New Bounds for Randomized List Update in the Paid Exchange Model

Susanne Albers
Department of Computer Science, Technical University of Munich, Germany
albers@in.tum.de

Maximilian Janke
Department of Computer Science, Technical University of Munich, Germany
maximilian.janke@in.tum.de

Abstract

We study the fundamental list update problem in the paid exchange model P^d. This cost model was introduced by Manasse, McGeoch and Sleator [18] and Reingold, Westbrook and Sleator [24]. Here the given list of items may only be rearranged using paid exchanges; each swap of two adjacent items in the list incurs a cost of d. Free exchanges of items are not allowed. The model is motivated by the fact that, when executing search operations on a data structure, key comparisons are less expensive than item swaps.

We develop a new randomized online algorithm that achieves an improved competitive ratio against oblivious adversaries. For large d, the competitiveness tends to 2.2442. Technically, the analysis of the algorithm relies on a new approach of partitioning request sequences and charging expected cost. Furthermore, we devise lower bounds on the competitiveness of randomized algorithms against oblivious adversaries. No such lower bounds were known before. Specifically, we prove that no randomized online algorithm can achieve a competitive ratio smaller than 2 in the partial cost model, where an access to the i-th item in the current list incurs a cost of $i - 1$ rather than i. All algorithms proposed in the literature attain their competitiveness in the partial cost model. Furthermore, we show that no randomized online algorithm can achieve a competitive ratio smaller than 1.8654 in the standard full cost model. Again the lower bounds hold for large d.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases self-organizing lists, online algorithm, competitive analysis, lower bound

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.12

Funding Work supported by the European Research Council, Grant Agreement No. 691672, project APEG.

1 Introduction

In this paper we revisit the *list update problem*, one of the most basic problems in the theory of online algorithms [7, 26]. The goal is to maintain an unsorted linear linked list of items so that a sequence of accesses to these items can be served with low total cost. Unsorted linear lists are sensible when one has to store a small dictionary consisting of a few dozen items. Moreover, they have interesting applications in data compression. In fact, the standard compression program `bzip2` relies on a combination of the Burrows-Wheeler transform and linear list encoding [9, 10, 19, 25].

Early work on the list update problem dates back to the 1960s [21]. Over the past decades an extensive body of literature has been developed, see e.g. [1, 2, 5, 6, 7, 8, 11, 12, 14, 24, 26] and references therein. List update in the *standard model* is well understood. In this setting the cost of an access is equal to the depth of the referenced item in the current list. Immediately after an access the requested item may be moved at no extra cost to any position closer to the front of the list (free exchanges). Any other swap of two adjacent items

in the list incurs a cost of 1 and is called a *paid exchange*. During the last years, research on the list update problem has explored (1) alternative cost models [13, 15, 20, 22], (2) refined input models capturing locality of reference [2, 6, 11], and (3) the value of algorithmic service abilities [8, 15, 17].

We investigate the list update problem in the P^d *model*, i.e. the paid exchange model, introduced by Manasse, McGeoch and Sleator [18] as well as Reingold, Westbrook and Sleator [24]. In this model there are no free exchanges and each paid exchange, swapping a pair of adjacent items in the list, incurs a cost of d, where $d \geq 1$ is a real-valued constant. The model is motivated by the fact that the execution time of a program swapping a pair of adjacent items in the list is typically much higher than that of the program doing one iteration of the search loop. Moreover, bringing a referenced element closer to the front of the list does incur cost. As main result we develop nearly tight upper and lower bounds on the competitive ratio achieved by randomized online algorithms.

Problem formulation

In the list update problem we are given an unsorted linear linked list L of n items. An algorithm is presented with a sequence $\sigma = \sigma(1), \ldots, \sigma(m)$ of requests that must be served in the order of occurrence. Each request $\sigma(t)$, $1 \leq t \leq m$, specifies an item in the list. In order to serve a request, an algorithm has to access the requested item, i.e. it has to start at the front of the list and search linearly through the items until the referenced item is found. Hence an access to the i-th item in the list incurs a cost of i. In the *standard model*, immediately after a request, the referenced item may be moved at no extra cost to any position closer to the front of the list. These exchanges are called free exchanges. Moreover, at any time, two adjacent items in the list may be swapped at a cost of 1. Such exchanges are called paid exchanges.

In contrast, in the *paid exchange model* P^d, there are no free exchanges. The list can only be rearranged using paid exchanges. Each paid exchange incurs a cost of d, where $d \geq 1$ is an arbitrary, real-valued constant. Following Reingold, Westbrook and Sleator [24] we assume that the service of a request $\sigma(t)$, $1 \leq t \leq m$, proceeds as follows: First an algorithm performs a number of paid exchanges; then the item referenced by $\sigma(t)$ is accessed. In general, in both the standard and the P^d model, the goal is to serve a request sequence so that the total cost is as small as possible.

An online algorithm has to serve each request without knowledge of any future requests. Given a request sequence σ, let $C_A(\sigma)$ and $C_{\mathrm{OPT}}(\sigma)$ denote the costs incurred by an online algorithm A and an optimal offline algorithm OPT in serving σ. A deterministic online algorithm A is called *c-competitive* if there exists an α such that $C_A(\sigma) \leq c \cdot C_{\mathrm{OPT}}(\sigma) + \alpha$ holds for all request sequences σ. The value α must be independent of the input σ but may depend on the list length n. A randomized online algorithm A is *c-competitive against oblivious adversaries* if there exists an α such that $\mathbf{E}[C_A(\sigma)] \leq c \cdot C_{\mathrm{OPT}}(\sigma) + \alpha$ holds for all σ. Here the expectation is taken over the random choices made by A.

Previous work

Due to the wealth of results on the list update problem, we only mention the most important contributions relevant to our work. First we focus on the standard model. In their seminal paper [26], Sleator and Tarjan showed that the deterministic MOVE-TO-FRONT algorithm is 2-competitive. This is the best competitiveness a deterministic online strategy can attain [16]. Subsequent research developed randomized online algorithms against oblivious adversaries.

Irani [12] gave a SPLIT algorithm that is 1.9375-competitive. Reingold et al. [24] devised a family of COUNTER algorithms and showed that the best of these achieve a competitive ratio of $85/49 \approx 1.7346$. In the same paper a generalized RANDOM RESET algorithm attains a competitive ratio of $\sqrt{3} \approx 1.7321$. A family of TIMESTAMP algorithms was developed in [1]. They attain a competitiveness equal to the Golden Ratio $(1+\sqrt{5})/2 \approx 1.6180$. The best randomized algorithm currently known is 1.6-competitive [3]. The algorithm is a combination of specific instances of the COUNTER and TIMESTAMP families.

As for lower bounds, Teia [27] showed that no randomized online algorithm can be better than 1.5-competitive against oblivious adversaries. Ambühl et al. [5] showed that no projective randomized online algorithm can be better than 1.6-competitive against oblivious adversaries in the partial cost model. In the *partial cost model* an access to the i-th item in the list incurs a cost of $i-1$ rather than i. Moreover, an algorithm is projective if it suffices to consider pairs of items. Specifically, the relative order of any two items in the list only depends on the previous requests to those elements. All the algorithms mentioned above, except for SPLIT, are projective.

Recent research on the standard model has proposed models capturing locality of reference in request sequences [2, 6, 11]. Furthermore, Lopez-Ortiz et al. [17] analyzed the value of paid exchanges. They showed a lower bound of 12/11 on the worst-case ratio between the performance of an offline algorithm that uses only free exchanges and that of an offline algorithm that uses both paid and free exchanges. The optimal offline algorithm does not need to use free exchanges [23]. Boyar et al. [8] analyzed the list update problem with advice, where an online algorithm has partial information on future requests.

We next consider the P^d model. So far only COUNTER and RANDOM RESET algorithms have been studied. The best deterministic online algorithm currently known achieves a competitive ratio of $(5+\sqrt{17})/2 \approx 4.5616$ [4]. No deterministic online algorithm can be better than 3-competitive [24]. Reingold et al. [24] gave a randomized COUNTER algorithm. For $d=1$, the algorithm is 2.75-competitive against oblivious adversaries. For increasing d, the competitiveness decreases and tends to $(5+\sqrt{17})/4 \approx 2.2808$. For small values of d, their RANDOM RESET algorithm achieves competitive ratios that are slightly smaller than those of the COUNTER algorithm. No lower bounds on the performance of randomized online algorithms against oblivious adversaries are known.

As for other cost models, Munro [22] and Kamali et al. [13] proposed settings that allow rearranging large parts of a list at lower costs. Specifically, after an access to the i-th item in the list, all items up to position i can be rearranged at a cost proportional to i. In an even stronger model of Kamali et al. [13], which is interesting in data compression applications, the whole list can be rearranged free of charge, while accessing an item in the ith position only incurs cost $\Theta(\lg i)$.

Our contribution

We present a comprehensive study of the list update problem in the P^d model. First in Section 3 we develop a new randomized online algorithm TIMESTAMP(l,p), which is defined for any positive integer l and any probability p, $0 \le p \le 1$. The strategy incorporates features of the TIMESTAMP algorithm in the standard model and the COUNTER algorithm in the P^d model. In the P^d model one cannot afford to move a referenced item closer to the front of the list on every request to that item. Therefore, for every item in the list, TIMESTAMP(l,p) maintains a mod l counter. These counters are initialized independently and uniformly at random. When an item is requested, its counter value is decremented by 1. If the counter switches from 0 to $l-1$, we say that a *master request* to that item occurs. On a master

request to x, with probability p, item x is moved to the front of the list. Otherwise, with probability $1-p$, item x is inserted in front of the first item y in the list such that (a) at most one master request to y occurred since the last master request to x and (b) the possible master request to y was also handled according to this latter policy, i.e. y was not moved to the front.

TIMESTAMP(l,p) achieves an improved competitive ratio of $c < 2.2442$ against oblivious adversaries, as d grows. A main contribution of this paper is a new analysis technique. TIMESTAMP(l,p) is projective so that it can be analyzed on pairs of items. However in the P^d model, in contrast to the standard model, it is hard to keep track of the optimal offline algorithm. Specifically, it does not hold true anymore that after two consecutive requests to the same item, that item precedes the other item in the optimum list. Therefore we define a more general phase partitioning of request sequences. A phase ends whenever the optimum offline algorithm OPT changes the relative order of the two considered items in the list. Hence the analysis is guided by OPT's moves. In particular, each phase can be assigned the cost of one paid exchange performed by OPT. The challenge is to estimate the cost of TIMESTAMP(l,p) without making any assumptions on the request pattern at the end of a phase. In order to analyze any phase, we also devise a new framework to charge expected service cost.

In Section 4 we develop lower bounds. We prove that, against oblivious adversaries and as d goes to infinity, no randomized online algorithm can achieve a competitive ratio smaller than 1.8654. Furthermore, we show that no randomized online algorithm can attain a competitiveness smaller than $2-1/(2d)$ against oblivious adversaries in the partial cost model, for general d. No lower bound against oblivious adversaries was known before.

In order to establish our lower bounds, we devise a probability distribution on request sequences: The items of a given list are requested in cyclic order. The number of consecutive requests to the same item is distributed geometrically with mean $2d$. We then compare expected online and offline cost. The analysis of the expected cost incurred by OPT is quite involved. In the partial cost model we partition a request sequence into phases, each ending with a certain surplus of requests to the same item, so that OPT will move that item to the front of its list. As for the lower bound in the full cost model, we prove that we may restrict ourselves to the partial cost model and request sequences referencing two items, provided that we consider projective offline algorithms. Unfortunately, OPT is not projective. Therefore, we define a family of projective offline algorithms and analyze their cost using a Markov chain.

Although the competitive ratio of $c < 2.2442$ achieved by TIMESTAMP(l,p) is a relatively small improvement over the previous best bound of 2.2808, our work – together with the lower bounds – significantly tightens the gap on the best competitiveness of randomized online algorithms in the P^d model.

2 Preliminaries

Given any algorithm A for list update in the P^d model, let $C_A(\sigma)$ denote its costs incurred in serving a request sequence σ. We will consider both the *full cost model* and the *partial cost model*. Again, in the former model, an access to the i-th item in the current list incurs a cost of i. In the latter one the access cost is $i-1$ only. Observe that for every algorithm the total full cost exceeds the total partial cost by precisely $m = |\sigma|$, the length of σ. Hence, if an online algorithm is c-competitive in the partial cost model, it is also c-competitive in the full cost model, too. Therefore, we will analyze TIMESTAMP(l,p) in the partial cost model.

We will prove in Section 3 that TIMESTAMP(l,p) is projective so that it can be analyzed using item pairs. An algorithm is projective if, for any request sequences σ and any pair x, y of items, the relative order of x and y in the list is always the same as if only references to x and y were served on the respective two-item list. Formally consider an algorithm A, a list L and two distinct items $x, y \in L$. Let σ be an arbitrary request sequence. Starting from an initial list configuration $L(0)$, let $L_A(\sigma)$ be the list state immediately after A has served σ. Let $L_A(\sigma)_{xy}$ be the list obtained from $L_A(\sigma)$ by deleting all items other than x and y. Next consider the projected request sequence σ_{xy}, obtained from σ by deleting all requests that are neither to x nor to y. Moreover, let $L(0)_{xy}$ be the list derived from $L(0)$ by removing all items, except for x and y. Finally, starting with $L(0)_{xy}$, let $L_A(\sigma_{xy})$ be the list immediately after A has served σ_{xy}. If A is a randomized algorithm, its random choices on σ_{xy} are identical to those on the requests to x and y in σ.

▶ **Definition 1.** *An algorithm A is projective if and only if, for any request sequence σ, starting list L and any pair $x, y \in L$ of distinct items, $L_A(\sigma)_{xy} = L_A(\sigma_{xy})$.*

This property is desirable since it reduces the analysis to two-item lists. Formally, the following holds:

▶ **Proposition 2.** *Consider the partial cost model. A projective online algorithm A is c-competitive if and only if it is c-competitive on request sequences referencing only two items, which are served on a two-item list.*

Since this proposition is well known in the literature, we give the proof in the full version of the paper. We call an algorithm A *strictly c-competitive on σ* if we have $C_A(\sigma) \leq c \cdot C_{\mathrm{OPT}}(\sigma)$. We will also apply this notion to subsequences $\lambda = \sigma(t) \dots \sigma(t')$ of σ. It is an obvious but very useful fact that A is strictly c-competitive on a sequence σ if we can divide σ into subsequences $\sigma = \lambda_1 \dots \lambda_h$ such that A is strictly c-competitive for each λ_i, $1 \leq i \leq h$. Here we assume that OPT serves the entire sequence σ and evaluate OPT's cost on each subsequence $\lambda_1, \dots, \lambda_h$.

3 The TIMESTAMP(l,p)-algorithm

3.1 The algorithm

Our new algorithm TIMESTAMP(l,p), we refer to it by TS(l,p) or TS for short, is the generalization of TIMESTAMP(p) for the standard cost model [1]. In the P^d model item exchanges are expensive and one can afford them only once in a while. Therefore, for every item x in the given list L, our algorithm maintains a mod l counter $c(x)$, taking values in $\{0, \dots, l-1\}$, for some positive integer l. The counter is initialized uniformly at random and independently of other items.

Consider a request $\sigma(t)$, referencing item x. There are two cases. If $c(x) > 0$ before the request, then $c(x)$ is decremented by 1 and the position of x remains unchanged in the current list. On the other hand, if $c(x) = 0$, then a *master request* occurs. TIMESTAMP(l,p) resets $c(x)$ to $l-1$ and moves x closer to the front of the list, choosing among two policies. With probability p, item x is simply moved to the front of the list (Policy 1). With probability $1-p$, item x is moved more reluctantly (Policy 2). Specifically, the algorithm determines the longest suffix $\lambda(t)$ of the request sequence ending with $\sigma(t)$ such that $\lambda(t)$ contains exactly one master request to x, namely the one at $\sigma(t)$. The algorithm then identifies the first item z in the current list such that at most one master request to z occurs in $\lambda(t)$ and additionally the possible master request, if existent, was served using Policy 2. The algorithm

TIMESTAMP(l,p) moves x in front of z in the list. Observe that x satisfies the conditions formulated for item z. If $z = x$ the item is not moved. In particular, x does never move backward in the list, which is sensible. The intuition of Policy 2 is to pass items whose request frequency, measured in terms of master requests, is not higher than that of x. A pseudo-code description of TIMESTAMP(l,p) is given in Algorithm 1.

Note that, for any item x, two master requests are separated by $l-1$ regular requests to x so that the item is not moved too often. Since $c(x)$ is initialized uniformly at random, the cycles consisting of a master request followed by $l-1$ regular requests to x are shifted in a random fashion in σ. In particular, with probability $1/l$ a request to x happens to be a master request.

Algorithm 1 TIMESTAMP(l,p).

Let $\sigma(t) = x$;
if $c(x) > 0$ **then**
    $c(x) \leftarrow c(x) - 1$;
else // $\sigma(t)$ is a master request
    $c(x) \leftarrow l - 1$;
    With probability p, serve $\sigma(t)$ using **Policy 1** and with probability $1-p$ serve it using **Policy 2**;
    Policy 1: Move x to the front of the list.
    Policy 2: Let $\lambda(t)$ be the longest suffix of the sequence ending with $\sigma(t)$ in which exactly one master request to x occurs. Let z be the first item in the current list for which at most one master request occurs in $\lambda(t)$ and the possible master request was served using Policy 2. If $z \neq x$ move x in front of z in the list.

Theorem 3 gives the competitive ratio of TS(l,p), which is the maximum of six expressions. Nonetheless, the maximum can be determined exactly and truly optimal, algebraic values for p, l and hence the competitive ratio c can be computed. Details are given in the full paper. By plugging in $p = 0.45787$ and $\varphi = l/d = 1.19390$, the reader can verify that indeed $c < 2.2442$. For the optimal choice of p and φ, we have $c_1 = c_2 = c_3$ while the other ratios are smaller.

▶ **Theorem 3.** *Let $\varphi = \frac{l}{d}$. TS(l,p) is c-competitive, where c is the maximum of the following expressions:*

$$c_1 = 1 + \left(\tfrac{1}{2} + \max\{1, 2p\}(1-p)\right)\varphi \quad c_2 = \tfrac{7-3p}{4} + \tfrac{1}{\varphi} \quad c_3 = 1 + \tfrac{3p}{2} - p^2 + \tfrac{2p}{\varphi}$$
$$c_4 = \tfrac{3-p+p^2}{2} + \tfrac{2(1-2p+2p^2)}{\varphi} \quad c_5 = \tfrac{3+p-p^2}{2} + \tfrac{2p^2}{\varphi} \quad c_6 = 2 - p + \tfrac{1-p}{\varphi}.$$

As d goes to infinity, $c < 2.2442$, when choosing $p \approx 0.45787$ and l such that $\varphi \approx 1.19390$.

Figure 1 depicts the max-function we wish to minimize, considering two ranges for p and φ each. A thorough analysis shows indeed that it is identical to the function $\max\{c_1, \ldots, c_4\}$.

3.2 The analysis

We will prove that TS(l,p) is c-competitive in the partial cost model, for the ratio c stated in Theorem 3. As explained in Section 2 this immediately implies c-competitiveness in the full cost model. In [1] it is shown that TS(l,p) is projective for $l = 1$. This is easily generalized to the case of arbitrary l. A proof of the following proposition is contained in the full paper.

▶ **Proposition 4.** *The algorithm TS(l,p) is projective.*

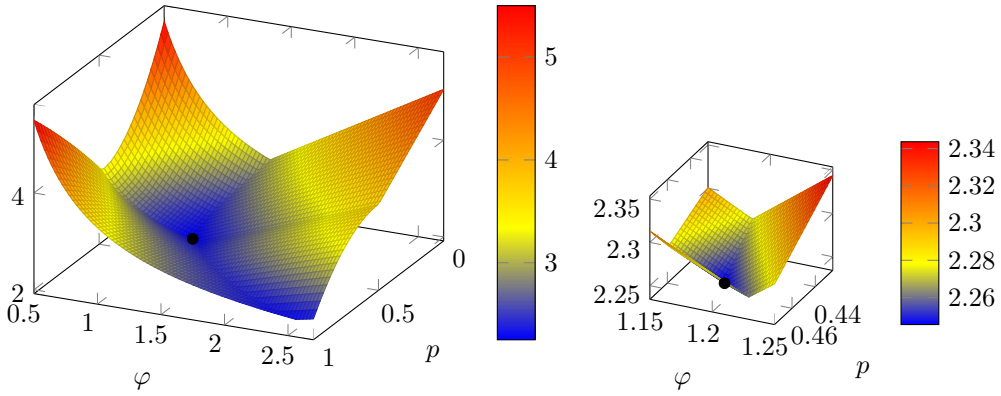

Figure 1 The function of Theorem 3. (The plot is colored.)

Therefore, we may consider an arbitrary request sequence σ, referencing two items x and y that is served on a two-item list. We will compare the expected cost incurred by TS(l, p) to the cost of OPT on σ.

A simple observation is that at any time while TS(l, p) serves σ, the counter value of any item is uniformly distributed over $\{0, \ldots, l-1\}$, independently of the counters of other items. This holds true because the counter value of an item, at any time, is equal to its initial value minus the number of requests to that item served so far modulo l. We will use this fact repeatedly. Of course, care needs to be taken since the choices of TS(l, p) are highly correlated with the counter values.

The analysis of TS(l, p) crucially relies on a new phase partitioning of σ, together with a sophisticated cost charging scheme. More precisely, the service cost of a request to an item is paid at the next master request to that item, provided it occurs in the current phase. Extra care needs to be taken about items where this master request belongs to the next phase. The cost will then be further redistributed so that the expected service cost within a phase can be analyzed independently of counter values at the beginning or during the phase.

Phases and pre-refined cost

Phase partitioning

Given an arbitrary request sequence σ, we partition it into phases. The first phase starts with the first request in σ. Whenever OPT exchanges the two items x and y, the current phase ends and a new phase starts with the request to the item just moved forward. Recall that in response to a request, an algorithm may first exchange items. Then the request is served. Hence, when OPT swaps x and y, the most recent request served is the final one of the current phase. The upcoming request is the first one in the new phase. The last phase ends with the last request in σ.

Suppose that σ has been partitioned into phases $\lambda_1, \ldots, \lambda_k$. We will prove that, for any phase λ_i, TS(l, p)'s expected cost is bounded from above by c times the cost paid by OPT, where c is the ratio given in Theorem 3. This establishes the theorem. In analyzing a phase, we charge OPT a cost of d for the item swap at the end of the phase, in addition to the service costs. We will charge this cost of d also in the last phase λ_k. This way we overestimate OPT's cost on σ by d, which does not affect the competitive ratio.

Now consider an arbitrary phase $\lambda = \lambda_i$. In what follows, y denotes the item stored at the first position in OPT's list. Item x is the one stored at the second position, behind y.

Thus OPT incurs a service cost of 1 for each reference to x. Requests to y cost 0. Item x will be the *good item* because its service cost can be used to balance TS(l,p)'s cost. Item y will be the *bad item*.

Pre-refined cost

We wish to evaluate the algorithms' cost in λ without considering neighboring phases and by focusing on master requests. Therefore, in a series of two steps, we will pass over to the *refined cost setting*. First, we define the *pre-refined cost* for TS(l,p). The service cost incurred by TS(l,p) on a request to an item $z \in \{x,y\}$ is charged at the next master request to z in the phase, if it exists. This charging scheme is applied even if the reference to z itself is a master request. We still have to take care of service cost incurred on requests that are not followed by a master request in the phase.

For the item y, we simply append l requests to y at the end of the phase, right before OPT moves the element y to the back of its list. Then an additional master request to y occurs at the end of the phase and the service cost of previous, unpaid requests to y is covered. Indeed we will add $2l$ requests to y so that any phase ends with two master requests to y. This will be convenient in the further phase analysis. The service cost of OPT does not increase by the addition of requests to y. We remark that in analyzing a phase, we will make no assumptions regarding the end of the preceding phase. In particular, the analysis will not assume that $2l$ requests to x (or y) where appended to the preceding phase as this would impact the behavior of TS(l,p).

As for the requests to x that are not followed by a master request to x in the phase, we have to be more careful. Adding additional requests to x at the end of the phase is infeasible, since it would increase the costs of OPT. Therefore, regarding requests to x that are not followed by a master request to x in a given phase, we ignore their cost. Instead, at the first master request to x in the phase, if it exists, we charge TS(l,p) a cost of l no matter how many non-master requests have occurred so far. We show in the full paper that TS(l,p)'s expected cost does not decrease when changing over to the pre-refined cost setting. For further reference, the following definition summarizes this cost charging scheme.

▶ **Definition 5.** *In the pre-refined cost setting, $2l$ requests to the bad item y are appended at the end of a given phase.* TS(l,p)*'s cost incurred at a request to $z \in \{x,y\}$ is charged to the next master request to z in the phase, if such a request exists. At the first master request to the good item x in the phase, if it exists, a cost of l is charged to* TS(l,p).

▶ **Lemma 6.** TS(l,p)*'s expected cost in a phase does not decrease when passing over to the pre-refined cost setting.*

The proof is given in the full paper. The main idea is to show that, for item x, the expected extra cost charged at the first master request covers the expected cost ignored at the end of the phase. Note that there might be no master request to x in a phase consisting of less than l requests to x. In this case every request to x is a master request with probability $1/l$ and thus pessimistically charged cost $1/l \cdot l = 1$ in expectation. From now on we evaluate TS(l,p)'s cost in the pre-refined cost setting.

Counter fixing for x and refined cost

Given a phase λ, we split OPT's cost so that OPT also incurs service cost at the master requests to x, in addition to the cost for the paid exchange. In particular, this will allow us to fix the counter value of x at the beginning of λ and compare the cost of TS(l,p) to

that of OPT. Let $\mathbf{E}[C_{\mathrm{TS}}(\lambda)]$ denote TS(l,p)'s expected cost in λ. Moreover, let c_x be the counter value of x at the beginning of λ. Finally $\mathbf{E}[C^c_{\mathrm{TS}}(\lambda)]$ denotes TS(l,p)'s expected cost conditioned on $c_x = c$. Since c_x takes any value in $\{0,\ldots,l-1\}$ with equal probability $1/l$, $\mathbf{E}[C_{\mathrm{TS}}(\lambda)] = 1/l \cdot \sum_{c=0}^{l-1} \mathbf{E}[C^c_{\mathrm{TS}}(\lambda)]$.

Assume that λ contains $kl+j$ requests to x, for some $k \geq 0$ and $0 \leq j \leq l-1$. Then OPT's cost in λ is equal to $C_{\mathrm{OPT}}(\lambda) = d + kl + j$. Let k_c be the number of master requests to x in λ conditioned on $c_x = c$. If $c < j$, then $k_c = k+1$; otherwise $k_c = k$.

Define $C^c_{\mathrm{OPT}}(\lambda) := d + k_c l$. Then $1/l \cdot \sum_{c=0}^{l-1} C^c_{\mathrm{OPT}}(\lambda) = d + \sum_{c=0}^{l-1} k_c = d + j(k+1) + (l-j)k = d + kl + j = C_{\mathrm{OPT}}(\lambda)$. This implies

$$\frac{\mathbf{E}[C_{\mathrm{TS}}(\lambda)]}{C_{\mathrm{OPT}}(\lambda)} = \frac{1/l \cdot \sum_{c=0}^{l-1} \mathbf{E}[C^c_{\mathrm{TS}}(\lambda)]}{1/l \cdot \sum_{c=0}^{l-1} C^c_{\mathrm{OPT}}(\lambda)} \leq \max_c \left\{ \frac{\mathbf{E}[C^c_{\mathrm{TS}}(\lambda)]}{C^c_{\mathrm{OPT}}(\lambda)} \right\}.$$

Hence in the following we consider a fixed $c_x = c$ and upper bound $\mathbf{E}[C^c_{\mathrm{TS}}(\lambda)]/C^c_{\mathrm{OPT}}(\lambda)$. We emphasize that $C^c_{\mathrm{OPT}}(\lambda)$ charges service cost l to every master request to x.

We next partition a given phase λ into a *prephase* and a *postphase*, i.e. $\lambda = \lambda_{\mathrm{pre}}\lambda_{\mathrm{post}}$. The prephase λ_{pre} starts at the first request in λ and ends right before the first master request to x in the phase. If no master request to x exists in λ, then λ_{pre} ends with the last request in λ and the postphase is empty. Note that λ_{pre} cannot be empty, since the first request of the phase must go to y. The cost of d the algorithm OPT incurs due to the paid exchange made at the beginning of the phase accounts for the prephase, while the cost of l the optimum algorithm pays at any master request to x is charged in the postphase.

The remainder of this section is devoted to evaluating TS(l,p)'s expected cost on λ_{pre} and λ_{post}. For the analysis on λ_{post}, in a second step, we change over to a refined cost setting that applies in λ_{post}. In this framework TS(l,p) is charged a pessimistic cost of l at every master request to x if item x is not at the front of the list when the request occurs. If after a master request to x algorithm TS(l,p) has item x at the front of the list, then the request is charged an additional cost of $l/2$ to pay the service cost of references to y until the next master request to y occurs. A master request to y is assigned a cost of l if item y has been at the back of TS(l,p)'s list since the last master request to y. This covers the remaining cost for requests to y. Note in this case the preceding master request to y must have been treated using Policy 2. Lemma 8 below shows that TS(l,p)'s expected cost does not decrease when passing over to the refined cost.

▶ **Definition 7.** *In the refined cost setting for* TS(l,p) *in* λ_{post}*, a master request to* x *is charged a* base cost *of* l *if the master request is the first one to* x *in* λ_{post} *(and hence* λ*) or if* x *is not at the front of* TS(l,p)*'s list when the request is presented. An* additional cost *of* $l/2$ *is charged at the master request to* x *if, after service of the request,* x *is at the front of* TS(l,p)*'s list. A master request to* y *in* λ_{post} *is charged a* bad cost *of* l *if* y *has been at the back of* TS(l,p)*'s list since the last master request to* y*.*

▶ **Lemma 8.** *The expected cost of* TS(l,p) *in* λ_{post} *does not decrease when passing over to the refined cost.*

Proof. First consider a master request to either x or y, assuming that the referenced item is at the front of TS(l,p)'s list when the request occurs. Then the item must have been at the front of the list since the last master request to the item. Hence, no cost needs to be charged.

Next consider a master request to x and suppose that x is not at the front of TS(l,p)'s list when the request occurs. The refined cost framework charges the maximum possible service cost for l references to x. This reasoning also applies to the first master request to x in λ_{post}.

We next examine a master request Y to item y, assuming that y is not at the front of TS(l,p)'s list when Y occurs. If y has been at the back of TS(l,p)'s list since the last master request to y, the refined cost framework charges a cost of l to Y, which is equal to the true service cost for the last l requests to y. An overpayment might occur if the last master request to y was in the previous phase.

The interesting case is the one where y was at the front of TS(l,p)'s list after the previous master request to y was served. Then there must exist a master request X to item x where TS(l,p) moved x to the front of the list. We now assign the cost to be paid at Y to X instead. Let c_y^X denote the counter value of y at request X. There are c_y^X non-master requests to y, each incurring a cost of 1, that need to be paid at Y. This cost is now assigned to X. In the following we prove that the expected cost assigned to X, over TS(l,p)'s random choices and the counter value of y at X, is bounded above by $l/2$ times the probability that x is at the front of TS(l,p)'s list after X has been served. We remark that we analyze a slightly more pessimistic cost charging scheme. Any master request X to x is charged a cost of c_y^X if x resides at the front of TS(l,p)'s list after the service of X, independently of whether or not x was just exchanged with y.

Given any master request X to x, let $q(c) = q^X(c)$ be the random variable denoting the probability that item x is at the front of TS(l,p)'s list after X has been served, conditioned on $c_y^X = c$. For simplicity we denote the counter c_y^X of y at X by c_y. Then the expected cost charged at X is exactly

$$\mathbf{E}_{c_y}\left[q(c_y)c_y\right] = \sum_{c=0}^{l-1} \mathbf{P}\left[c_y = c\right] \mathbf{P}\left[x \text{ is at the front after serving } X \mid c_y = c\right] c_y.$$

The probability of x being at the front of TS's list after the service of X is

$$\mathbf{E}_{c_y}\left[q(c_y)\right] = \sum_{c=0}^{l-1} \mathbf{P}\left[c_y = c\right] \mathbf{P}\left[x \text{ is at the front after serving } X \mid c_y = c\right].$$

In the refined cost framework, the expected additional cost charged at X is $\mathbf{E}_{c_y}\left[q(c_y)\right] \cdot l/2 \geq \mathbf{E}_{c_y}\left[q(c_y)\right] \mathbf{E}_{c_y}\left[c_y\right]$. Hence it remains to prove that $\mathbf{E}_{c_y}[q(c_y)c_y] \leq \mathbf{E}_{c_y}[q(c_y)]\mathbf{E}_{c_y}[c_y]$. The last inequality holds if and only if the covariance of the random variables c_y and $q(c_y)$ is not positive. The latter property in turn holds if $q(c_y)$ is (non-strictly) decreasing in c_y. We verify this property by giving a complete characterization of the function $q(c_y)$. Let r denote the number of requests to y between X and the master request to x before that.

- If $c_y < l - r$, then the master request preceding X is to x and the probability of x being at the front of the list after X is 1.
- If $l - r \leq c_y < 2l - r$, then the two master requests preceding X are xy (in that order). The element x is moved to the front if and only if either X is served with Policy 1 or the preceding master request to y is served with Policy 2. Hence, the probability of x being at the front of the list is $q(c_y) = p + (1-p)^2$.
- If $2l - r \leq c_y$, then the two master requests preceding X both go to y and the probability of x being at the front of the list after the service of X is exactly p, the probability that X was handled by TS(l,p) using Policy 1.

In particular the function $q(c_y)$ is decreasing. ◀

The cost analysis of a phase

Consider an arbitrary phase λ. Let λ_{post} be its postphase, which starts with a master request to x and ends with at least two master requests to y, according to the phase adjustment we did when changing over to the pre-refined cost. We next modify λ_{post} so that we can partition it into subphases, each ending with at least two master requests to y. For this

purpose consider two consecutive master requests to x that are preceded by a single master request to y. The next proposition shows that we can insert l new requests to y, thereby generating a new master request to y, without decreasing the strict competitiveness on λ_{post}. The proof is given in the full version of the paper.

▶ **Proposition 9.** *Consider a subsequence of consecutive master requests $x_0y_1x_1x_2$ in λ_{post} where master request z_i goes to z, $z \in \{x, y\}$. Add l new requests to y before x_1.* TS(l, p)*'s expected refined cost on the resulting sequence of master requests $x_0y_1y_2x_1x_2$ is at least as high as that on the former sequence.* OPT*'s cost does not change.*

Hence in λ_{post}, whenever two master requests to x are preceded by exactly one master request to y, we insert l new requests to y. Again, this creates a second master request to y. We then partition λ_{post} into subphases, each ending with two master requests to y. More precisely, the first subphase starts with the first master request in λ_{post}. A subphase ends immediately before a master request to x that is preceded by at least two master requests to y. Observe that whenever at least two consecutive master requests to x occur, they start a new subphase. We obtain two types of subphases, specified by their master requests.

- Type 1: $x(yx)^k y^{2+i}$ for some $i, k \geq 0$.
- Type 2: $x^{2+j}(yx)^k y^{2+i}$ for some $i, j, k \geq 0$.

In the following four lemmas we analyze TS(l, p) on any subphase. If the subphase is the first one in λ_{post}, we analyze it jointly with the preceding prephase λ_{pre}. Together the four lemmas imply Theorem 3. The cost ratios c_i, $1 \leq i \leq 6$, stated in the lemmas are identical to those of Theorem 3. In Lemma 10 we first consider Type 2 subphases as TS(l, p)'s performance ratio on those phases can be bounded independent of the preceding master requests. Then Lemmas 11, 12 and 13 address Type 1 subphases with the preceding master requests. Lemma 11 considers the case that a Type 1 subphase is preceded by two master requests to y. Note that this is, in particular, the case for any Type 1 subphase that is not the first subphase in the given λ. Lemmas 12 and 13 address the special cases where a Type 1 subphase is preceded (a) by master requests to x and y, in this order, or (b) by a master request to x. In both cases such a subphase must be the first one in λ_{post}. Moreover, λ_{pre} consists of less than two master requests to y in these special cases. In fact in case (b), there is no master request in λ_{pre}. Proofs of all the lemmas are presented in the full paper.

▶ **Lemma 10.** TS(l, p) *is strictly* $\max\{c_1, c_2, c_4\}$*-competitive on any subphase of Type 2, including a possible preceding prephase.*

▶ **Lemma 11.** TS(l, p) *is strictly* $\max\{c_1, c_3, c_4\}$*-competitive on any subphase of Type 1, including a possible prephase, if the subphase is preceded by two master requests to y.*

▶ **Lemma 12.** TS(l, p) *is strictly* $\max\{c_1, c_4, c_5\}$*-competitive on any subphase of Type 1 and its leading prephase if the subphase is preceded by master requests to x and y in this order.*

▶ **Lemma 13.** TS(l, p) *is strictly* $\max\{c_1, c_4, c_6\}$*-competitive on any subphase of Type 1 and its leading prephase if the subphase is preceded by a master request to x.*

4 Lower bounds

We develop lower bounds in the partial cost and the full cost P^d models.

▶ **Theorem 14.** *Let A be a randomized online algorithm for list update in the partial cost P^d model. If A is c-competitive against oblivious adversaries, then $c \geq 2 - \frac{1}{2d}$. This holds even for request sequences referencing only two items.*

We conjecture that a lower bound of $2 - \frac{1}{2d}$ also holds for the full cost P^d model. The difficulty is to bound the cost of OPT from above on request sequences referencing a general set of n items. We can establish the following lower bound in the full cost P^d model.

▶ **Theorem 15.** *Let A be a randomized online algorithm for list update in the full cost P^d model. If A is c-competitive against oblivious adversaries, then c is at least*

$$\frac{1}{1+2W\left(\frac{-1}{2e}\right)} - O\left(\frac{1}{d}\right) \approx 1.8654.$$

Here W is the upper branch of the Lambert-W-function, i.e. $W\left(\frac{-1}{2e}\right)$ is largest value x satisfying $2xe^{x+1} = -1$.

Table 1 presents the values of the lower bound of Theorem 15, for small values of d, up to $d = 100$. For $d = 1$, i.e. the standard cost model, our bound matches the one by Teia [27].

Table 1 The lower bound in the full cost P^d model for various d.

d	$c(d)$
1	1.5
2	1.8036
3	1.8270
4	1.8337
5	1.8420

d	$c(d)$
6	1.8438
7	1.8485
10	1.8531
20	1.8594
100	1.8642

A probability distribution on request sequences

For the proof of the two theorems above, we use Yao's minimax principle [28]. We define a probability distribution on request sequences and compare the expected cost incurred by any (deterministic) online algorithm A to that of OPT. For the definition of our probability distribution, we describe how to sample a request sequence according to it. Let $L(0) = [x_0 \dots x_{n-1}]$ be a starting list of n items; x_i precedes x_{i+1} for $i = 0, \dots, n-2$. Throughout the sampling process the list is static, i.e. no rearrangement of items is done. The items will be requested in a cyclic fashion, in decreasing order of index. Each item will be referenced a certain number of times.

Formally, in addition to $L(0)$, the sampling process takes as input a number $N \in \mathbb{N}$ and a starting item $x \in L(0)$. Typically, the starting item is equal to the last item x_{n-1} in the list but the process is defined for any $x \in L(0)$. The sampling process produces a request sequence consisting of N segments. Throughout this section a *segment* is a maximal subsequence of requests to the same item. Moreover, to simplify notation, we set $x_i = x_{i \bmod n}$, for all $i \in \mathbb{Z}$.

Initially, the request sequence σ to be produced is equal to the empty string. In each step of the sampling procedure, if $N > 0$, a request to the current item x is appended at the end of σ. Then with probability $p = 1/(2d)$, the value N is decremented and $x = x_i$ is replaced by x_{i-1}. Hence in this event the segment of requests to x_i ends and a segment of references to x_{i-1} starts. Note that the length of a segment is geometrically distributed with $p = 1/(2d)$ and thus equal to $2d$ in expectation. The process stops when $N = 0$. A pseudo-code description of the sampling process is given in Algorithm 2. Let $\mathcal{S}_N[x]$ denote the resulting probability distribution on request sequences with starting item x. Furthermore, let $\mathcal{S}_N = \mathcal{S}_N[x_{n-1}]$.

The lower bound construction in the full cost model will work with sequences generated according to this particular process. In the partial cost model we will have to extend the process so that the request sequences admit a phase partitioning and end with a complete phase.

Algorithm 2 SampleRequestSequence.

Input: $L(0) = [x_0 \dots x_{n-1}]$, $N \in \mathbb{N}$ and $x \in L(0)$.
$\sigma :=$ empty string;
while $N > 0$ **do**
    Append x to σ;
    With probability $p = \frac{1}{2d}$, decrement N and replace $x = x_i$ by x_{i-1};
Return σ;

We next introduce the notion of an algorithm being *uncompetitive*. It will be particularly useful when deriving our lower bound in the full cost model. Nonetheless, the notion applies to both the partial and the full cost models and it will always be clear from the context which model is used.

▶ **Definition 16.** *Let $c \geq 1$. An online algorithm A is* c-uncompetitive against an offline algorithm B *if, for every $\varepsilon > 0$, there exists an initial list $L(0)$ such that*

$$\lim_{N\to\infty} \frac{\mathbf{E}_{\sigma\sim\mathcal{S}_N}[C_A(\sigma)]}{\mathbf{E}_{\sigma\sim\mathcal{S}_N}[C_B(\sigma)]} \geq c - \varepsilon.$$

If $B = \mathrm{OPT}$, algorithm A is simply c-uncompetitive. *Finally, A is* c-uncompetitive on two-item sequences (against B) *if the above condition holds with $\varepsilon = 0$, for lists consisting of two items.*

The terminology is relevant due to the following obvious fact.

▶ **Lemma 17.** *If a (possibly randomized) online algorithm A is c-uncompetitive, then its competitive ratio is not smaller than c.*

In a first step we bound the expected cost incurred by any online algorithm A on request sequences generated according to $\mathcal{S}_N$ from below. In Theorem 14, we consider the partial cost model and request sequences referencing two items. In the proof of Theorem 15, we show that we may restrict ourselves to the partial cost model and, again, may focus on two-item request sequences if we consider a projective offline algorithm. Therefore, the following Lemma 18 will be essential in the proofs of Theorems 14 and 15. Let $L(0) = [xy]$ be a list consisting of two items x and y; where x is initially stored in front of y. Consider the distribution $\mathcal{S}_N = \mathcal{S}_N[y]$.

▶ **Lemma 18.** *Let $L(0) = [xy]$. For every online algorithm A and every N, we have in the partial cost model*

$$\mathbf{E}_{\sigma\sim\mathcal{S}_N}[C_A(\sigma)] \geq dN.$$

Proof. It suffices to prove the lemma for deterministic algorithms because any randomized algorithm is a probability distribution over deterministic ones. Hence, let us assume for the sake of a contradiction that there were a deterministic online algorithm A and an $N \in \mathbb{N}$ such that

$$C = \mathbf{E}_{\sigma\sim\mathcal{S}_N}\left[C_A^{\mathrm{part}}(\sigma)\right] < dN.$$

Among all possible choices, we choose such a pair (A, N) with N minimal and, in case of ties, C minimal. We clearly have $N > 0$. Given a sequence σ we denote by $\alpha(\sigma)$ the number of leading requests to y and by $\beta(\sigma)$ the number of requests to x following those.

We show that the algorithm A does not immediately move the element y to the front of its list. Indeed, let us write $\sigma = y^{\alpha(\sigma)}\sigma'$. If we choose $\sigma \sim \mathcal{S}_N$ and pass over to σ', it is the same as choosing $\sigma' \sim S_{N-1}[x]$. Now if A were to move the item y immediately to the front, it would incur exactly cost d on the sequence $y^{\alpha(\sigma)}$. Then, by the minimality of N, it will incur an expected cost of at least $d(N-1)$ on σ' and hence a total expected cost of at least dN. This is a contradiction to the assumption that $C < dN$.

Hence we may assume that A does not immediately move y to the front of its list. For $\sigma \sim \mathcal{S}_N$ we consider two cases. With probability $\frac{1}{2d}$, the sequence σ starts with a single request to y, i.e. we have $\alpha(\sigma) = 1$. Then $\sigma = yx^{\beta(\sigma)}\sigma''$. Note that we have $\sigma" \sim S_{N-2}$ if we pick $\sigma \sim \mathcal{S}_N|_{\alpha(\sigma)=1}$. On the sequence $yx^{\beta(\sigma)}$ the algorithm A incurs cost of exactly 1 at the first request. By the minimality of N we have for the remainder $\sigma"$ of the sequence σ

$$\mathbf{E}_{\sigma"}\left[C_A(\sigma")\right] \geq d(N-2).$$

Note that this is obviously true if $N - 2 \leq 0$ holds.

On the other hand, with probability $1 - \frac{1}{2d}$, we have $\alpha(\sigma) > 1$. In this case the algorithm A incurs cost 1 on the first request to y. We consider the algorithm B which behaves like the algorithm A after having read a request to y, i.e. on the input sequence σ it behaves like A would on the corresponding suffix of $y\sigma$. By the minimality of C we have $\mathbf{E}_{\sigma\sim\mathcal{S}_N}[C_B(\sigma)] \geq C$. Note that sampling σ from $\mathcal{S}_N$ conditioned on it starting with at least two requests to y is the same as sampling σ from $\mathcal{S}_N$ and appending a request to y to its front. Hence we get

$$\mathbf{E}_{\sigma\sim\mathcal{S}_N}\left[C_A(\sigma) \mid \alpha(\sigma) > 1\right] = 1 + \mathbf{E}_{\sigma\sim\mathcal{S}_N}\left[C_B(\sigma)\right] \geq 1 + C.$$

In total we have

$$\begin{aligned} C &= \tfrac{1}{2d}\mathbf{E}_{\sigma\sim\mathcal{S}_N}\left[C_A(\sigma) \mid \alpha(\sigma) = 1\right] + \left(1 - \tfrac{1}{2d}\right)\mathbf{E}_{\sigma\sim\mathcal{S}_N}\left[C_A(\sigma) \mid \alpha(\sigma) > 1\right] \\ &\geq \tfrac{d(N-2)+1}{2d} + \left(1 - \tfrac{1}{2d}\right)(C+1) \\ &= \tfrac{dN}{2d} + \left(1 - \tfrac{1}{2d}\right)C. \end{aligned}$$

The last inequality implies $C \geq dN$. Again, we have reached the desired contradiction. ◀

The challenging part in the proofs of Theorems 14 and 15 is to bound the expected cost incurred by OPT on sequences drawn according to $\mathcal{S}_N$. In the following we sketch some of the main ideas. Full analyses are presented in the full paper.

Proof sketch for Theorem 14

We focus on request sequences referencing two items x and y, and hence on sequences $\sigma \sim \mathcal{S}_N$ with initial list $L(0) = [xy]$. Such sequences consist of N segments that in turn reference y and x. Given a sequence σ' and an item $z \in \{x, y\}$, let $|\sigma'|_z$ be the number of requests to z in σ'.

The optimal algorithm for two-item sequences is the following: We consider the case where y is at the front of the list of OPT and x is at the back. The opposite case works symmetrically. If y is requested next, OPT will obviously not move it to the back, but wait, until x is requested. If x is requested next, OPT needs to decide whether to move x to the front of its list. It does so if and only if there is a prefix σ' of future requests

with $|\sigma'|_x = |\sigma'|_y + 2d$ and no (non-empty) prefix σ'' of σ' satisfies $|\sigma''|_x \leq |\sigma''|_y$. There is a special case if prefix σ' comprises the entire sequence of future requests. Then we only require in the first condition that $|\sigma'|_x \geq |\sigma'|_y + d$ holds true. In the following we will analyze a simpler algorithm O, which omits this special case. While O is only close-to-optimal, it still gives a good upper bound on OPT. Additionally, we will consider the algorithm $\bar{O}$ that always keeps its list in opposite order, compared to the list of O. On each request in a given sequence, exactly one of the two algorithms has a service cost of 1.

Given any sequence σ, let $\tilde{C}_O(\sigma)$ and $\tilde{C}_{\bar{O}}(\sigma)$ be the pure service cost of O and $\bar{O}$. Furthermore, let $D_O(\sigma)$ be the cost incurred by O for paid exchanges. Define $K(\sigma) = \tilde{C}_{\bar{O}}(\sigma) - \tilde{C}_O(\sigma) - 2D_O(\sigma)$. For $\sigma \sim \mathcal{S}_N$, $K(\sigma)$ is a random variable, which we denote by E_N. We will show that $\mathbf{E}_{\sigma\sim\mathcal{S}_N}[C_O(\sigma)] = dN - \mathbf{E}[E_N]/2$. Thus, by Lemma 18, the value $\mathbf{E}[E_N]/2$ is a lower bound for the cost the algorithm O saves compared to any online algorithm. The heart of the analysis is to estimate that $\lim_{N\to\infty} \mathbf{E}[E_N]/N \geq 2d(2d-1)/(4d-1)$. Together with Lemmas 18, this implies that any online algorithm A is c-uncompetitive against OPT with $c \geq dN/(dN - \frac{N}{2}\frac{2d(2d-1)}{4d-1}) = 2 - \frac{1}{2d}$. In particular, by Lemma 17 its competitive ratio is at least $2 - 1/(2d)$.

For the analysis of $\mathbf{E}[E_N]$, we partition a request sequence $\sigma \sim \mathcal{S}_N$ into phases. Each phase λ consists of a series of subphases $\mu_1, \ldots, \mu_l$. We describe how to obtain μ_1. At the beginning of λ, let $z \in \{x, y\}$ be the current item in the sampling process, i.e. the first segment in λ consists of requests to z. Let $z' \in \{x, y\}$, $z' \neq z$, be the other item. Starting at the beginning of λ we scan the generated requests, adding them to μ_1 until one of the following events occurs. (1) $|\mu_1|_z = |\mu_1|_{z'}$ or (2) $|\mu_1|_z = |\mu_1|_{z'} + 2d$. In case (1) we call μ_1 a *zero-subphase*; in case (2) μ_1 is an *up-subphase*. When event (1) or (2) occurs, the remaining requests of the current segment are appended to μ_1. These requests form the *post-subphase*. Then μ_1 ends. Each following subphase μ_i, for $i > 1$, is obtained in the same way, starting at the end of μ_{i-1}. Phase λ ends when, for the first time, an up-subphase is obtained. Formally, algorithm O moves item z to the front of the list right before the up-subphase.

We extend the sampling process so as to obtain sequences ending with a complete phase. This induces a slightly related probability distribution of request sequences, which is not critical though. For any λ, let $C = K(\lambda)$ and let $R = R(\lambda)$ be the number of segments in λ. At the heart of the analysis, using a recurrence relation, we prove $\lim_{N\to\infty} \mathbf{E}[E_N]/N \geq \mathbf{E}[C]/\mathbf{E}[R]$. We argue that $\mathbf{E}[C]$ is the total length of all post-subphases in λ. Then we show $\mathbf{E}[C] = (2d-1)/(1-P_0)$ and $\mathbf{E}[R] = (4d-1)/(2d(1-P_0))$, where P_0 is the probability that a subphase happens to be a zero-subphase. This yields the desired bound.

Proof sketch for Theorem 15

In this setting we are given a list $L = L(0)$ with n items. Again, we focus on request sequences generated according to $\mathcal{S}_N$ (Algorithm 2). We first prove that we may restrict ourselves to the partial cost model and two-item request sequences if we focus on *projective* offline algorithms: If every deterministic online algorithm is c-uncompetitive on two-item sequences against a projective offline algorithm B in the partial cost model, then every deterministic online algorithm is c-uncompetitive against B in the full cost model. Nonetheless, the analysis of the offline algorithm O developed for the proof of Theorem 15 cannot be employed because O and OPT are not projective.

Therefore, we define a family of projective offline algorithms B_h, for $0 < h < 2d$. Consider a request sequence to be served on a list L consisting of n items. When presented with any request, B_h works as follows: If the next h requests reference the same element $z \in L$, algorithm B_h moves z to the front of its list. Otherwise it does not change the list. Algorithm

B_h is projective, for sequences σ generated by $\mathcal{S}_N$, because items are requested in cyclic order in σ and a projection to item pairs does not create longer subsequences of the same item.

Given the result for projective offline algorithms stated above, it suffices to analyze B_h on two-item sequences. Let $L(0) = [xy]$ be the starting list. Technically, the main step is to show that, for any $\sigma \sim \mathcal{S}_N$, the expected cost incurred by B_h is

$$\mathbf{E}_{\sigma\sim\mathcal{S}_N}[C_{B_h}(\sigma)] = \frac{\left(\frac{1-(1-p)^h}{p} - h(1-p)^{h-1}\right) + (1-p)^{h-1}d}{2-(1-p)^{h-1}}N + o(N), \tag{1}$$

where $p = 1/(2d)$. In order to establish this equation, given $\sigma \sim \mathcal{S}_N$, we write $\sigma = z_1^{\alpha_1} z_2^{\alpha_2} \dots z_N^{\alpha_N}$, where $z_1 = y$ and $z_2 = x$. If we consider the algorithm B_h at the beginning of the subsequence $z_t^{\alpha_t}$, there can be three cases.

- If z_t is at the back of the list of B_h and $\alpha_t \geq h$ holds, we say the sequence $z_t^{\alpha_t}$ is of *type* h. The algorithm B_h will incur a cost d on it because it immediately moves z_t to the front.
- If z_t is at the back of the list of B_h and $\alpha_t = j < h$, we say the sequence $z_t^{\alpha_t}$ is of *type* $(j, 1)$. The algorithm B_h will incur cost j on it.
- If z_t is at the front of the list of B_h, the algorithm will incur no cost on it. Further we have $t > 1$ and $\alpha_{t-1} = j < h$, for some j. We say the sequence $z_t^{\alpha_t}$ is of *type* $(j, 2)$.

For a type $w \in W_h = \{1, \dots, h-1\} \times \{1, 2\} \cup \{h\}$, let $P(t)_w$ be the probability that the sequence $z_t^{\alpha_t}$ is of this type if we sample $\sigma \sim \mathcal{S}_N$. The process forms a Markov chain: From type $(j, 1)$, for $j < h$, we pass to the type $(j, 2)$ with probability 1. From every other type w we pass to type $(j, 1)$, for $j < h$, with probability $p(1-p)^{j-1}$ and to type h with probability $(1-p)^{h-1}$. Using basic Markov analysis we evaluate $\lim_{t\to\infty} P(t)_w$, for $w = (j, 1)$, $w = (j, 2)$ and $w = h$. Using the respective expressions, we obtain (1). Using this expression, we can immediately verify the competitive ratios in table 1 using the following optimal choices of h.

Table 2 The best choices of h, for small values of d.

d	$c(d)$	h
1	1.5	1
2	1.8036	3
3	1.8270	5
4	1.8337	6
5	1.8420	8

d	$c(d)$	h
6	1.8438	10
7	1.8485	11
10	1.8531	16
20	1.8594	31
100	1.8642	154

To obtain the lower bound for $d \to \infty$ we set $h = \lfloor 2\tilde{h}d \rfloor$ for some $\tilde{h} > 0$ to be determined later. Then $\mathbf{E}_{\sigma\sim\mathcal{S}_N}[C_{B_h}(\sigma)]$ is of the form $(1 + \frac{2\tilde{h}e^{-\tilde{h}}}{e^{-\tilde{h}}-2} + O\left(\frac{1}{d}\right))dN + o(N)$. Lemma 18 implies that every online algorithm A is c-uncompetitive against the algorithm B_h in the full cost model, where c is at least $(1 + 2\tilde{h}e^{-\tilde{h}}/(e^{-\tilde{h}} - 2))^{-1}$ as $d \to \infty$. Lemma 17 ensures that the competitive ratio of A is not smaller than the above expression. Theorem 15 follows if we set $\tilde{h} = W(-1/(2e)) + 1$.

References

1 S. Albers. Improved randomized on-line algorithms for the list update problem. *SIAM J. Comput.*, 27(3):682–693, 1998.

2 S. Albers and S. Lauer. On list update with locality of reference. *J. Comput. Syst. Sci.*, 82(5):627–653, 2016.

3 S. Albers, B. von Stengel, and R. Werchner. A combined BIT and TIMESTAMP algorithm for the list update problem. *Inf. Process. Lett.*, 56(3):135–139, 1995.
4 S. Albers and J. Westbrook. Self-organizing data structures. In *Online Algorithms, The State of the Art*, Springer LNCS 1442, pages 13–51, 1998.
5 C. Ambühl, B. Gärtner, and B. von Stengel. Optimal lower bounds for projective list update algorithms. *ACM Trans. Algorithms*, 9(4):31:1–31:18, 2013.
6 S. Angelopoulos and P. Schweitzer. Paging and list update under bijective analysis. *J. ACM*, 60(2):7:1–7:18, 2013.
7 A. Borodin and R. El-Yaniv. *Online computation and competitive analysis.* Cambridge University Press, 1998.
8 J. Boyar, S. Kamali, K.S. Larsen, and A. López-Ortiz. On the list update problem with advice. *Inf. Comput.*, 253:411–423, 2017.
9 M. Burrows and D. Wheeler. A block sorting lossless data compression algorithm. *Technical Report 124*, 1994.
10 M. Crochemore, R. Grossi, J. Kärkkäinen, and G.M. Landau. Computing the Burrows-Wheeler transform in place and in small space. *J. Discrete Algorithms*, 32:44–52, 2015.
11 R. Dorrigiv, M.R. Ehmsen, and A. López-Ortiz. Parameterized analysis of paging and list update algorithms. *Algorithmica*, 71(2):330–353, 2015.
12 S. Irani. Two results on the list update problem. *Inf. Process. Lett.*, 38(6):301–306, 1991.
13 S. Kamali, S. Ladra, A. López-Ortiz, and D. Seco. Context-based algorithms for the list-update problem under alternative cost models. In *Proc. 2013 Data Compression Conference (DCC)*, IEEE, pages 361–370, 2013.
14 S. Kamali and A. López-Ortiz. A survey of algorithms and models for list update. In *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of his 66th Birthday*, pages 251–266, 2013.
15 S. Kamali and A. López-Ortiz. Better compression through better list update algorithms. In *Proc. 2014 Data Compression Conference (DCC)*, IEEE, pages 372–381, 2014.
16 R. Karp and P. Raghavan. Personal communication cited in [24].
17 A. López-Ortiz, M.P. Renault, and A. Rosén. Paid exchanges are worth the price. In *Proc. 32nd International Symposium on Theoretical Aspects of Computer Science (STACS15)*, LIPIcs 30, pages 636–648, 2015.
18 M.S. Manasse, L.A. McGeoch, and D.D. Sleator. Competitive algorithms for on-line problems. In *Proc. 20th Annual ACM Symposium on Theory of Computing*, pages 322–333, 1988.
19 G. Manzini. An analysis of the Burrows-Wheeler transform. *J. ACM*, 48(3):407–430, 2001.
20 C. Martínez and S. Roura. On the competitiveness of the move-to-front rule. *Theor. Comput. Sci.*, 242(1-2):313–325, 2000.
21 J. McCabe. On serial files with relocatable records. *Operations Research*, 12:609–618, 1965.
22 J.I. Munro. On the competitiveness of linear search. In *Proc. 8th Annual European Symposium on Algorithms (ESA00)*, Springer LNCS 1879, pages 338–345, 2000.
23 N. Reingold and J. Westbrook. Off-line algorithms for the list update problem. *Inf. Process. Lett.*, 60(2):75–80, 1996.
24 N. Reingold, J. Westbrook, and D.D. Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11(1):15–32, 1994.
25 J. Sirén. Burrows-Wheeler transform for terabases. In *Proc. 2016 Data Compression Conference (DCC)*, IEEE, pages 211–220, 2016.
26 D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.
27 B. Teia. A lower bound for randomized list update algorithms. *IPL*, 47(1):5–9, 1993.
28 A.C.C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proc. 18th IEEE Annual Symposium on Foundations of Computer Science*, pages 222–227, 1977.