# On the List Update Problem with Advice

Joan Boyar[1,⋆], Shahin Kamali[2,⋆⋆], Kim S. Larsen[1,⋆], and Alejandro López-Ortiz[2]

[1] University of Southern Denmark
Department of Mathematics and Computer Science
Campusvej 55, 5230 Odense M, Denmark
{joan,kslarsen}@imada.sdu.dk
[2] University of Waterloo, School of Computer Science
200 University Avenue West, Waterloo, ON N2L 3G1, Canada
{s3kamali,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 is 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.

## 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[1]. 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.

⋆ Supported in part by the Danish Council for Independent Research and the Villum Foundation.
⋆⋆ Supported in part by Dr. Derick Wood Graduate Scholarship in Computer Science.

[1] Similar to other works, we consider the *static* list update problem in which there is no insertion or deletion.

Similar to other online problems, the standard method for comparing online list update algorithms is competitive analysis. The competitive ratio of an online algorithm $\mathbb{A}$ is the maximum ratio between the cost of $\mathbb{A}$ for serving any sequence and the cost of OPT for serving the same sequence. Here, 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 [13]); this converges to 2 for large lists. There are 2-competitive (hence optimal) algorithms for the problem; these include Move-To-Front (MTF) [20] and TIMESTAMP [1].

Although competitive analysis has been accepted as the standard tool for comparing online algorithms, there are objections to it. One relevant objection in this context 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 [4]. 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 $\mathbb{A}$ by writing the access cost of $\mathbb{A}$ for serving each character in unary[2]. 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 compared to the subsequent savings in the size of the compressed file compared with the original file, due to the availability of the partial information.

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 [10,11,7,6]. Here, we use a natural model from [7,6] that assumes advice bits are written on a tape, and the online algorithm can access the tape at any time. The advice complexity of an algorithm is then the length of the shortest prefix of the tape that includes all accessed bits. Since its introduction, many online problems have been studied under the advice model. These include classical online problems such as paging [7,12,15], $k$-server [11,6,19], and bin packing [9].

## 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 OPT($\sigma$) is sufficient to optimally serve a sequence $\sigma$, where 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

[2] Encodings other than unary correspond to other cost models for list update.

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 $1.\bar{6}$. 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 $x$.

## 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 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* [16]. In a subset transfer, after 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. After 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 to after $x$. Provided with this, the algorithm can always maintain the same list as Opt. The total number of bits read by the algorithm will be equal to $\text{Opt}(\sigma) - n$. □

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.

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 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 $yyyxx$, and for a round of type 1, the requests $yxxxx$. For example, if $B = 011\ldots$, we will have $\sigma = \langle yyyxx, yxxxx, yxxxx, \ldots \rangle$.

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, which is easy to prove.

**Lemma 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 $[x, y]$.*

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 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. □

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 $yxxxx$), 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 guessing problem, defined in [11,5].

**Definition 4 ([5]).** *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 5 ([5]).** *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.* [3]

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

[3] In this paper we use $\log n$ to denote $\log_2(n)$.

**Theorem 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 read at least $(1 + (7\gamma - 7)\log(7\gamma - 7) + (8 - 7\gamma)\log(8 - 7\gamma))/5 \times n$ 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., $\mathbb{A} \geq \textsc{Opt} + w$, where $w$ is the number of wrong guesses for critical requests. Since $\mathbb{A}$ has a competitive ratio of $\gamma$, we have $\mathbb{A} \leq \gamma\,\textsc{Opt}$. Consequently, we have $w \leq (\gamma - 1)\,\textsc{Opt}(\sigma)$ and by Lemma 3, $w \leq 7(\gamma - 1)/5 \times n$. This implies that if $\mathbb{A}$ has a competitive ratio of $\gamma$, the 2-SGKH algorithm makes at most $7(\gamma - 1)/5 \times n$ mistakes for an input bitstring $B$ of size $n/5$, i.e., at least $n/5 - 7(\gamma - 1)/5 \times n = (8 - 7\gamma) \times n/5$ 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 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. □

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 1 and 6 imply the following corollary.

**Corollary 7.** *For any fixed list, $\Theta(n)$ 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 8.** *For any sequence $\sigma$, we have either* $\textsc{Timestamp}(\sigma) \leq 1.\bar{6}\,\textsc{Opt}(\sigma)$, $\textsc{MtfO}(\sigma) \leq 1.\bar{6}\,\textsc{Opt}(\sigma)$, *or* $\textsc{MtfE}(\sigma) \leq 1.\bar{6}\,\textsc{Opt}(\sigma)$.

To prove the theorem, we show that for any sequence $\sigma$, $\textsc{Timestamp}(\sigma) + \textsc{MtfO}(\sigma) + \textsc{MtfE}(\sigma) \leq 5\,\textsc{Opt}(\sigma)$. 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 analysis of projective algorithms,[4] 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 ensures 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 copy of $\sigma$ in which all items except $x$ and $y$ are removed. We refer the reader to [8, p. 16] for details on the factoring lemma. Since MtfO, MtfE, and Timestamp are projective and cost-independent, to prove Theorem 8, it suffices to prove the following lemma:

**Lemma 9.** *Under the partial cost model, for any sequence $\sigma_{xy}$ of two items, we have* $\text{MtfO}(\sigma_{xy}) + \text{MtfE}(\sigma_{xy}) + \text{Timestamp}(\sigma_{xy}) \leq 5 \times \text{Opt}(\sigma_{xy})$.

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 following request is to the same item. It is known that this algorithm is optimal for lists of two items [17].

[4] Almost all existing algorithms for the list update problem are projective; the only exceptions are Transpose, Move-Fraction [20], and Split [13]; see [14] for a survey.

**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 at most 3 (under the partial cost model). The final ordering of the items will be $[ab]$ in three of the cases.

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

**Lemma 10.** *Consider a subsequence of two items $a$ and $b$ of the form $(ba)^{2i}$, 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 at most $3i$ (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 repetition of *baba* as a *round.* We show that MtfO and MtfE have a cost of at most 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 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. □

**Lemma 11.** *Consider a subsequence of two items $a$ and $b$ which has form $\langle baa \rangle$. The total cost that* MtfE *and* MtfO *incur together for serving this subsequence is less than or equal to 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. □

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

| 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) | $1+0+0=1$ | $[a\overset{\downarrow}{b}]$ $[a\overset{\downarrow}{b}]$ $[a\overset{\downarrow}{b}]$ | $(1,1) \rightarrow 2$ | $1+2=3$ |
| $[ab]$ | (0,1) | $1+1+1=3$ | $[\overset{\curvearrowleft}{ab}]$ $[b\overset{\downarrow}{a}]$ $[\overset{\curvearrowleft}{ba}]$ | $(1,0) \rightarrow 1$ | $3+1=4$ |
| $[ab]$ | (1,0) | $1+0+0=1$ | $[a\overset{\downarrow}{b}]$ $[a\overset{\downarrow}{b}]$ $[a\overset{\downarrow}{b}]$ | $(0,1) \rightarrow 3$ | $1+3=4$ |
| $[ab]$ | (1,1) | $1+1+0=2$ | $[\overset{\curvearrowleft}{ab}]$ $[\overset{\curvearrowleft}{ba}]$ $[a\overset{\downarrow}{b}]$ | $(0,0) \rightarrow 1$ | $2+1=3$ |
| $[ba]$ | (0,0) (0,1) (1,0) (1,1) | $\leq 0+1+1=2$ | - | $\leq 2$ | $2+2=4$ |

Using Lemmas 10 and 11, we are ready to prove Lemma 9:

*Proof (Lemma 9, and consequently Theorem 8).*

Consider a sequence $\sigma_{xy}$ of two items $x$ and $y$. We use the *phase partitioning technique* as discussed in [8]. 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 ($t \geq 1$) 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, MtfO 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), $x^j yy$. MtfO 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 $2k$ for serving the phase; it incurs a cost of 1 for all requests in $(yx)^{2i}$ except the very first one, 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 $x^j(yx)^k yy$. By Lemma 10, the cost incurred by MtfO and MtfE is at most $3i$ for serving $(yx)^{2i}$. We show that for the remaining two requests to $y$, MtfO and MtfO 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 at most $2 \times (3(i-1))$ (Lemma 10), the total cost for the next request to $y$ is at most 2, and the total cost for subsequent $xyy$ is at most 4 (Lemma 11). 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 10, MtfO and MtfE each incur a total cost of at most $3i$ 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 10, MtfO and MtfE together incur a total cost of $2 \times 3(i-1)$ for $x^j(yx)^{2i-2}$. By Lemma 11, 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.

In fact, the upper bound provided in Theorem 3 for the competitive ratio of the better algorithm among Timestamp, MtfO and MtfE is tight under the partial cost model. To show this, we make use of the following lemma.

**Lemma 12.** *Consider a sequence $\sigma_\alpha = x(yxxx\ yxxx)^k$, i.e., a single request to $x$, followed by $k$ repetitions of $(yxxx\ yxxx)$. Assume the list is initially ordered as $[xy]$. We have* MtfO$(\sigma)$ = MtfE$(\sigma) = 4k$ *while* Opt$(\sigma) = 2k$ *(under the partial cost model).*

**Table 3.** 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.

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

*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 requests $y$. 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 $\sigma_\alpha$. 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 $\sigma_\alpha$. □

**Theorem 13.** *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 12, 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 12).We show that for each round $yyxx$ of $\sigma_\beta$, the cost of each algorithm 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 request 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. □

## 4 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 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. The offline version of the list update problem is known to be NP-hard [3]. In this sense, our algorithm can be seen as a linear-time approximation algorithm with an approximation ratio of $1.\bar{6}$; this is, to the best of our knowledge, the best deterministic offline algorithm for the problem. It should be mentioned that there is a randomized online algorithm Bit which also has a competitive ratio of $1.\bar{6}$ against an oblivious adversary [18]. Bit maintains a bit for each item and flips the bit on each access; whenever the bit becomes '0' it moves the item to the front. The bits are initially set uniformly at random; hence, Bit uses $l$ bits of advice for lists of length $l$. Comb is another randomized algorithm which makes use of a linear number of random bits and improves the competitive ratio to 1.6 [2]. We can conclude that there are online algorithms which achieve 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.

We proved that with two bits of advice, one can achieve a (deterministic) algorithm with a competitive ratio of at most $1.\bar{6}$. This bound is tight under the partial cost model (Theorem 13); however, the lower bound argument for the competitive ratio of this algorithm does not extend to the full cost mode, i.e., the upper bound of $1.\bar{6}$ might be overly pessimistic. All studied projective algorithms have the same competitive ratio under partial and full cost models; our algorithm might be distinctive in this sense.

While two bits of advice can break the lower bound of 2 on the competitive ratio of online algorithms, it remains open whether this can be done with one bit of advice. Regardless, it is not hard to see that any algorithm with one bit of advice has a competitive ratio of at least 1.5. We conjecture that this lower bound can be improved and leave it as future work.

## References

1. Albers, S.: Improved randomized on-line algorithms for the list update problem. SIAM J. Comput. 27(3), 682–693 (1998)
2. Albers, S., von Stengel, B., Werchner, R.: A combined BIT and TIMESTAMP algorithm for the list update problem. Inf. Proc. Lett. 56, 135–139 (1995)
3. Ambühl, C.: Offline list update is NP-hard. In: Paterson, M. (ed.) ESA 2000. LNCS, vol. 1879, pp. 42–51. Springer, Heidelberg (2000)
4. Bentley, J.L., Sleator, D., Tarjan, R.E., Wei, V.K.: A locally adaptive data compression scheme. Commun. ACM 29, 320–330 (1986)
5. Böckenhauer, H.-J., Hromkovič, J., Komm, D., Krug, S., Smula, J., Sprock, A.: The string guessing problem as a method to prove lower bounds on the advice complexity. In: Du, D.-Z., Zhang, G. (eds.) COCOON 2013. LNCS, vol. 7936, pp. 493–505. Springer, Heidelberg (2013)
6. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R.: On the advice complexity of the $k$-server problem. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 207–218. Springer, Heidelberg (2011)
7. Böckenhauer, H., Komm, D., Královič, R., Královič, R., Mömke, T.: On the advice complexity of online problems. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 331–340. Springer, Heidelberg (2009)
8. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press (1998)
9. Boyar, J., Kamali, S., Larsen, K.S., López-Ortiz, A.: Online bin packing with advice. CoRR abs/1212.4016 (2012)
10. Dobrev, S., Královič, R., Pardubská, D.: Measuring the problem relevant information in input. RAIRO Inform. Theor. Appl. 43(3), 585–613 (2009)
11. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. Theoret. Comput. Sci. 412(24), 2642–2656 (2011)
12. Hromkovič, J., Královič, R., Královič, R.: Information complexity of online problems. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 24–36. Springer, Heidelberg (2010)
13. Irani, S.: Two results on the list update problem. Inf. Proc. Lett. 38, 301–306 (1991)
14. Kamali, S., López-Ortiz, A.: A survey of algorithms and models for list update. In: Brodnik, A., López-Ortiz, A., Raman, V., Viola, A. (eds.) Ianfest-66. LNCS, vol. 8066, pp. 251–266. Springer, Heidelberg (2013)
15. Komm, D., Královič, R.: Advice complexity and barely random algorithms. RAIRO Inform. Theor. Appl. 45(2), 249–267 (2011)
16. Reingold, N., Westbrook, J.: Optimum off-line algorithms for the list update problem. Tech. Rep. YALEU/DCS/TR-805, Yale University (1990)
17. Reingold, N., Westbrook, J.: Off-line algorithms for the list update problem. Inf. Proc. Lett. 60(2), 75–80 (1996)
18. Reingold, N., Westbrook, J., Sleator, D.D.: Randomized competitive algorithms for the list update problem. Algorithmica 11, 15–32 (1994)
19. Renault, M.P., Rosén, A.: On online algorithms with advice for the $k$-server problem. In: Solis-Oba, R., Persiano, G. (eds.) WAOA 2011. LNCS, vol. 7164, pp. 198–210. Springer, Heidelberg (2012)
20. Sleator, D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM 28, 202–208 (1985)