorithms, the list is in the reverse of the initial ordering, while the bits maintained by MTFO are the same as in the beginning of the phase, since there have been four requests to each item, i.e., they are all 1.

For the next requests to  $a_l, a_{l-1}, \dots, a_1$ , MTFO reverses the list and incurs a cost of  $l(l+1)/2$ . OPT has the same cost and does not move the items. Consequently, for the remaining requests to  $a_l^3, a_{l-1}^3, \dots, a_1^3$ , MTFO incurs a cost of  $2l^2 + l$ , while OPT incurs a cost of  $l(l+1)/2 + 2l$ .

To summarize, in each phase, the cost of MTFO is  $5l^2 + o(l^2)$ , while the cost of OPT is  $2l^2 + o(l^2)$ . The actions and costs of the algorithms can be summarized as follows.

MTFO :

$$[a_1 \dots a_l] \xrightarrow[l^2/2+o(l^2)]{a_1 \dots a_l} [a_l \dots a_1] \xrightarrow[2l^2+o(l^2)]{a_1^3 \dots a_l^3} [a_l \dots a_1] \xrightarrow[l^2/2+o(l^2)]{a_l \dots a_1} [a_1 \dots a_l] \xrightarrow[2l^2+o(l^2)]{a_l^3 \dots a_1^3} [a_1 \dots a_l]$$

OPT :

$$[a_1 \dots a_l] \xrightarrow[l^2/2+o(l^2)]{a_1 \dots a_l} [a_1 \dots a_l] \xrightarrow[l^2/2+o(l^2)]{a_1^3 \dots a_l^3} [a_l \dots a_1] \xrightarrow[l^2/2+o(l^2)]{a_l \dots a_1} [a_l \dots a_1] \xrightarrow[l^2/2+o(l^2)]{a_l^3 \dots a_1^3} [a_1 \dots a_l]$$

We can extend the above lower bound to show that MTFE is at least 2.5-competitive. In doing so, consider the sequence  $\langle (a_1, a_2, \dots, a_l)\sigma_\delta \rangle$ . Note that after serving the subsequence in parentheses, all bits maintained by MTFE become 1, and the same analysis as above holds for serving  $\sigma_\delta$ . More

generally, for any initial setting of the bits maintained by a Move-To-Front-Every-Other-Access algorithm, we can start a sequence with a single request to each item having bit 0. After this subsequence, all bits are 1 and we can continue the sequence with  $\sigma_\delta$  to prove a lower bound of 2.5 for the competitive ratio of these algorithms. The starting subsequence adds an extra term of at most  $l^2/2 + o(l^2)$  to the costs of both OPT and the algorithm. This extra term can be ignored for sufficiently long sequences, i.e., when the value of  $m$  is asymptotically larger than  $l$ .  $\square$

As mentioned earlier, an upper bound of 2.5 for the competitive ratio of MTF2 was claimed earlier [19]. Here we include the proof for completeness since it does not appear to have ever been published.

**Lemma 4.2.** *The competitive ratio of any algorithm  $A$  which belongs to the Move-To-Front-Every-Other-Access family of algorithms is at most 2.5.*

*Proof.* We prove the statement for the partial cost model. Since  $A$  has the projective property and is cost-independent, the upper bound argument extends to the full cost model. Consider a sequence  $\sigma_{xy}$  of two items  $x$  and  $y$ . As before, we use the phase partitioning technique and partition  $\sigma_{xy}$  into phases as in the proof of Lemma 3.2. Recall that a phase ends with two consecutive requests to the same item in  $\sigma_{xy}$ . A phase has type 1 (respectively 2) if the relative order of  $x$  and  $y$  is  $[xy]$  (respectively  $[yx]$ ) at the beginning of the phase. Recall that a phase of type 1 has one of the following three forms ( $j \geq 0$  and  $k \geq 1$ ):

$$(a) x^j y y \quad (b) x^j (y x)^k y y \quad (c) x^j (y x)^k x$$

A phase of type 2 has exactly the same form as above with  $x$  and  $y$  interchanged. To prove the lemma, we show that its statement holds for every *two consecutive* phases. First, we consider each phase separately and show that the cost of MTF2 is at most 2 times that of OPT for all phases except a specific phase type that we call a *critical phase*. Table 4 shows the costs incurred by MTF2 and OPT for each phase. Note that phases of the form (b) and (c) are divided into two and three cases, respectively. The last row in the table corresponds to a critical phase. We discuss the different phases of type 1 separately. Similar analyses, with  $x$  and  $y$  interchanged, apply to the phases of type 2.

Note that before serving a phase of type 1, the list is ordered as  $[xy]$  and the first  $j$  requests to  $x$  have no cost. Consider phases of the form (a),  $x^j y y$ .

MTF2 incurs a total cost of at most 2 for serving  $yy$  and OPT incurs a cost of 1. So, the ratio between the costs of MTF2 and OPT is at most 2.

Next, consider phases of the form (b) with  $k = 2i$  ( $i$  is a positive integer). By Lemma 3.3, the cost incurred by MTF2 is at most  $3i$  for serving  $(yx)^{2i}$ . For the remaining two requests to  $y$ , MTF2 incurs a cost of at most 2. In total, the cost of MTF2 is at most  $3i + 2$  compared to  $2i + 1$  for OPT, and the ratio between them is less than 2.

Next, assume  $k$  is odd and  $k = 2i - 1$ , i.e., the phase has the form  $x^j(yx)^{2i-2}yxyy$ . The total cost of MTF2 for  $(yx)^{2i-2}$  is at most  $3(i - 1)$  (Lemma 3.3), and its cost for the next requests to  $xyyy$  is at most 4. In total, it incurs a cost of at most  $3i + 1$  for the phase, which is no more than twice the cost  $2i$  of OPT.

Next, consider phases of the form (c). Assume  $k$  is even, i.e., the phase has the form  $x^j(yx)^{2i}x$ . By Lemma 3.3, MTF2 incurs a cost of at most  $3i$  for  $(yx)^{2i}$  and a cost of at most 1 for the single request to  $x$ . The cost of the algorithm will be  $3i + 1$  compared to  $2i$  of OPT, and the ratio between them is no more than 2. Next, assume  $k$  is odd, i.e., the phase has the form  $x^j(yx)^{2i}yxx$  or  $x^jyxx$  (as before,  $i$  is a positive integer). In the first case, by Lemma 3.3, MTF2 incurs a cost of  $3i$  for  $x^j(yx)^{2i}$  and a cost of at most 3 for  $yxx$ . This sums to  $3i + 3$  while OPT incurs a cost of  $2i + 1$ ; the ratio between these two is no more than 2. If the phase has the form  $x^jyxx$ , we refer to it as a critical phase. The cost of MTF2 for such a phase can be as large as 3 while OPT incurs a cost of 1. However, we show that the cost of MTF2 in two consecutive phases is no more than twice the cost of OPT.

Consider two consecutive phases in  $\sigma_{xy}$ . If none of the phases are critical, the cost of MTF2 is at most twice that of OPT in both phases and we are done. Assume one of the phases is critical while the other phase is not. Let  $\text{OPT}_1$  and  $\text{OPT}_2$  denote the cost of OPT for the critical and non-critical phases, respectively. We have  $\text{OPT}_1 \leq \text{OPT}_2$  because OPT incurs a cost of 1 for critical phases and a cost of at least 1 for other phases (see Table 4). The cost of MTF2 for serving the two phases is at most  $3\text{OPT}_1 + 2\text{OPT}_2$  which is no more than  $2.5(\text{OPT}_1 + \text{OPT}_2)$  (since  $\text{OPT}_1 \leq \text{OPT}_2$ ). This implies that the cost of MTF2 is no more than 2.5 more than that of OPT for the two phases. Finally, assume both phases are critical. Thus, they form a subsequence  $(x^jyxx)(x^{j'}yxx)$  in  $\sigma_{xy}$ . MTF2 moves  $y$  to the front for exactly one of the two requests to  $y$ . Thus, it incurs a cost of 1 for one of the phases and a cost of at most 3 for the other phase. In total, its cost is no more than 4, while OPT incurs a cost of 2 for the two phases.  $\square$



Phase	MTF2	OPT'	ratio
$x^j yy$	$\leq 2$	1	$\leq 2$
$x^j (yx)^{2i} yy$	$\leq 3i + 2$	$2i + 1$	$< 2$
$x^j (yx)^{2i-2} yxyy$	$\leq 3(i-1) + 4$	$2i$	$\leq 2$
$x^j (yx)^{2i} x$	$\leq 3i + 1$	$2i$	$\leq 2$
$x^j (yx)^{2i} yxx$	$\leq 3i + 3$	$2i + 1$	$\leq 2$
$x^j yxx$	3	1	3

Table 4: The costs of MTF2 and OPT for a phase of type 1 (i.e., the initial ordering of items is  $xy$ ). The ratio between the cost of MTF2 and OPT for each phase, except the critical phase (the last row), is at most 2.

From Lemmas 4.1 and 4.2, we conclude the following theorem:

**Theorem 4.3.** *The competitive ratio of Move-To-Front-Every-Other-Access algorithms is 2.5.*

## 5. Concluding remarks

It is generally assumed that the offline oracle that generates advice bits has unbounded computational power. We used this assumption when we showed that  $\text{OPT}(\sigma)$  bits are sufficient to achieve an optimal solution in Section 2. However, for the algorithm introduced in Section 3, the advice bits can be generated in polynomial time. Table 5 provides a summary of the provided bounds for the competitive ratio of different algorithms.

The offline version of the list update problem is known to be NP-hard [5]. In this sense, our algorithm can be seen as a linear-time approximation algorithm with an approximation ratio of at most  $1.6$ ; this is, to the best of our knowledge, the best deterministic offline algorithm for the problem. As mentioned earlier, the randomized algorithm COMB [4], which is  $1.6$ -competitive, implies the existence of an online algorithm achieving a competitive ratio of at most  $1.6$  when provided a linear (in the length of the list) number of advice bits. However, from a practical point of view, it is not clear how an offline oracle can smartly generate such bits of advice. Moreover, our results (Theorem 4.3) indicate that, regardless of how the initial bits are generated, algorithm BIT has a competitive ratio of  $2.5$  against adaptive adversaries. This follows since an adaptive adversary can learn the original random bits from the behavior of BIT by requesting all items once, and then give requests to change 0-bits to 1-bits. This initial subsequence has constant length (proportional to the length of the list). After this, with a renaming of items based

Algorithm	Lower Bound		Upper Bound (both models)
	partial cost model	full cost model	
Best of MTFO, MTFE, and TIMESTAMP	1.6̄ (Theorem 3.6)	1.6 (Theorem 3.9)	1.6̄ (Theorem 3.1)
Better of MTFO and MTFE	2 (Lemma 3.5)	1.75 (Lemma 3.7)	2 (Lemma 3.10)
MTF2 (MTFO, MTFE, etc.)	2.5 (Lemma 4.1)	2.5 (Lemma 4.1)	2.5 (Lemma 4.2)

Table 5: Summary of proved lower and upper bounds for the competitive ratio of different algorithms.

on their current order in the list, the adversary can treat BIT as MTFO and give  $\sigma_\delta$  from the proof of Lemma 4.1.

We proved that the competitive ratio of the best algorithm among MTFE, MTFO, and TIMESTAMP is at least 1.6 and at most 1.6̄. Similarly, for the better algorithm between MTFE and MTFO the competitive ratio is between 1.75 and 2. It would be interesting to close these gaps.

## Acknowledgements

The authors would like to thank the referees for inquisitive and constructive comments. The first and third authors were supported in part by the Villum Foundation and the Danish Council for Independent Research, Natural Sciences.

## References

- [1] Albers, S., 1998. A competitive analysis of the list update problem with lookahead. *Theoret. Comput. Sci.* 197 (1–2), 95–109.
- [2] Albers, S., 1998. Improved randomized on-line algorithms for the list update problem. *SIAM J. Comput.* 27 (3), 682–693.
- [3] Albers, S., Mitzenmacher, M., 1998. Average case analyses of list update algorithms, with applications to data compression. *Algorithmica* 21 (3), 312–329.

- [4] Albers, S., von Stengel, B., Werchner, R., 1995. A combined BIT and TIMESTAMP algorithm for the list update problem. *Inform. Process. Lett.* 56, 135–139.
- [5] Ambühl, C., 2000. Offline list update is NP-hard. In: *Proc. 8th European Symp. on Algorithms (ESA)*. Vol. 1879 of *Lecture Notes in Comput. Sci.*, Springer. pp. 42–51.
- [6] Ambühl, C., Gärtner, B., von Stengel, B., 2013. Optimal lower bounds for projective list update algorithms. *ACM Trans. Algorithms* 9 (4), Article No. 31.
- [7] Angelopoulos, S., Dürr, C., Kamali, S., Renault, M. P., Rosén, A., 2015. Online bin packing with advice of small size. In: *Proc. 14th Algorithms and Data Structures Symp. (WADS)*. Vol. 9214 of *Lecture Notes in Comput. Sci.*, Springer. pp. 40–53.
- [8] Bachrach, R., El-Yaniv, R., Reinstadtler, M., 2002. On the competitive theory and practice of online list accessing algorithms. *Algorithmica* 32 (2), 201–245.
- [9] Bentley, J. L., Sleator, D., Tarjan, R. E., Wei, V. K., 1986. A locally adaptive data compression scheme. *Commun. ACM* 29 (4), 320–330.
- [10] Bianchi, M. P., Böckenhauer, H.-J., Hromkovič, J., Keller, L., 2012. Online coloring of bipartite graphs with and without advice. In: *Proc. 18th Computing and Combinatorics Conf. (COCOON)*. Vol. 7434 of *Lecture Notes in Comput. Sci.*, Springer. pp. 519–530.
- [11] Böckenhauer, H.-J., Hromkovič, J., Komm, D., Krug, S., Smula, J., Sprock, A., 2013. The string guessing problem as a method to prove lower bounds on the advice complexity. In: *Proc. 19th Computing and Combinatorics Conf. (COCOON)*. Vol. 7936 of *Lecture Notes in Comput. Sci.*, Springer. pp. 493–505.
- [12] Böckenhauer, H.-J., Komm, D., Kráľovič, R., Kráľovič, R., 2011. On the advice complexity of the  $k$ -server problem. In: *Proc. 38th International Colloquium on Automata, Languages, and Programming (ICALP)*. Vol. 6755 of *Lecture Notes in Comput. Sci.*, Springer. pp. 207–218.

- [13] Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R., Mömke, T., 2009. On the advice complexity of online problems. In: Proc. 20th International Symp. on Algorithms and Computation (ISAAC). Vol. 5878 of Lecture Notes in Comput. Sci., Springer. pp. 331–340.
- [14] Borodin, A., El-Yaniv, R., 1998. Online Computation and Competitive Analysis. Cambridge University Press.
- [15] Boyar, J., Kamali, S., Larsen, K. S., López-Ortiz, A., 2014. Online bin packing with advice. In: Proc. 31st Symp. on Theoretical Aspects of Computer Science (STACS). pp. 174–186.
- [16] Dobrev, S., Královič, R., Pardubská, D., 2008. How much information about the future is needed? In: Proc. 34th International Conf. on Current Trends in Theory and Practice of Computer Science (SOFSEM). Vol. 4910 of Lecture Notes in Comput. Sci., Springer. pp. 247–258.
- [17] Emek, Y., Fraigniaud, P., Korman, A., Rosén, A., 2011. Online computation with advice. Theoret. Comput. Sci. 412 (24), 2642 – 2656.
- [18] Forišek, M., Keller, L., Steinová, M., 2012. Advice complexity of online coloring for paths. In: Proc. 6th International Conf. on Language and Automata Theory and Applications (LATA). Vol. 7183 of Lecture Notes in Comput. Sci., Springer. pp. 228–239.
- [19] Grüne, A., 2003. MTF2 is not 2-competitive, unpublished manuscript.
- [20] Gupta, S., Kamali, S., López-Ortiz, A., 2013. On advice complexity of the k-server problem under sparse metrics. In: Proc. 20th International Colloquium on Structural Information and Communication Complexity (SIROCCO). pp. 55–67.
- [21] Hromkovič, J., Královič, R., Královič, R., 2010. Information complexity of online problems. In: Proc. 35th Symp. on Mathematical Foundations of Computer Science (MFCS). Vol. 6281 of Lecture Notes in Comput. Sci., Springer. pp. 24–36.
- [22] Irani, S., 1991. Two results on the list update problem. Inform. Process. Lett. 38, 301–306.

- [23] Kamali, S., López-Ortiz, A., 2013. A survey of algorithms and models for list update. In: *Space-Efficient Data Structures, Streams, and Algorithms*. Vol. 8066 of *Lecture Notes in Comput. Sci.*, Springer. pp. 251–266.
- [24] Kamali, S., López-Ortiz, A., 2014. Better compression through better list update algorithms. In: *Proc. 23rd Data Compression Conf. (DCC)*. pp. 372–381.
- [25] Komm, D., Královič, R., 2011. Advice complexity and barely random algorithms. *RAIRO Inform. Theor. Appl.* 45 (2), 249–267.
- [26] Reingold, N., Westbrook, J., 1996. Off-line algorithms for the list update problem. *Inform. Process. Lett.* 60 (2), 75–80.
- [27] Reingold, N., Westbrook, J., Sleator, D. D., 1994. Randomized competitive algorithms for the list update problem. *Algorithmica* 11, 15–32.
- [28] Renault, M. P., Rosén, A., 2011. On online algorithms with advice for the  $k$ -server problem. In: *Proc. 9th International Workshop in Approximation and Online Algorithms (WAOA)*. Vol. 7164 of *Lecture Notes in Comput. Sci.*, Springer. pp. 198–210.
- [29] Seibert, S., Sprock, A., Unger, W., 2013. Advice complexity of the online coloring problem. In: *Proc. 8th International Conf. on Algorithms and Complexity (CIAC)*. Vol. 7878 of *Lecture Notes in Comput. Sci.*, Springer. pp. 345–357.
- [30] Sleator, D., Tarjan, R. E., 1985. Amortized efficiency of list update and paging rules. *Commun. ACM* 28, 202–208.