Scheduling in the Random-Order Model

Susanne Albers[1] and Maximilian Janke[2]

[1]Department of Computer Science, Technical University of Munich, albers@in.tum.de
[2]Department of Computer Science, Technical University of Munich, 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.

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, 5–8, 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 $\frac{85}{49} \approx 1.7346$. 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 $\frac{1+\sqrt{5}}{2} \approx 1.6180$. 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 $\frac{12}{11}$ 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 $\frac{5+\sqrt{17}}{2} \approx 4.5616$ [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 $\frac{5+\sqrt{17}}{4} \approx 2.2808$. 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, $0 \leq p \leq 1$. 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 $2 - \frac{1}{2d}$ 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 fairly important and well-known in the literature, since it reduces the analysis to two-item lists.

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.*

Proof. Obviously, if A is c-competitive, then it is also c-competitive on request sequences that reference only two items and are served on a two-item list. In the following we prove the reverse statement.

Consider an algorithm $B \in \{A, \mathrm{OPT}\}$ and any request sequence $\sigma = \sigma(1), ..., \sigma(m)$ to be served on a list L consisting of n items. Fix an arbitrary pair $x, y \in L$ of distinct items. Let $R_B^{xy}(\sigma)$ be the number of requests $\sigma(t)$, $1 \le t \le m$, such that (a) $\sigma(t)$ references x or y and (b) immediately before the service of $\sigma(t)$ the referenced item is behind the other item of $\{x, y\}$ in the current list maintained by B. Let $E_B^{xy}(\sigma)$ be the number of paid exchanges of x and y. Set $C_B^{xy}(\sigma) = R_B^{xy}(\sigma) + d \cdot E_B^{xy}(\sigma)$. Intuitively, $C_B^{xy}(\sigma)$ is the cost incurred by items x and y on requests to those items when B serves σ. Recall that we work in the partial cost model. Considering all pair of items we have

$$C_B(\sigma) = \sum_{\substack{x,y \in L \\ x \neq y}} C_B^{xy}(\sigma).$$

Since A is projective, there holds $C_A^{xy}(\sigma) = C_A(\sigma_{xy})$, for any $x \neq y$. Using the fact that A is c-competitive on request sequences referencing two items, we obtain

$$C_A(\sigma) = \sum_{\substack{x,y \in L \\ x \neq y}} C_A^{xy}(\sigma) = \sum_{\substack{x,y \in L \\ x \neq y}} C_A(\sigma_{xy}) \le \sum_{\substack{x,y \in L \\ x \neq y}} (c \cdot C_{\mathrm{OPT}}(\sigma_{xy}) + \alpha),$$

for some constant α. Note that $C_{\mathrm{OPT}}(\sigma_{xy})$ is the optimum cost of serving σ_{xy} on a two-item list and hence not larger than $C^{xy}_{\mathrm{OPT}}(\sigma)$. We conclude

$$C_A(\sigma) \leq \sum_{\substack{x,y\in L\\ x\neq y}} (c\cdot C_{\mathrm{OPT}}(\sigma_{xy})+\alpha) \leq c\cdot \sum_{\substack{x,y\in L\\ x\neq y}} C_{\mathrm{OPT}}(\sigma_{xy}) + \binom{n}{2}\cdot\alpha \leq c\cdot C_{\mathrm{OPT}}(\sigma) + \binom{n}{2}\cdot\alpha,$$

and A is c-competitive. □

Developing techniques not relying on the notion of projectivity would be a huge breakthrough result in the study of the List Update Problem. To date only Irani [12] has ever proposed a sensible algorithm for general numbers of lists items that is not projective. All other algorithms [1, 3, 24, 26] are projective and use the partial cost model. Techniques relying on projectivity and the partial cost model cannot lead to better competitive ratios [5] and thus there is considerable interest in finding further improvement without using Proposition 2. See also [5, 14]. This central problem deserves a name for future ease of reference. Thus we propose the *Projectivity Hypothesis*. Proving or disproving this result would be a major breakthrough in the study of the List-Update Problem. For deterministic online algorithms in the standard model the hypothesis is true [16, 26] while it is false in the offline setting [17]. The hypothesis is most interesting for randomized algorithms in the standard model and related settings.

Hypothesis (The Projectivity Hypothesis)**.** *The optimum competitive ratio in the List Update Problem for general list sizes n is obtained by a projective algorithm in the partial cost model.*

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 $\lambda_1,\dots,\lambda_h$. We will use this fact a lot.

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 $\frac{1}{l}$ 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. We leave the technical details of the computation to the appendix. By plugging in $p = 0.45787$ and $\varphi = \frac{l}{d} = 1.19390$, 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 on the following page depicts the max-function we wish to minimize for Theorem 3, considering two ranges for p and φ each. A thorough analysis shows indeed that it is identical to the function $\max\{c_1, \ldots, c_4\}$. We give a brief sketch on how we solve the minimization problem in Theorem 3 in Appendix A, including an exact algebraic but complicated description of the optimal values p and φ.

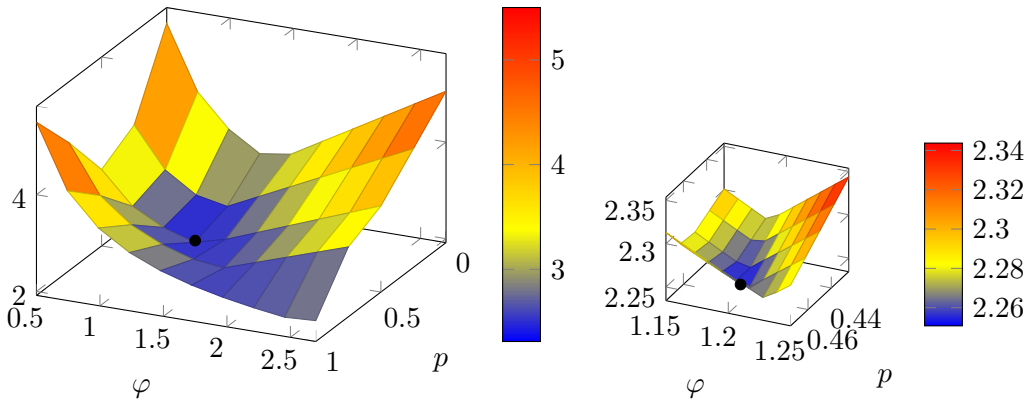

Figure 1: The function of Theorem 3. (The plot is colored.)

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.

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

Proof. In order to show that TS is projective it suffices to see that it is projective for every choice of the initial values of the counters. Hence, suppose for contradiction sake, that TS(l,p) is not projective for some fixed choice of counters. Then let $\sigma = \sigma(1),\ldots,\sigma(m)$ be a shortest request sequence for which projectivity is violated, i.e. TS(l,p) is projective on $\sigma' = \sigma(1),\ldots,\sigma(m-1)$ but not on σ. In this case $\sigma(m)$ must be a master request to some item, say x, since otherwise a list configuration does not change. Let y be an item such that the relative order of x and y in the full list differs from that in the two-item list, consisting of x and y, after the service of $\sigma(m)$. Prior to the service of $\sigma(m)$, item y precedes x in both the full list and the two-item list. If this were not the case, there could be no conflict because referenced items never move backward in the list.

Let $\lambda(m)$ be the longest suffix of σ containing only one master request to x, namely $\sigma(m)$. Then $\lambda(m)_{xy}$ is the corresponding suffix in σ_{xy}. Observe that $\lambda(m)$ and $\lambda(m)_{xy}$ start after the same master request to x in σ or, alternatively, at the beginning of σ.

First assume that $\sigma(m)$ is served (a) using Policy 1 or (b) using Policy 2 and at most one master request to y occurs in $\lambda(m)$ and $\lambda(m)_{xy}$ that is also served using Policy 2. In this case TS(l,p) moves x in front of y in the full list and the two-item list. Hence there is no conflict.

Next assume that $\sigma(m)$ is served using Policy 2 and one master request to y occurs in $\lambda(m)$ and $\lambda(m)_{xy}$ that is served using Policy 1. After the master request to y, this item is at the front of the lists. Consider any item z with at most one master request in $\lambda(m)$ and $\lambda(m)_{xy}$, where the possible master request is served using Policy 2. Suppose that this possible master request to z occurs after the master request to y, say at time t in σ. After its service, item z stays behind y in the two-item list. This also holds true if $\lambda(t)_{yz}$ contains two or more master requests to y. Thus immediately before the service of $\sigma(m)$, item y precedes z in the two-item list. Since TS(l,p) is projective on σ' this relation also holds for the full list. Hence when TS(l,p) serves $\sigma(m)$, item x passes z in the full list but stays behind y in both the full and the two-item list.

Therefore, in the remainder of the proof we may concentrate on the case that $\sigma(m)$ is served using Policy 2 and at least two master requests to y occur in $\lambda(m)$ and $\lambda(m)_{xy}$. In the two-item list $\mathrm{TS}(l,p)$ leaves x behind y. We next examine the configuration in the full list.

Again, consider any item z with at most one master request in $\lambda(m)$ and $\lambda(m)_{xy}$, where such a possible master request is served using Policy 2. Let $\sigma(t)$ be the last master request to y in σ. After the service of $\sigma(t)$, item y precedes z in the full list and in the two-item list consisting of y and z. This holds true even if $\sigma(t)$ is served using Policy 2 because at most one master request to z occurs in $\lambda(t)$ and $\lambda(t)_{yz}$ and such a request is served using Policy 2. After $\sigma(t)$ there can be at most one master request to z, issued at some request $\sigma(t')$, with $t' > t$. Again, this request would be served using Policy 2. In the two-item list, $\mathrm{TS}(l,p)$ would keep z behind y because two master requests to y are contained in $\lambda(t')$ and $\lambda(t')_{yz}$.

We conclude that y precedes z in the two-item list immediately before $\sigma(m)$. Since $\mathrm{TS}(l,p)$ is projective on σ' this relation also holds for the full list. Therefore, when $\mathrm{TS}(l,p)$ serves $\sigma(m)$, it will pass z but not y. Again, x is stored behind y in the full list. We have a contradiction to the assumption that projectivity is violated on σ. □

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 $\mathrm{TS}(l,p)$ to the cost of OPT on σ.

A simple observation is that at any time while $\mathrm{TS}(l,p)$ 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 $\mathrm{TS}(l,p)$ are highly correlated with the counter values.

The analysis of $\mathrm{TS}(l,p)$ 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, $\mathrm{TS}(l,p)$'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.*

Proof. First consider the service cost incurred by TS(l, p) on requests to y that are not followed by a master request to y in the phase. This cost is properly charged at the first new master request occuring within the $2l$ extra requests to y appended at the end of the phase.

Next consider item x and let r be the number of requests to x in the phase. First assume that $r < l$. At the beginning of the phase, the counter value $c(x)$ takes any of the values in $\{0, \ldots, l-1\}$ with equal probability $1/l$. Hence, with probability r/l a master request to x occurs in the phase, resulting in a cost of l for TS(l, p) in the pre-refined cost setting. Hence TS(l, p)'s expected pre-refined cost is $r/l \cdot l = r$, and this is an upper bound on TS(l, p)'s actual cost for the references to x.

We next examine the case $r \geq l$ so that at least one master request to x occurs in the phase. If x's counter value is equal to c at the beginning of the phase, then there are exactly c non-master request to x before the first master request. Hence TS(l,p)'s expected actual cost up to the first master request is at most $1/l \cdot \sum_{c=0}^{l-1} c = (l-1)/2$ and the pre-refined cost overcharges by an extra amount of at least $l - (l-1)/2 = (l+1)/2$.

On the other hand, consider the end of the phase. If the counter of x is equal to c at the phase end, then the cost of $l-c$ requests to x is not paid in the pre-refined cost. Each such request may incur a cost of 1. Again the counter value of x takes any of the values in $\{0,\ldots,l-1\}$ with equal probability $1/l$. Thus the expected cost not paid is at most $1/l \cdot \sum_{i=0}^{l-1}(l-c) = (l+1)/2$, which is covered by the extra cost charged at the first master request to x. □

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}.

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.*

Let us first make the following observation, which we will need quite often before proving Proposition 9.

Lemma 10. *Let xyx be three consecutive master requests. Then with probability $1 - p + p^2$ item x precedes y in* TS(l, p)*'s list after the service of the second master request to x. If the next master request goes to y, the probability of* TS(l, p) *exchanging the elements x and y at this request is $1 - 2p + 2p^2$. The analogous statement holds with the roles of x and y interchanged.*

Proof. Consider the second master request to x in xyx. After TS(l, p) has served this request, item x is stored in front of y in the list if and only if (a) the request was served using Policy 1 or (b) the request to x as well as the one to y were both served using Policy 2. The total probability of these events is $p + (1-p)^2 = 1 - p + p^2$. The analogous statement holds for item y on a sequence of master requests yxy. Hence after the service of the last master request to y in the sequence $xyxy$, item y precedes x in TS(l, p)'s list with probability $1 - p + p^2$. Before the service, y was in front of x with probability $1 - (1 - p + p^2) = p - p^2$. We conclude that the probability of an item swap at the last master request to y is $1 - p + p^2 - (p - p^2) = 1 - 2p + 2p^2$. Again, an analogous statement holds for x on a sequence $yxyx$. □

Proof of Proposition 9. Obviously the insertion of l requests to y does not change OPT's cost. We need to verify that TS(l,p)'s cost does not decrease. First consider the cost of references to y. If y is not at the front of TS(l,p)'s list when y_1 is served, the bad cost charged at the next master request to y after x_2 is now due at y_2. Hence it suffices to evaluate the cost charged to x_1 and x_2 before and after the sequence modification. Thereafter item x definitely precedes y in TS(l,p)'s list and further cost does not differ.

If the master request before x_0 is also to x, then y precedes x in TS(l,p)'s list after the service of y_1 if and only if y_1 was served with Policy 1, which happens with probability p. Hence with probability p a base cost of l occurs at x_1. On the other hand, if the master request preceding x_0 is to y, then after the service of y_1, item y precedes x in TS(l,p)'s list with probability $1-p+p^2$, cf. Lemma 10. Observe that $p \leq 1-p+p^2$. Hence with probability at most $1-p+p^2$ a base cost of l is charged at x_1. After the service of x_1, item x precedes y in TS(l,p)'s list with probability $1-p+p^2$, see again Lemma 10. Hence with the latter probability, an additional cost of $l/2$ is assigned to x_1. Moreover, with probability $1-(1-p+p^2) = p(1-p)$ a base cost of l is charged at x_2. In any case an additional cost of $l/2$ is due at x_2 because then x definitely resides at the front of the list. At most one paid exchange is made on x_1x_2, bringing x to the front of the list. In summary, before the sequence adjustment, the total expected cost of x_1x_2 is at most $(1-p+p^2)\cdot 3l/2 + p(1-p)l + l/2 + d = 3l/2 + (1-p+p^2)l/2 + d$.

After the sequence adjustment, item x is definitely at the back of TS(l,p)'s list when the new master request y_2 has been served. Thus a base cost of l is charged at x_1. With probability p request x_1 is served using Policy 1, bringing x to the front of the list. Hence, with probability p an additional cost of $l/2$ is charged at x_1, and with probability $1-p$ a base cost of l is assigned to x_2. Again, x_2 carries an additional cost of $l/2$. Exactly one paid exchange is done at x_1x_2. Thus the total expected cost of x_1x_2 is $l + pl/2 + (1-p)l + l/2 + d = 3l/2 + (2-p)l/2 + d$, which is not smaller than the cost before the adjustment. □

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 11 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 12, 13 and 14 address Type 1 subphases with the preceding master requests. Lemma 12 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 13 and 14 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}.

Lemma 11. 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 12. 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 13. 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 14. 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 15. *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 16. *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 16, 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].

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.

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

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

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 17. *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 18. *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 15, we consider the partial cost model and request sequences referencing two items. In the proof of Theorem 16, 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 19 will be essential in the proofs of Theorems 15 and 16. 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 19. *Let $L(0) = [xy]$. For every online algorithm A and every N, we have in the partial cost model*

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

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 15 and 16 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 15

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 19, 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 19, 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 18 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 16

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 16 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 $w \in W_h = \{1, \ldots, h-1\} \times \{1, 2\} \cup \{h\}$, 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.

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

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

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 19 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 18 ensures that the competitive ratio of A is not smaller than the above expression. Theorem 16 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.

A The minimization problem posed by Theorem 3

We briefly describe how to minimize the maximum of the expressions given in Theorem 3. For the minimization we assume that φ is real-valued. As d tends to infinity such a real value can be approximated arbitrarily closely. Consider the function $\max\{c_1, \ldots, c_6\}$, with c_i as specified in Theorem 3. Figure 1 depicts the max-function we wish to minimize, considering two ranges for p and φ each. (The plot is colored.) A thorough analysis shows indeed, that it is identical to the function $\max\{c_1, \ldots, c_4\}$.

We can first check that if $p = 0$, the minimum is equal to $c = 3$. If $p = 1$, the minimum is $c \approx 2.2808$, which is greater than our new bound of 2.2442. The ratio $c \approx 2.2808$, for $p = 1$, actually recovers an earlier result from [24]. As φ approaches 0 or ∞ the function of the theorem diverges to ∞. Hence we need to determine the minimum for $p \in (0, 1)$ and $\varphi \in (0, \infty)$. From analysis we derive that, for $p \in (0, 1)$ and $\varphi \in (0, \infty)$ to constitute a minimum of a function of the form $\max\{c_1, \ldots, c_6\}$, one of the following three conditions has to be satisfied.

1. (p, φ) is a local minimum of one of the c_i.
2. Two functions c_i and c_j (with $i \neq j$) take the same value at (p, φ) and their derivative at this value is singular.
3. At least three of the functions $c_1, \ldots, c_6$ take the same value at (p, φ).

We can compute all points satisfying the above conditions and the value the maximum attains at these points. The global minimum is obtained if the first three functions in the maximum take the same value. Computing the values (p, φ) where this happens requires solving polynomial equations of degree 4. Interestingly there is a closed algebraic description of these terms, namely

$$p = \frac{41}{16} - \frac{1}{16}\sqrt{\frac{1}{3}(2387 + \frac{9232}{\sqrt[3]{12473 + 12i\sqrt{253641}}} + 16\sqrt[3]{12473 + 12i\sqrt{253641}})}$$
$$- \frac{1}{2}\sqrt{\frac{2387}{96} - \frac{577}{12\sqrt[3]{12473 + 12i\sqrt{253641}}} - \frac{1}{12}\sqrt[3]{12473 + 12i\sqrt{253641}} - \frac{21641}{32\sqrt{\frac{1}{3}(2387 + \frac{9232}{\sqrt[3]{12473+12i\sqrt{253641}}} + 16\sqrt[3]{12473 + 12i\sqrt{253641}})}}}$$
$$\approx 0.45787$$

and $\varphi = \frac{4(2p-1)}{4p^2-9p+3} \approx 1.19390$. Both these are irrational, algebraic, real numbers.

A.1 General properties of the algorithm

A.2 The cost analysis of the algorithm

Lemma 20. *Consider a sequence of master requests yxyxy. Then* TS(l, p)*'s expected cost on the last two requests in this sequence is* $(3 - p + p^2)/2 \cdot l + 2(1 - 2p + 2p^2)d$.

Proof. Consider the last master request to x in $yxyxy$. By Lemma 10, immediately before the request, item y precedes x in TS(l, p)'s list with probability $1 - p + p^2$. Thus at the master request to x TS(l, p) pays a base cost of l with probability $1 - p + p^2$. After the request, with probability $1 - p + p^2$, item x is at the front of TS(l, p)'s list and an additional cost of $l/2$ is charged. At the last master request to y, with probability $p - p^2$ a bad cost of l is charged. This holds true because at the previous master request to y the item was at the front of TS(l, p)'s list with probability $1 - p + p^2$. Again by Lemma 10, at each of the last two master requests an item swap occurs with probability $1 - 2p + 2p^2$. In summary, the total expected cost of TS(l, p) is

$$\left(1 - p + p^2\right) l + \left(1 - p + p^2\right) \tfrac{l}{2} + \left(p - p^2\right) l + 2\left(1 - 2p + 2p^2\right) d = \tfrac{3-p+p^2}{2} l + 2(1 - 2p + 2p^2)d.$$

□

Proof of Proposition 9. Obviously the insertion of l requests to y does not change OPT's cost. We need to verify that TS(l, p)'s cost does not decrease. First consider the cost of references to y. If y is not at the front of TS(l, p)'s list when y_1 is served, the bad cost charged at the next master request to y after x_2 is now due at y_2. Hence it suffices to evaluate the cost charged to x_1 and x_2 before and after the sequence modification. Thereafter item x definitely precedes y in TS(l, p)'s list and further cost does not differ.

If the master request before x_0 is also to x, then y precedes x in TS(l, p)'s list after the service of y_1 if and only if y_1 was served with Policy 1, which happens with probability p. Hence with probability p a base cost of l occurs at x_1. On the other hand, if the master request preceding x_0 is to y, then after the service of y_1, item y precedes x in TS(l, p)'s list with probability $1 - p + p^2$, cf. Lemma 10. Observe that $p \leq 1 - p + p^2$. Hence with probability at most $1 - p + p^2$ a base cost of l is charged at x_1. After the service of x_1, item x precedes y in TS(l, p)'s list with probability $1 - p + p^2$, see again Lemma 10. Hence with the latter probability, an additional cost of $l/2$ is

assigned to x_1. Moreover, with probability $1-(1-p+p^2)=p(1-p)$ a base cost of l is charged at x_2. In any case an additional cost of $l/2$ is due at x_2 because then x definitely resides at the front of the list. At most one paid exchange is made on x_1x_2, bringing x to the front of the list. In summary, before the sequence adjustment, the total expected cost of x_1x_2 is at most $(1-p+p^2)\cdot 3l/2 + p(1-p)l + l/2 + d = 3l/2 + (1-p+p^2)l/2 + d$.

After the sequence adjustment, item x is definitely at the back of TS(l,p)'s list when the new master request y_2 has been served. Thus a base cost of l is charged at x_1. With probability p request x_1 is served using Policy 1, bringing x to the front of the list. Hence, with probability p an additional cost of $l/2$ is charged at x_1, and with probability $1-p$ a base cost of l is assigned to x_2. Again, x_2 carries an additional cost of $l/2$. Exactly one paid exchange is done at x_1x_2. Thus the total expected cost of x_1x_2 is $l + pl/2 + (1-p)l + l/2 + d = 3l/2 + (2-p)l/2 + d$, which is not smaller than the cost before the adjustment. □

For the proofs of Lemmas 11 to 14, the following lemma is useful.

Lemma 21. *Conditioned on λ_{pre} consisting of at least one master request to y,* TS(l,p)*'s expected service cost at this request is at most $l/2$.*

At a second master request to y, if it exists, TS(l,p)*'s expected service cost is at most $l/2+(1-p)l$. On further master requests to y no service cost is charged.*

Proof. Let r be the number of requests to y in λ_{pre}. Furthermore, let c_y denote the value of y's counter at the beginning of λ_{pre} and hence at the beginning of λ. The counter value is uniformly distributed over $\{0,\ldots,l-1\}$; each possible value occurs with probability $1/l$. We distinguish two cases.

Case 1: If $r \geq l$, then at least one master request to y occurs in λ_{pre}. We refer to the first such request by y_1. If $c_y = c$, then there are c non-master requests to y before y_1 and TS(l,p) is charged a service cost of at most c at y_1. If TS(l,p) serves y_1 using Policy 1, which happens with probability p, then item y definitely resides at the front of the algorithm's list after y_1 has been handled. In fact y resides at the front of TS(l,p)'s list immediately before y_1 is served. Recall that in response to a request, an algorithm may first swap items. Thereafter the request is served. Hence TS(l,p)'s expected service cost at y_1 is $\frac{1}{l}\sum_{c=0}^{l-1} c = (l-1)l/(2l) < l/2$.

Suppose that after y_1 another master request to y occurs in λ_{pre}. With probability at most $1-p$, the algorithm TS(l,p) incurs a service cost of l when handling the second master request. After this request, item y is definitely at the front of TS(l,p)'s list and the algorithm pays no further cost in λ_{pre}.

Case 2: Next assume that $r < l$. A master request y_1 to y only occurs in λ_{pre} if $c_y < r$. Hence TS(l,p)'s expected service cost is $\frac{1}{r}\sum_{c=0}^{r-1} c = (r-1)/2 < l/2$. □

Proof of Lemma 11. Consider any Type 2 subphase $x^{2+j}(yx)^k y^{2+i}$, where $i,j,k \geq 0$. A first observation is that in the subphase's suffix y^{2+i}, item y precedes x in TS(l,p)'s list after the service of the second master request to y and no further cost is incurred in this subsequence. Therefore, it suffices to consider the case $i=0$ and analyze $x^{2+j}(yx)^k y^2$. Next consider the prefix x^{2+j}. After the service of the second master request to x, item x resides at the front of TS(l,p)'s list. At any subsequent master request to x, the algorithm TS(l,p) only pays an additional cost of $l/2$ while OPT is charged a cost of l. This cost ratio is smaller than 1. Hence a worst-case cost ratio is attained for $j=0$, and we may focus on the subphase $x^2(yx)^k y^2 = x(xy)^{k+1}y$, for any $k \geq 0$.

In the following we first analyze the subphase assuming that it is preceded by two master requests to y. We distinguish cases depending on the value of k.

Case 1: First assume that $k = 0$, i.e. the subphase is equal to $xxyy$. On the first master request to x, the algorithm TS(l, p) incurs a base cost of l. With probability p, the algorithm TS(l, p) serves the request using Policy 1. Only in this case does the algorithm move x to the front of the list and has to pay an additional cost of $l/2$. This also implies that with probability $1 - p$, the algorithm TS(l, p) incurs a base cost of l at the next master request to x. In any case an additional cost of $l/2$ is due at the second master request to x.

On the first master request to y in the subphase no bad cost is charged. This holds true because y was at the front of TS(l, p)'s list after the service of the last master request to y preceding $xxyy$. If TS(l, p) serves the first master request to y in $xxyy$ with Policy 1, then y moves to the front of the list. Hence with probability $1 - p$ a bad cost of l is charged at the last master request to y in the subphase. A final observation is that TS(l, p) executes one paid exchange when serving the master requests xx as well as one paid exchange when handling yy in the subphase. We conclude that TS(l, p)'s expected cost is at most

$$l + p\tfrac{l}{2} + (1 - p)l + \tfrac{l}{2} + (1 - p)l + 2d = \tfrac{7}{2}l - \tfrac{3}{2}p + 2d.$$

Since OPT is charged a cost of $2l$ for the two master requests to x, the cost ratio c_2 stated in the lemma follows.

Case 2: Next assume that $k > 0$. We first consider $k = 1$, i.e. the subphase is equal to $xxyxyy$. For the first three requests xxy, the argument from the case $k = 0$ carries over, resulting in a service cost of $5/2 \cdot l - pl/2$ and an exchange cost of $d + pd$. Recall that on the master request to y in xxy, item y moves to the front of the list if TS(l, p) serves the request with Policy 1, which happens with probability p. This implies that the next master request to y is charged a bad cost of l with probability $1 - p$. Moreover, the third master request to x in the subphase is assigned a base cost of l with probability p. After that request, the referenced item x is at the front of the list with probability $1 - p + p^2$, by Lemma 10, and an additional cost of $l/2$ is charged in this case. As for the final request to y, with probability $1 - p + p^2$, item y has been at the front of the list at the last master request to y. Hence with probability $p - p^2$ a base cost of l is charged. Thus the total service cost is

$$\begin{aligned} & \tfrac{5}{2}l - p\tfrac{l}{2} + pl + (1 - p + p^2)\tfrac{l}{2} + (1 - p)l + (p - p^2)l = (4 - \tfrac{p^2}{2})l \leq (5 - 2p + \tfrac{p^2}{2})l \\ = \; & \tfrac{7-3p}{2}l + \tfrac{3-p+p^2}{2}l. \end{aligned}$$

Regarding TS(l, p)'s exchange cost, we observe that on the third master request to x in $xxyxyy$, algorithm TS(l, p) moves item x to the front of the list if it serves both the previous master request to y and the current one to x using Policy 1, which happens with probability p^2. If the element x is at the front after the third master request to x which happens with probability $1 - p + p^2$ according to Lemma 10 there is another exchange moving the element y to the front at one of the two master requests to y. In summary, the TS(l, p)'s total item exchange cost is

$$d + pd + p^2d + (1 - p + p^2)d = (2 + 2p^2)d \leq 4(1 - p + p^2)d = 2d + 2(1 - 2p + 2p^2)d.$$

Observe that OPT pays a cost of $3l$ on the three master requests to x in the subphase. Therefore, the ratio of TS(l, p)'s cost to OPT's cost is upper bounded by

$$\frac{\frac{1}{2}(7 - 3p)l + 2d + \frac{1}{2}(3 - p + p^2)l + 2(1 - 2p + 2p^2)d}{2l + l} \leq \max\{c_2, c_4\}.$$

For $k > 1$, on every further pair xy of master requests in the subphase, by Lemma 20, algorithm TS(l,p) incurs an expected cost of $(3-p+p^2)/2 \cdot l + 2(1-2p+2p^2)d$, while OPT has a cost of l. This yields again the cost ratio c_4 stated in the lemma.

Next suppose that the subphase is preceded by the prephase λ_{pre} of the given phase λ. Recall that we assume that the subphase is preceded by two master requests to y. Obviously TS(l,p)'s cost on λ_{pre} is highest if it consists of at least two master requests to y. By Lemma 21, TS(l,p)'s expected service cost on λ_{pre} is at most $l/2 + (1-p)l$. Algorithm TS(l,p) does only one paid exchange in λ_{pre} so that its total expected cost is upper bounded by $l/2 + (1-p)l + d$. We remind the reader that the cost of d incurred by OPT in swapping items at the beginning of λ is charged to λ_{pre}. We conclude that the ratio of TS(l,p)'s expected cost to OPT's cost is upper bounded by c_1.

We finally address the case that our subphase is not preceded by two master requests to y, but by a master request to x or by a pair xy of master requests. In this case the subphase is the first one in the phase, and we analyze it jointly with λ_{pre}. In the subphase only TS(l,p)'s cost on the first three requests xxy in the subphase is affected. After the service of the first master request to x, with a probability $q > p$, item x might reside at the front of TS(l,p)'s. Thus the expected additional cost increases by $(q-p)l/2$, but the expected base cost at the following master request to x decreases by $(q-p)l$. In total there is no cost increase. At the next master request to y a bad cost of l is charged with probability at most $1-p$. Note that at the most recent master request to y, preceding the subphase, y was at the front of the list with probability at least p. On the other hand, λ_{pre} consists of at most one master request to y and, by Lemma 21, TS(l,p)'s expected service cost is at most $l/2$. Again TS(l,p) performs only one paid exchange in λ_{pre} so that its total expected cost does not exceed $l/2 + d$. Compared to the previous case with at least two master requests to y in λ_{pre}, here, the cost saving of $(1-p)l$ in λ_{pre} is at least as high as the extra bad cost in the subphase. □

Proof of Lemma 12. We analyze a Type 1 subphase $x(yx)^k y^{2+i}$, where $k \geq 0$ and $i \geq 0$. By assumption the subphase is preceded by two master requests to y. As in the proof of Lemma 11 we may assume that $i = 0$ and analyze a subphase $x(yx)^k y^2 = (xy)^{k+1} y$.

We first examine the case $k = 0$, i.e. the subphase xyy. At the beginning, y is stored before x in TS(l,p)'s list, due to the two past master requests to y. At the master request to x, TS(l,p) incurs a base cost of l. If the algorithm serves the request using Policy 1, which happens with probability p, item x moves to the front of the list and there is an exchange cost of d as well as an additional cost of $l/2$. At the first master request to y in the subphase no bad cost is charged because y was at the front of the list after the previous master request to y had been handled. At the first master request to y, the item moves to the front of the list if and only if TS(l,p) serves this request as well as the last one to x using Policy 1, which happens with probability p^2.

Now consider the last master request to y in the subphase. After the service of the previous master request to y, item y is at the front of TS(l,p)'s list with probability $1-p+p^2$, see Lemma 10. Hence with probability $p-p^2$, a bad cost of l is charged at the last master request at y and a paid exchange occurs. We remark that this cost analysis also applies to the last master request in the subphase if $k \geq 1$. In summary, if $k = 0$, then TS(l,p)'s total expected cost is $l + pl/2 + pd + p^2 d + (p-p^2)(l+d) = (1 + 3p/2 - p^2)l + 2pd$. Since OPT incurs a cost of l, we obtain the cost ratio c_3 stated in the lemma.

If $k > 1$, then every other pair xy of master requests in $(xy)^{k+1} y$ incurs an expected refined cost of $(3-p+p^2)/2 \cdot l + 2(1-2p+2p^2)d$, see Lemma 20, whereas OPT has a cost of l. We obtain the ratio c_4 given in the lemma.

Finally suppose that the subphase is preceded by λ_{pre}. As in the proof of Lemma 11 we obtain that TS(l,p)'s total expected cost in the prephase is at most $l/2+(1-p)l+d$. Hence the ratio of TS(l,p)'s expected cost to OPT's cost is upper bounded by c_1. □

Proof of Lemma 13. We investigate a Type 1 subphase $x(yx)^k y^{2+i}$, where $k \geq 0$ and $i \geq 0$, that is preceded by master requests to x and y in this order. This implies that the subphase is the first one in the given phase λ and that λ_{pre} contains at most one master request to y. We analyze the subphase together with λ_{pre} assuming that the subphase does contain a master request to y because in this case TS(l,p)'s expected cost is highest.

As in the proof of Lemma 12 we assume $i=0$ and analyze the sequence $y(xy)^{k+1}y$, where the first y is the master request in λ_{pre}. For a proper cost analysis we need to distinguish two cases depending on the master request before the pair xy preceding our subphase.

First assume that the master request before the pair xy is to x. Hence we are faced with a sequence $xxy(xy)^{k+1}y$ and need to evaluate TS(l,p)'s cost on $y(xy)^{k+1}y$. By Lemma 21, TS(l,p)'s expected service cost on the first master request to y is at most $l/2$. Only if the algorithm serves the request with Policy 1 a paid exchange is performed. Thus TS(l,p)'s expected cost on λ_{pre} is upper bounded by $l/2+pd$. In the following we focus on the subphase.

In a first step we consider the case $k=0$. Hence we analyze TS(l,p)'s cost on the subphase xyy that is preceded by xxy. The full string is $xxy\boldsymbol{xyy}$, where the bold letters represent the subphase. On the first master request to x in the subphase, TS(l,p) incurs a base cost of l. Item x resides at the front of TS(l,p)'s list after the prefix xx of master requests have been served. Hence in the subphase xyy, TS(l,p) only performs an item swap on the master request to x if it serves this request as well as the predecessor master request to y with Policy 1. This happens with probability p^2. By Lemma 10, with probability $1-p+p^2$ item x is at the front of TS(l,p)'s list after the master request to x has been served. Thus with probability $1-p+p^2$ an additional cost of $l/2$ is incurred.

At the first master request to y in the subphase, with probability $1-p$ the algorithm TS(l,p) has to pay a bad cost of l. This holds true because after the service of the previous master request to y, item y was at the back of TS(l,p)'s list with probability $1-p$. By Lemma 10, with probability $1-2p+2p^2$ item y moves to the front of the list when TS(l,p) serves the first request to y in the subphase.

Finally, consider the last master request to y in the subphase. Again Lemma 10 implies that with probability $1-p+p^2$ item y is stored at the front of TS(l,p)'s list when the previous master request to y has been handled. Therefore, with probability $p-p^2$, at the last master request to y algorithm TS(l,p) incurs a bad cost of l and performs a paid exchange. We remark that this cost analysis also applies to the last master request to y in any subphase $(xy)^{k+1}y$, where $k \geq 0$.

In summary TS(l,p)'s expected cost on the subphase is

$$\begin{aligned} & l+(1-p+p^2)\tfrac{l}{2}+(1-p)l+(p-p^2)l+p^2d+(1-2p+2p^2)d+(p-p^2)d \\ = \;& (\tfrac{5}{2}-p)l+p(1-p)\tfrac{l}{2}+(1-p+2p^2)d. \end{aligned}$$

Therefore, TS(l,p)'s total expected cost on the prehase and the subphase is

$$\begin{aligned} & \tfrac{l}{2}+pd+(\tfrac{5}{2}-p)l+p(1-p)\tfrac{l}{2}+(1-p+2p^2)d \\ = \;& \tfrac{l}{2}+(1-p)l+d+(3+p-p^2)\tfrac{l}{2}+2p^2d. \end{aligned}$$

OPT pays an exchange cost of d at the beginning of the phase and a cost of l in the subphase. This yields a cost ratio of at most

$$\max\left\{\frac{l/2+(1-p)l+d}{d}, \frac{(3+p-p^2)l/2+2p^2d}{l}\right\},$$

which is upper bounded by $\max\{c_1, c_5\}$. If $k > 1$, we can simply apply Lemma 20, which ensures that on any further pair xy of master requests in a phase $(xy)^{k+1}y$ the algorithm TS(l,p) has an expected cost of $(3-p+p^2)/2 \cdot l + 2(1-2p+2p^2)d$. Again, OPT is charged a cost of l. Hence the cost ratio is c_4.

It remains to study the case that the master request before the pair xy preceding the considered subphase is to y. Thus we are given a sequence $yxy(xy)^{k+1}y$ and need to evaluate TS(l,p)'s cost on the sequence $y(xy)^{k+1}y$. Again the leading y is the master request in the prephase λ_{pre}. By Lemma 21 TS(l,p) incurs an expected service cost of at most $l/2$ on this master requests. With probability at most $1-p(1-p)$ the algorithm TS(l,p) performs a paid exchange on that request: Observe that if TS(l,p) serves the first y in $yxy(xy)^{k+1}y$ using Policy 1 and the subsequent x using Policy 2, then y is at the front of the list and no item swap takes place in λ_{pre}. Thus TS(l,p)'s total expected cost in the prephase is at most $l/2 + (1-p(1-p))d$.

For the analysis of the subphase we resort again to Lemma 20 which implies that, for each pair xy of master requests in the subphase $(xy)^{k+1}y$, the algorithm TS(l,p) incurs an expected cost of $(3-p+p^2)/2 \cdot l + 2(1-2p+2p^2)d$. The only exception is the first master request to x in the subphase. The analysis of Lemma 20 only assumes a cost of l if the element x is not at the front. The probability of the item not being at the front is $1-p(1-p)$ and hence there is an extra cost of $p(1-p)l$. Moreover, there is extra cost at the last master request to y in the subphase. By Lemma 10, after the service of the preceding master request to y, item y precedes x in TS(l,p)'s list with probability $1-p+p^2$. Therefore with probability $p-p^2$, algorithm TS(l,p) is charged a bad cost of l along with an exchange cost of d at the last master request in the subphase. In summary, the TS(l,p)'s total expected cost in the subphase is $(k+1)((3-p+p^2)/2 \cdot l + 2(1-2p+2p^2)d) + 2p(1-p)l + p(1-p)d$. Including TS$(l,p)$'s expected cost on λ_{pre} we obtain a total cost of at most

$$\begin{aligned} & l/2 + (1-p(1-p))d + (k+1)((3-p+p^2)/2 \cdot l + 2(1-2p+2p^2)d) \\ & +2p(1-p)l + p(1-p)d \\ = \; & l/2 + 2p(1-p)l + d + (k+1)((3-p+p^2)/2 \cdot l + 2(1-2p+2p^2)d). \end{aligned}$$

Since OPT pays an exchange cost of d at the beginning of the phase and is charged a cost of $(k+1)l$ in the subphase, the cost ratio is at most $\max\{c_1, c_4\}$. □

Proof of Lemma 14. We will analyze a Type 1 subphase $x(yx)^k y^{2+i}$, where $i, k \geq 0$, that is preceded by a master request to x. This implies that the subphase is the first one in λ. Moreover no master request to y occurs in λ_{pre} and TS(l,p) does not pay any cost in the prephase. As in the proof of Lemma 12 we assume that $i = 0$. Hence we will analyze a subphase $(xy)^{k+1}y$, for some $k \geq 0$. In the proof of Lemma 11 we studied a related sequence $x(xy)^{k+1}y$ that is preceded by two master requests to y. We can use the cost analysis of the latter sequence. Only the cost of the first requests differ.

More specifically, compared to TS(l,p)'s cost on $x(xy)^{k+1}y$, in our subphase the algorithm saves an additional cost of $pl/2$ on the first master request to x as well as a base cost of $(1-p)l$ on the second master request to x. Moreover, on the first master request to x algorithm TS(l,p) saves an

exchange cost of pd. On the other hand, on the first master request to y there might be a bad cost of $(1-p)l$ if we do not make any assumptions regarding the requests preceding the subphase. Hence the net cost change is $-pl/2-(1-p)l+(1-p)l-pd=-pl/2-pd$.

Consider the case $k=0$, i.e. the subphase xyy. In the proof of Lemma 11 we calculated an expected cost of $\frac{7}{2}l-\frac{3}{2}pl+2d$ for $xxyy$. Hence for xyy we immediately obtain an expected cost of $\frac{7}{2}l-2pl+(2-p)d=l/2+(1-p)l+d+(2-p)l+(1-p)d$. At the beginning of the phase, OPT pays an exchange cost of d. Moreover, it is charged a cost of l on the master request to x. We obtain a cost ratio of

$$\frac{l/2+(1-p)l+d+(2-p)l+(1-p)d}{d+l}\leq\max\{c_1,c_6\}.$$

Next consider the case $k=1$ with the corresponding subphase $xyxyy$. For the related sequence $xxyxyy$, the proof of Lemma 11 shows a cost of $(4-p^2/2)l+(2+2p^2)d$. Thus for the subphase $xyxyy$ we derive a cost of $(4-p^2/2)l+(2+2p^2)d-p/2\cdot l-pd=l/2+(1-p)l+d+(5/2+p/2-p^2/2)l+(1-p+2p^2)d$. Now OPT does a paid exchange at the beginning of the phase and is charged a cost of $2l$ for the two master requests to x. Hence the cost ratio is upper bounded by

$$\max\left\{\frac{l/2+(1-p)l+d}{d},\frac{(5/2+p/2-p^2/2)l+(1-p+2p^2)d}{2l}\right\}.$$

Simple algebraic manipulations show that the last expression is upper bounded by $\frac{3-p+p^2}{2}+\frac{2(1-2p+2p^2)}{\varphi}$. Hence the cost ratio is upper bounded by $\max\{c_1,c_4\}$.

Finally, if $k>1$, on any further pair xy of master requests in $(xy)^{k+1}y$ the algorithm $\mathrm{TS}(l,p)$ incurs an expected cost of $\frac{3-p+p^2}{2}l+2(1-2p+2p^2)d$ while OPT is charged a cost of l. The lemma follows. □

Remark: For the sake of a cleaner upper bound, we made a few concessions that vanish for large d. In the proof of Lemma 21 it would be sufficient to charge the first master request to y a service cost of $(l-1)/2$. This would reduce $\mathrm{TS}(l,p)$'s strict competitiveness in the prephase by $1/(2d)$. Similarly in Definition 7, specifying the refined cost setting, we could replace the additional cost of $l/2$ by $(l-1)/2$. Thereby, in Theorem 3 all cost ratios c_i, for $i>1$, would reduce by up to $1/l$. As d and hence l grows, these savings vanish. On the other hand, we could consider the more classical service model where, in response to a request, an algorithm must first serve the request and may then perform item swaps. Then in the proof of Lemma 21, a cost of $(l+1)/2$ needs to be charged to the first master request to y. In the refined cost setting, an additional cost of $(l+1)/2$ has to be assigned. Consequently, the cost ratios in Theorem 3 increase by a additive values of up to $1/(2d)$ and $1/l$, respectively. Again, for large d and l, these terms vanish.

B Development of the lower bounds

B.1 Preliminaries

Proof of Lemma 18. Suppose that A is $(c-\delta)$-competitive for some $\delta>0$. Then there exists a constant α, possibly depending on the starting list $L(0)$, such that for every request sequence σ

$$C_A(\sigma)\leq(c-\delta)C_{\mathrm{OPT}}(\sigma)+\alpha$$

holds. If A is a randomized algorithm, then $C_A(\sigma)$ denotes its expected cost on σ. For $\sigma \in \mathcal{S}_N$, we have $C_{\mathrm{OPT}}(\sigma) \geq \lfloor \frac{N}{2} \rfloor$ in both the partial cost and the full cost model. This holds true because two adjacent segments in σ reference different items and at least one of them cannot be stored at the front of A's list. Hence $C_{\mathrm{OPT}}(\sigma) \geq \lfloor \frac{N}{2} \rfloor \xrightarrow{N\to\infty} \infty$. Using this fact in the above inequality, we obtain, for every starting list $L(0)$,

$$\lim_{N\to\infty} \frac{\mathbf{E}_{\sigma\sim\mathcal{S}_N}\left[C_A(\sigma)\right]}{\mathbf{E}_{\sigma\sim\mathcal{S}_N}\left[C_{\mathrm{OPT}}(\sigma)\right]} \leq c - \delta + \lim_{N\to\infty} \frac{\alpha}{C_{\mathrm{OPT}}(\sigma)} = c - \delta.$$

On the other hand, since A is c-uncompetitive, there exists a starting list such that

$$\lim_{N\to\infty} \frac{\mathbf{E}_{\sigma\sim\mathcal{S}_N}\left[C_A(\sigma)\right]}{\mathbf{E}_{\sigma\sim\mathcal{S}_N}\left[C_{\mathrm{OPT}}(\sigma)\right]} \geq c - \frac{\delta}{2}$$

holds, and we obtain a contradiction. □

B.2 Lower bounding the cost of every online algorithm

Proof of Lemma 19. 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 σ" $\sim S_{N-2}$ 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 σ" of the sequence σ

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

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 $\mathbf{E}_{\sigma\sim\mathcal{S}_N}[C_B^{\mathrm{part}}(\sigma)] \geq C$. 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

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

In total we have

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

The last inequality is equivalent to $C \geq dN$. Again, we have reached the desired contradiction. □

B.3 Full proof of Theorem 15

The challenge in the proof of Theorem 15 is to upper bound the expected cost incurred by OPT on request sequences generated according to $\mathcal{S}_N$. Recall that we consider request sequences referencing two items. We are given an initial list $L(0) = [xy]$. The probability distribution $\mathcal{S}_N$ specifies request sequences consisting of N segments that reference in turn x and y. In the following, given a sequence σ' and a item $z \in \{x, y\}$, let $|\sigma'|_z$ denote the number of requests to z in σ'.

Instead of the true optimal offline algorithm OPT we will work with a simpler offline algorithm O that we will exactly specify later. Loosely speaking, at any time, O moves an item $z \in \{x, y\}$ to the front of its list if there is a prefix σ' of future requests such that $|\sigma'|_z = |\sigma'|_{z'} + 2d$ and there is no prefix σ'' of σ' with $|\sigma'|_z < |\sigma'|_{z'}$. Here z' is the other item of the pair $\{x, y\}$. In addition, in the cost analysis, we will consider the algorithm $\bar{O}$ that always keeps its list in opposite order as the algorithm O. Hence on each request exactly one of the two algorithms has a service cost of 1 and both algorithms perform the same number of paid exchanges. It turns out that the cost of both algorithms allows us to bound the optimum offline cost.

Representing expected offline cost

Given a request sequence σ, let $\tilde{C}_O(\sigma)$ and $\tilde{C}_{\bar{O}}(\sigma)$ denote the cost incurred by O and $\bar{O}$ for serving requests only. Furthermore, $D_O(\sigma)$ is the cost incurred by O in performing paid exchanges. Define

$$K(\sigma) = \tilde{C}_{\bar{O}}(\sigma) - \tilde{C}_O(\sigma) - 2D_O(\sigma) = \tilde{C}_{\bar{O}}(\sigma) - C_O(\sigma) - D_O(\sigma).$$

We will consider the variable $K(\sigma)$, for a sequence $\sigma \in \mathcal{S}_N$ as well as subsequences of these. In the former case we will assume that O has starting list $L(0)$, while in the latter case we assume O to have whichever starting list it has upon reaching said subsequence. For σ drawn according to $\mathcal{S}_N$, $K(\sigma)$ is a random variable and we let $E_N(\sigma) = K(\sigma)$.

Lemma 22. *We have*

$$\mathbf{E}_{\sigma\sim\mathcal{S}_N}\left[C_O(\sigma)\right] = dN - \frac{\mathbf{E}[E_N]}{2}.$$

Proof. We first observe that

$$\mathbf{E}_{\sigma\sim\mathcal{S}_N}[\tilde{C}_{\bar{O}}(\sigma) + \tilde{C}_O(\sigma)] = \mathbf{E}_{\sigma\sim\mathcal{S}_N}[|\sigma|] = 2dN.$$

The first equation holds because on each request exactly one of the two algorithms O or $\bar{O}$ has a service cost of 1. The second equation holds because the expected length of each segment in a request sequence generated according to $\mathcal{S}_N$ is $2d$. Given any sequence σ we have

$$2C_O(\sigma) = 2(\tilde{C}_O(\sigma) + D_O(\sigma)) = \tilde{C}_{\bar{O}}(\sigma) + \tilde{C}_O(\sigma) - K(\sigma).$$

Passing over to expected values we obtain

$$2\mathbf{E}_{\sigma\sim\mathcal{S}_N}[C_O(\sigma)] = \mathbf{E}_{\sigma\sim\mathcal{S}_N}[\tilde{C}_{\bar{O}}(\sigma) + \tilde{C}_O(\sigma)] - \mathbf{E}_{\sigma\sim\mathcal{S}_N}[K(\sigma)] = 2dN - \mathbf{E}[E_N].$$

The lemma follows by dividing the equation by 2. □

Our main goal in the remainder of this section is to compute $\frac{\mathbf{E}[E_N]}{2}$. Specifically, we will prove the following lemma.

Lemma 23. *There exists an algorithm O such that* $\lim_{N\to\infty} \frac{\mathbf{E}[E_N]}{N} \geq \frac{2d(2d-1)}{4d-1}$.

The proof of this lemma requires some work. Given the lemma, we can finish Theorem 15.

Proof of Theorem 15. Let A be any online algorithm. By Lemma 19 and Lemma 22 we have

$$\frac{\mathbf{E}_{\sigma\sim\mathcal{S}_N}\left[C_A^{\text{part}}(\sigma)\right]}{\mathbf{E}_{\sigma\sim\mathcal{S}_N}\left[C_O^{\text{part}}(\sigma)\right]} \geq \frac{dN}{dN - \frac{\mathbf{E}[E_N]}{2}} = \frac{d}{d - \frac{\mathbf{E}[E_N]}{2N}}.$$

Now using Lemma 23 we get

$$\lim_{N\to\infty} \frac{\mathbf{E}_{\sigma\sim\mathcal{S}_N}\left[C_A^{\text{part}}(\sigma)\right]}{\mathbf{E}_{\sigma\sim\mathcal{S}_N}\left[C_O^{\text{part}}(\sigma)\right]} \geq \frac{d}{d - \frac{2d(2d-1)}{2(4d-1)}} = \frac{2d(4d-1)}{2d(4d-1) - 2d(2d-1)} = \frac{4d-1}{2d} = 2 - \frac{1}{2d},$$

i.e. the algorithm A is $\left(2 - \frac{1}{2d}\right)$-uncompetitive against O on two-item sequences in the partial cost model. Lemma 18 finishes the proof. □

Computing $\mathbf{E}[E_N]$

It remains to prove Lemma 23. In order to evaluate $\mathbf{E}[E_N]$, we have to partition a request sequence drawn according to our probability distribution into phases. Moreover, we have to extend the sampling process so that sequences ending with a complete phase are generated.

We first describe the phase partitioning, for a given σ generated by to our probability distribution. The first phase starts with the first request in σ. Any other phase starts at the end of the previous phase. Every phase λ consists of subphases $\mu_1, \ldots, \mu_l$, for some $l \geq 1$, so that $\lambda = \mu_1 \ldots \mu_l$. 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.

We state some properties of the subphases. Obviously, each subphase is either a zero-subphase or an up-subphase but cannot be both. Consider μ_1 and again let z be the current item when λ and thus μ_1 starts. While μ_1 grows, there is always a surplus of requests to z in μ_1 until event (1) occurs. Thus if μ_1 is a zero-subphase, it ends with a segment referencing z', and z is again the current item of the sampling process when the next subphase starts. Inductively, z is always the current item in our sampling algorithm when a new subphase in the given phase λ starts. On the other hand, the last subphase of λ ends with a segment of requests to z. Thus the current item an the beginning of the next phase is z'

Finally, we extend our sampling process. In order to sample a request sequence according to $\mathcal{S}_N$, we sample a *dominating sequence* $\tilde{\sigma}$ according to the process described in Algorithm 2 with the following properties: Sequence $\tilde{\sigma}$ consists of at least N segments and ends with a complete phase, i.e. the last subphase is an up-subphase. The generated request sequences induce a related probability distribution, which we denote by $\tilde{\mathcal{S}}_N$. Now let σ be the longest prefix of $\tilde{\sigma}$ consisting of at least N segments. It is immediate that σ is distributed according to $\mathcal{S}_N$.

We are ready to define our offline algorithm O. We first consider $\tilde{\sigma}$ and let $\tilde{O}$ be the algorithm that acts as follows. Initially, $\tilde{O}$ starts with the list $L(0) = [xy]$. At the beginning of each phase λ, the algorithm $\tilde{O}$ has the current item z, referenced by the first segment in λ, at the back of its list. The algorithm moves z to the front of its list at the beginning of the unique up-subphase ending λ. No further exchanges are made. Algorithm O acts on σ like $\tilde{O}$ does on this prefix of $\tilde{\sigma}$.

The next lemma shows that the costs of O and $\tilde{O}$ on σ and $\tilde{\sigma}$, respectively, are closely related. For $\tilde{\sigma}$ sampled according to $\tilde{\mathcal{S}}_N$, consider $K(\tilde{\sigma})$ where in the latter cost variable algorithm O is replaced by $\tilde{O}$. Let $\tilde{E}_N = K(\tilde{\sigma})$.

Lemma 24. *We have* $\mathbf{E}[E_N] = \mathbf{E}[\tilde{E}_N] - \Theta(d)$.

Proof. Let $\tilde{\sigma} = \sigma\sigma'$. Then we have $\mathbf{E}[E_N] = \mathbf{E}[\tilde{E}_N] - \mathbf{E}[K(\sigma')]$. Observe that $\tilde{O}$ only does a paid exchange on σ', moving the current item z to the front of its list, if the entire up-subphase ending $\tilde{\sigma}$ and consisting of at least $2d$ requests to z is contained in σ'. On each request in σ' exactly one of the algorithms $\tilde{O}$ and the one keeping the list in reverse order, has a service cost of 1. Thus the definition of $K(\cdot)$ implies

$$0 \leq K(\sigma') \leq |\sigma'|$$

and it suffices to show that $\mathbf{E}[|\sigma'|]$ is in $O(d)$.

The string σ' consists of segments, each having an expected length of $2d$. Again, let z be current item in the sampling process when the last phase in $\tilde{\sigma}$ starts. The phase definitely ends if there is a segment of requests to z having length at least $2d$, which happens with probability $(1-p)^{2d-1}$. Thus in σ' the expected number of segments to z is at most $1/(1-p)^{2d-1}$, and each such segment is preceded by a segment to the other item z'. Assuming that the last segment to z consists of more than $2d$ requests, the expected number of requests following the first $2d$ ones is $2d$. We conclude

$$\mathbf{E}[|\sigma'|] \leq \left(\frac{2}{(1-p)^{2d-1}} + 1\right) \cdot 2d$$

and the last expression is $O(d)$. □

Our phase partitioning induces a probability distribution of subsequences of requests representing a phase. Let $\mathcal{P} = \mathcal{P}[z]$ denote this probability distribution, assuming that z is the current item in the sampling process when the phase starts. Given $\lambda \sim \mathcal{P}$ we set $C(\lambda) = K(\lambda)$ where we assume in the definition of K that the algorithms O and $\tilde{O}$ have the current element z at the back of their lists. Moreover, let $R(\lambda)$ be the number of segments in λ. We abbreviate $C = C(\lambda)$ and and $R = R(\lambda)$. We next put these random variables into context.

For $N \leq 0$ we obviously have $\tilde{E}_N = 0$. For $N > 0$, let $R[N]$ and $C[N]$ denote random variables which are independent and identically distributed to R and C, respectively. We can derive the following recursion:

$$\tilde{E}_N = \tilde{E}_{N-R[N]} + C[N].$$

Indeed, the length of the first phase is distributed according to R (respectively $R[N]$) while its cost is distributed according to C (respectively $C[N]$). If we have $R[N] = r$, the sequence σ' following the first phase is distributed according to $\tilde{\mathcal{S}}_{N-r}$. Inductively we have $\mathbf{E}[K(\sigma')] = \mathbf{E}[\tilde{E}_{N-r}]$. Note that this is vacuously true if $R[N] = r \geq N$ holds.

Lemma 25. *Let R and C be two (not necessarily independent) random variables with values in the positive natural numbers and bounded expected values. Given N, let $R[N]$ and $C[N]$ denote copies of the random variables R respectively C. Let $\tilde{E}_N = 0$, for $N \leq 0$. Given $N > 0$, recursively define*

$$\tilde{E}_N = \tilde{E}_{N-R[N]} + C[N].$$

Then we have

$$\lim_{N\to\infty} \mathbf{E}\left[\frac{\tilde{E}_N}{N}\right] \geq \frac{\mathbf{E}[C]}{\mathbf{E}[R]}.$$

Proof. Taking the recurrence and passing over to expected values, we obtain $\mathbf{E}[\tilde{E}_N] = \mathbf{E}[\tilde{E}_{N-R[N]}] + \mathbf{E}[C[N]]$. Given $N > 0$, let us denote by R_N^- the random variable

$$R_N^- = \sum_{i=1}^{\lfloor N/\mathbf{E}[R] - N^{3/4} \rfloor} R[i].$$

First observe that if we have $R_N^- < N$, then by repeatedly applying the recurrence relation, the random variable $\tilde{E}_N$ increases at least $\lfloor N/\mathbf{E}[R] - N^{3/4} \rfloor$ times by $\mathbf{E}[C]$, i.e. there holds

$$\mathbf{E}\left[\tilde{E}_N \mid R_N^- < N\right] \geq \left\lfloor N/\mathbf{E}[R] - N^{3/4} \right\rfloor \mathbf{E}[C].$$

Hence we are interested in bounding the probability $P_N^- = \mathbf{P}[R_N^- \geq N]$. The expected value of the random variable R_N^- is simply

$$\mathbf{E}[R_N^-] = \left\lfloor \tfrac{N}{\mathbf{E}[R]} - N^{3/4} \right\rfloor \mathbf{E}[R] \leq N - N^{3/4}\mathbf{E}[R]$$

while its variance is

$$\mathrm{Var}[R_N^-] = \left\lfloor \tfrac{N}{\mathbf{E}[R]} - N^{3/4} \right\rfloor \mathrm{Var}[R] = \Theta(N).$$

The last equation holds because $\mathrm{Var}[R]$ is bounded. This holds true because R takes values in the positive natural numbers and has a bounded expected value. Thus Chebyshev's inequality yields

$$P_N^- = \mathbf{P}[R_N^- \geq N] \leq \mathbf{P}\left[R_N^- \geq \mathbf{E}[R_N^-] + N^{3/4}\mathbf{E}[R]\right] \leq \frac{\mathrm{Var}[R_N^-]}{\mathbf{E}[R]^2 N^{3/2}} = \Theta\left(N^{-1/2}\right) \xrightarrow{N\to\infty} 0.$$

In particular we get

$$\mathbf{E}\left[\frac{\tilde{E}_N}{N}\right] \geq \left(1 - P_N^-\right) \cdot \frac{\mathbf{E}[\tilde{E}_N | R_N^- < N]}{N} \geq \left(1 - P_N^-\right) \cdot \frac{\left\lfloor N/\mathbf{E}[R] - N^{3/4} \right\rfloor \mathbf{E}[C]}{N} \xrightarrow{N\to\infty} \frac{\mathbf{E}[C]}{\mathbf{E}[R]}.$$

□

Corollary 26. *We have*

$$\lim_{N\to\infty} \mathbf{E}\left[\frac{E_N}{N}\right] \geq \frac{\mathbf{E}[C]}{\mathbf{E}[R]}.$$

Proof. This is a consequence of the previous lemma and Lemma 24 according to which the difference between $\mathbf{E}[\tilde{E}_N]$ and $\mathbf{E}[E_N]$ is $\Theta(d)$. □

Our final two tasks consist in determining $\mathbf{E}[C]$ and $\mathbf{E}[R]$. We introduce yet another value: Assume that our sampling algorithm is about to generate a subphase. Let P_0 be the probability that it is a zero-sequence. We will compute that $\mathbf{E}[C] = \frac{2d-1}{1-P_0}$ and $\mathbf{E}[R] = \frac{4d-1}{2d(1-P_0)}$. Hence it is not necessary to know the exact value of P_0. Formally one should note that it is not 0 or 1, which is easy to verify.

Computing C

In order to compute the expected value of C we first need the following fact.

Lemma 27. *Let λ be a phase, then $C(\lambda) = K(\lambda)$ is equal to the total length of all post-subphases in λ.*

Proof. By a slight abuse of notation assume that algorithm O handles λ, although we introduced the algorithm $\tilde{O}$ for the treatment of sequences consisting of complete phases. Both algorithms work the same way. Let z be the current item when λ starts. As argued when introducing the phase partitioning, each subphase starts with a segment of requests to z. Each zero-subphase ends with a segment of requests to the other item z'. The last up-subphase ends with a segment of references to z. At the beginning of λ, algorithm O has z at the back of its list and moves it to the front right before the final up-subphase in λ.

Thus in each post-subphase of λ the algorithm O will not incur any cost while the algorithm $\bar{O}$ has cost of 1 per request. No item exchanges are made while a post-subphase is handled. Let μ be an arbitrary subphase of λ and μ' be the post-subphase of μ. Then we have

$$K(\mu') = \tilde{C}_{\bar{O}}(\mu') - \tilde{C}_O(\mu') - 2D_O(\mu') = |\mu'| - 0 - 2 \cdot 0 = |\mu'|.$$

Let μ'' be the part of μ which is not the post-subphase μ'. We verify that $K(\mu'') = 0$: If μ is a zero-subphase, then $\tilde{C}_{\bar{O}}(\mu'') = \tilde{C}_O(\mu'')$ and $D_O(\mu'') = 0$. Otherwise we have $\tilde{C}_{\bar{O}}(\mu'') = \tilde{C}_O(\mu'') + 2d$ and $D_O(\mu'') = d$. In both cases we get

$$K(\mu'') = \tilde{C}_{\bar{O}}(\mu'') - \tilde{C}_O(\mu'') - 2D_O(\mu'') = 0.$$ □

The length of a post-subphase is distributed geometrically. It has length k, for $k \geq 0$, with probability $p(1-p)^k$. Hence its expected length is $\frac{1-p}{p} = 2d - 1$. The number of subphases is distributed shifted geometrically, namely there is at least one subphase and, for $k \geq 1$, the probability of there being k subphases is exactly $(1 - P_0)P_0^{k-1}$. Therefore, in expectation there are $\frac{1}{1-P_0}$ subphases. We conclude that

$$\mathbf{E}[C] = \frac{2d-1}{1-P_0}.$$

Computing R

Consider a phase λ sampled according to $\mathcal{P}$. Assume w.l.o.g. that x is the current item and y is the other item at the beginning of λ. Given $j > 0$, let Z_j denote the expected number of segments in a subphase conditioned on it starting with j requests to the item x. This variable is interesting because it relates to R, as we can see from the next lemma.

Lemma 28. *We have* $\mathbf{E}[R] = \frac{Z_1}{1-P_0}$.

Proof. Let Z_0 be the number of segments in a subphase conditioned on it being a zero-subphase, and let Z_u be the number of segments conditioned on the subphase being an up-subphase. There holds

$$Z_1 = P_0 Z_0 + (1 - P_0) Z_u.$$

On the other hand, we know that a phase consists of exactly one up-subphase while the number of zero-subphases is geometrically distributed with mean $\frac{P_0}{1-P_0}$. Hence we have

$$\mathbf{E}[R] = \frac{P_0}{1 - P_0} Z_0 + Z_u = \frac{Z_1}{1 - P_0}.$$ □

It remains to compute the values of the Z_j. If a subphase starts with j requests to x, there are two cases. (1) With probability $1 - p$, the next request also goes to x. If $j = 2d - 1$, then the subphase ends and consists of one segment. Otherwise the number of segments is by definition Z_{j+1}. (2) With probability p the next request goes to y. Again, if $j = 1$, the subphase ends and consists of two segments. Otherwise, its expected number of segments is exactly $1 + Z_{2d-j+1}$. Hence we have for $j = 1$:

$$Z_1 = (1 - p) Z_2 + 2p$$

and for $1 < j < 2d - 1$:

$$Z_j = (1 - p) Z_{j+1} + p(1 + Z_{2d-j+1})$$

and for $j = 2d - 1$:

$$Z_{2d-1} = (1 - p) + p(1 + Z_2).$$

One can verify that a solution to this system of equations is given by

$$Z_i = 2 - \frac{i^2 - 2(i - d)}{2d(2d - 1)}$$

and with some basic linear algebra one can see that this solution is unique, hence it must be the value of the Z_j. In particular we have

$$Z_1 = 2 - \frac{1 - 2 + 2d}{2d(2d - 1)} = 2 - \frac{1}{2d} = \frac{4d - 1}{2d}$$

and hence by the previous lemma

$$\mathbf{E}[R] = \frac{Z_1}{1 - P_0} = \frac{4d - 1}{2d(1 - P_0)}.$$

We are ready to prove Lemma 23.

Proof of Lemma 23. By Corollary 26 and the values we computed for $\mathbf{E}[C]$ and $\mathbf{E}[R]$ we obtain

$$\lim_{N \to \infty} \frac{\mathbf{E}[E_N]}{N} \geq \frac{\mathbf{E}[C]}{\mathbf{E}[R]} = \frac{2d(2d-1)}{4d-1}.$$

□

B.4 Full proof of Theorem 16

For the proof of Theorem 16 we first show that it suffices to evaluate algorithms in the partial cost model on request sequences referencing two items. Nonetheless we cannot resort to the cost analysis developed for OPT in Appendix B.3 because it would require an optimal offline algorithm to be projective, which is not the case. Therefore, we present the following lemma.

Lemma 29. *Let B be an offline algorithm that is projective for every request sequence sampled according to $\mathcal{S}_N$, given any $N \in \mathbb{N}$ and an initial list $L(0)$ with n items. If every deterministic online algorithm is c-uncompetitive on two-item sequences against B in the partial cost model, then every deterministic online algorithm is c-uncompetitive against B in the full cost model.*

Proof. Let $L = L(0)$ be a list consisting of n items and $\sigma = \sigma(1), \ldots, \sigma(m)$ be an arbitrary request sequence, generated according to $\mathcal{S}_N$ (Algorithm 2), to be served on L. Consider any algorithm. In the full cost model the service cost of any request $\sigma(t) = x$ differs from that in the partial cost model by an additive value of 1. This additive 1 can be split into $n-1$ portions of value $1/(n-1)$, each of which is charged to one of the $n-1$ pairs $\{x, y\}$, for $y \in L \setminus \{x\}$.

Consider any algorithm $C \in \{A, B\}$. The partial cost $C_C^{\text{part}}(\sigma)$ of C on σ can be expressed as in the proof of Proposition 2. For any pair of distinct items $\{x, y\} \in L$, let $R_C^{xy}(\sigma)$ be the number of requests $\sigma(t)$, $1 \le t \le m$, such that (a) $\sigma(t)$ references x or y and (b) immediately before the service of $\sigma(t)$ the referenced item is behind the other item of $\{x, y\}$ in the current list maintained by C. Let $E_C^{xy}(\sigma)$ be the number of paid exchanges of x and y. Then $C_C^{xy}(\sigma) = R_C^{xy}(\sigma) + d \cdot E_C^{xy}(\sigma)$ is the cost incurred by items x and y while algorithm C serves requests to those items. There holds

$$C_C^{\text{part}}(\sigma) = \sum_{\substack{x,y \in L \\ x \neq y}} C_C^{xy}(\sigma)$$

and with the arguments of the preceding paragraph we obtain

$$C_C^{\text{full}}(\sigma) = \sum_{\substack{x,y \in L \\ x \neq y}} \left(C_C^{xy}(\sigma) + \frac{|\sigma_{xy}|}{n-1} \right), \tag{2}$$

where σ_{xy} is the sequence obtained by projecting σ on the pair $\{x, y\}$, cf. Section 2. This way of expressing full cost was used before, see e.g. [12, 27].

For any $N \in \mathbb{N}$, let $\mathcal{S} = \mathcal{S}_N$ be the probability distribution over request sequences generated according to Algorithm 2. Furthermore, for any pair $x, y \in L$, let N_{xy} be the number of segments referencing x or y in sequences of $\mathcal{S} = \mathcal{S}_N$. Let $\mathcal{S}_{xy}$ be the probability distribution over request sequences generated by Algorithm 2 with the list $L(0)_{xy}$ and N_{xy} as the number of segments to be produced. The offline algorithm B is projective so that $C_B^{xy}(\sigma) = C_B^{\text{part}}(\sigma_{xy})$, for any request sequence σ. Equation (2) yields

$$\begin{aligned}
\mathbf{E}_{\sigma \sim \mathcal{S}} \left[C_B^{\text{full}}(\sigma) \right] &= \mathbf{E}_{\sigma \sim \mathcal{S}} \left[\sum_{x,y \in L} \left(C_B^{xy}(\sigma) + \frac{|\sigma_{xy}|}{n-1} \right) \right] = \sum_{x,y \in L} \mathbf{E}_{\sigma \sim \mathcal{S}} \left[C_B^{xy}(\sigma) + \frac{|\sigma_{xy}|}{n-1} \right] \\
&= \sum_{x,y \in L} \mathbf{E}_{\sigma \sim \mathcal{S}} \left[C_B^{\text{part}}(\sigma_{xy}) + \frac{|\sigma_{xy}|}{n-1} \right].
\end{aligned}$$

Sampling a request sequence σ according to $\mathcal{S}$ and generating the projected sequence σ_{xy} is the same as sampling a sequence according to $\mathcal{S}_{xy}$. Therefore the last equation yields

$$\mathbf{E}_{\sigma\sim\mathcal{S}}\left[C_B^{\text{full}}(\sigma)\right] = \sum_{x,y\in L} \mathbf{E}_{\sigma\sim\mathcal{S}_{xy}}\left[C_B^{\text{part}}(\sigma) + \frac{|\sigma|}{n-1}\right]. \tag{3}$$

Similarly, for the online algorithm A we obtain

$$\mathbf{E}_{\sigma\sim\mathcal{S}}\left[C_A^{\text{full}}(\sigma)\right] = \sum_{x,y\in L} \mathbf{E}_{\sigma\sim\mathcal{S}}\left[C_A^{xy}(\sigma) + \frac{|\sigma_{xy}|}{n-1}\right].$$

Note that there is a randomized algorithm, which we call A_{xy}, such that given $\sigma' \in \mathcal{S}_{xy}$ and the starting list $L(0)_{xy}$ there holds

$$\mathbf{E}[C_{A_{xy}}^{\text{part}}(\sigma')] = \mathbf{E}_{\sigma\sim\mathcal{S}|\sigma_{xy}=\sigma'}[C_A^{xy}(\sigma)].$$

Namely A_{xy} generates a sequence $\sigma \sim \mathcal{S}$ with $\sigma_{xy} = \sigma'$, simulates A on it and keeps its list identically to that of A with respect to x and y. It follows that

$$\mathbf{E}_{\sigma\sim\mathcal{S}}\left[C_A^{\text{full}}(\sigma)\right] = \sum_{x,y\in L} \mathbf{E}_{\sigma\sim\mathcal{S}_{xy}}\left[\mathbf{E}\left[C_{A_{xy}}^{\text{part}}(\sigma)\right] + \frac{|\sigma|}{n-1}\right].$$

Using the last equation and (3) we obtain

$$\begin{aligned}
\frac{\mathbf{E}_{\sigma\sim\mathcal{S}}\left[C_A^{\text{full}}(\sigma)\right]}{\mathbf{E}_{\sigma\sim\mathcal{S}}\left[C_B^{\text{full}}(\sigma)\right]} &\geq \min_{xy}\left\{\frac{\mathbf{E}_{\sigma\sim\mathcal{S}_{xy}}\left[\mathbf{E}\left[C_{A_{xy}}^{\text{part}}(\sigma)\right] + \frac{|\sigma|}{n-1}\right]}{\mathbf{E}_{\sigma\sim\mathcal{S}_{xy}}\left[C_B^{\text{part}}(\sigma) + \frac{|\sigma|}{n-1}\right]}\right\} \\
&\geq \min_{xy}\left\{\frac{\mathbf{E}_{\sigma\sim\mathcal{S}_{xy}}\left[\mathbf{E}\left[C_{A_{xy}}^{\text{part}}(\sigma)\right]\right]}{\mathbf{E}_{\sigma\sim\mathcal{S}_{xy}}\left[C_B^{\text{part}}(\sigma)\right]\left(1 + \frac{8d}{n-1}\right)}\right\}
\end{aligned}$$

if $N_{xy} \geq 2$ holds. The last inequality follows because, for every $\sigma \in \mathcal{S}_{xy}$, there holds $\mathbf{E}_{\sigma\sim\mathcal{S}_{xy}}[|\sigma|] = 2dN_{xy}$ and any algorithm incurs a cost of at least $\lfloor N_{xy}/2 \rfloor \geq N_{xy}/4$, if $N_{xy} \geq 2$. Hence there holds $\mathbf{E}_{\sigma\sim\mathcal{S}_{xy}}[|\sigma|] \leq 8d\mathbf{E}_{\sigma\sim\mathcal{S}_{xy}}[C_B^{\text{part}}(\sigma)]$.

By assumption every deterministic online algorithm is c-uncompetitive on two-item sequences against B in the partial cost model. Hence the same holds true for the randomized algorithm A_{xy}. This implies that every ratio in the above minimum approaches a value of at least $c/(1 + \frac{8d}{n-1})$ as $N \to \infty$. The latter expression is at least $c - \varepsilon$, for any $\varepsilon > 0$, if n is chosen large enough. We conclude that A is c-uncompetitive against B in the full cost model. □

We next define a family of offline algorithms B_h, for any h with $0 < h < 2d$. 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 $z \in L$, the algorithm B_h moves z to the front of its list. Otherwise it does not change the list. Obviously B_h is projective. Hence in view of Lemma 29 it suffices to analyze B_h on two-item sequences. Let $L(0) = [xy]$ be the starting list.

Lemma 30. *We have*

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

where $p = \frac{1}{2d}$. *If we set* $h = \lfloor 2\tilde{h}d \rfloor$ *this term is of the form* $\left(1 + \frac{2\tilde{h}e^{-\tilde{h}}}{e^{-\tilde{h}}-2} + O\left(\frac{1}{d}\right)\right) dN + o(N)$.

Proof. Given $\sigma \sim \mathcal{S}_N$, we need to analyze the expected cost incurred by B_h, where $0 < h < 2d$. 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 sequence $z_t^{\alpha_t}$, 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 see that, for $w = (j, 1)$ or $w = (j, 2)$, we have

$$\lim_{t\to\infty} P(t)_w = P_j := \frac{p(1-p)^{j-1}}{2\sum_{i=1}^{h-1} p(1-p)^{i-1} + (1-p)^{h-1}} = \frac{p(1-p)^{j-1}}{2-(1-p)^{h-1}}.$$

as well as for $w = h$

$$\lim_{t\to\infty} P(t)_h = P_h := \frac{(1-p)^{h-1}}{2-(1-p)^{h-1}}.$$

In particular the expected cost of the algorithm B_h on $\mathcal{S}_N$ can be approximated by

$$\begin{aligned}
\mathbf{E}_{\sigma\sim\mathcal{S}_N}[C_{B_h}(\sigma)] &= \sum_{t=1}^{N}\left(\sum_{j=1}^{h-1} P(t)_{(j,1)} j + P(t)_e d\right) \leq \left(\sum_{j=1}^{h-1} P_j j + P_e d\right) N + o(N) \\
&= \frac{p\sum_{j=1}^{h-1}(1-p)^{j-1} j + (1-p)^{h-1}d}{2-(1-p)^h} N + o(N) \\
&= \frac{\left(\frac{1-(1-p)^h}{p} - h(1-p)^{h-1}\right) + (1-p)^{h-1}d}{2-(1-p)^h} N + o(N).
\end{aligned}$$

We observe that, for $h = \lfloor 2\tilde{h}d \rfloor$, we have $(1-p)^h = \left(1 - \frac{1}{2d}\right)^{\lfloor 2\tilde{h}d \rfloor} = e^{-\tilde{h}} + O\left(\frac{1}{d}\right)$. We similarly have $(1-p)^{h-1} = e^{-\tilde{h}} + O\left(\frac{1}{d}\right)$. Substituting this in the previous term for $\mathbf{E}_{\sigma\sim\mathcal{S}_N}[C_{B_h}(\sigma)]$ yields

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

Table 3: The best choices of h, for small values of d.

$$\begin{aligned}\mathbf{E}_{\sigma\sim\mathcal{S}_N}[C_{B_h}(\sigma)] &= \left(\frac{2\cdot(1-e^{-\tilde{h}})-2\tilde{h}e^{-\tilde{h}}+e^{-\tilde{h}}}{2-e^{-\tilde{h}}}+O\left(\frac{1}{d}\right)\right)dN+o(N)\\ &= \left(1+\frac{2\tilde{h}e^{-\tilde{h}}}{e^{-\tilde{h}}-2}+O\left(\frac{1}{d}\right)\right)dN+o(N).\ \square\end{aligned}$$

Proof of Theorem 16. Lemmas 19, 29 and 30 imply that every online algorithm A is c-uncompetitive against the algorithm B_h in the full cost model, where c is at least

$$\left(1+\frac{2\tilde{h}e^{-\tilde{h}}}{e^{-\tilde{h}}-2}\right)^{-1}$$

as $d\to\infty$. Lemma 18 ensures that the competitive ratio of A is not smaller than the above expression. The bound stated in Theorem 16 is obtained if we set

$$\tilde{h}=W\left(-\frac{1}{2e}\right)+1,$$

i.e. we need to choose $\tilde{h}$ maximal such that $2(\tilde{h}-1)e^{\tilde{h}}=-1$. $\square$

In Table 1 we gave lower bounds, for small values of d. By Lemmas 19, 29 and 30, every online algorithm is c-uncompetitive against B_h in the full cost P^d model, where c is at least

$$\frac{d(2-(1-p)^h)}{\left(\frac{1-(1-p)^h}{p}-h(1-p)^{h-1}\right)+(1-p)^{h-1}d}.$$

The following table shows the best choices of h.