arXiv:2503.17264v2 [cs.DS] 29 Apr 2025

A 3.3904-Competitive Online Algorithm for List Update with Uniform Costs

Mateusz Basiak
University of Wrocław

Marcin Bienkowski
University of Wrocław

Martin Böhm
University of Wrocław

Marek Chrobak
University of California, Riverside

Łukasz Jeż
University of Wrocław

Jiří Sgall
Charles University

Agnieszka Tatarczuk
University of Wrocław

Abstract

We consider the List Update problem where the cost of each swap is assumed to be 1. This is in contrast to the "standard" model, in which an algorithm is allowed to swap the requested item with previous items for free. We construct an online algorithm Full-Or-Partial-Move (*Fpm*), whose competitive ratio is at most 3.3904, improving over the previous best known bound of 4.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases List update, work functions, amortized analysis, online algorithms, competitive analysis

Funding Suppported by Polish National Science Centre grants 2022/45/B/ST6/00559, 2020/39/B/ST6/01679 and 2022/47/D/ST6/02864, National Science Foundation grant CCF-2153723, and by grant 24-10306S of GA ČR.

This work is licensed under the Creative Commons Attribution (CC BY) License.

1 Introduction

The List Update problem. In the online *List Update problem* [3, 21, 22], the objective is to maintain a set of items stored in a linear list in response to a sequence of access requests. The cost of accessing a requested item is equal to its distance from the front of the list. After each request, an algorithm is allowed to rearrange the list by performing an arbitrary number of swaps of adjacent items. In the model introduced by Sleator and Tarjan in their seminal 1985 paper on competitive analysis [26], an algorithm can repeatedly swap the requested item with its preceding item at no cost. These swaps are called *free*. All other swaps are called *paid* and have cost 1 each. As in other problems involving self-organizing data structures [7], the goal is to construct an *online* algorithm, i.e., operating without the knowledge of future requests. The cost of such an algorithm is compared to the cost of the optimal *offline* algorithm; the ratio of the two costs is called the *competitive ratio* and is subject to minimization.

Sleator and Tarjan proved that the algorithm Move-To-Front (*Mtf*), which after each request moves the requested item to the front of the list, is 2-competitive [26]. This ratio is

known to be optimal if the number of items is unbounded. Their work was the culmination of previous extensive studies of list updating, including experimental results, probabilistic approaches, and earlier attempts at amortized analysis (see [15] and the references therein).

As shown in subsequent work, *Mtf* is not unique — there are other strategies that achieve ratio 2, such as *TimeStamp* [2] or algorithms based on work function [9]. In fact, there are infinitely many algorithms that achieve ratio 2 [14, 22].

The uniform cost model. Following [25], we will refer to the cost model of [26] as *standard*, and we will denote it here by LUP_S. This model has been questioned in the literature for not accurately reflecting true costs in some implementations [25, 24, 23, 19], with the concept of free swaps being one of the main concerns.

A natural approach to address this concern, considered in some later studies (see, e.g., [25, 7, 22, 4, 13]), is simply to charge cost 1 for *any* swap. We will call it here the *uniform cost model* and denote it LUP_1. A general lower bound of 3 on the competitive ratio of deterministic algorithms for LUP_1 was given by Reingold *et al.* [25]. Changing the cost of free swaps from 0 to 1 at most doubles the cost of any algorithm, so *Mtf* is no worse than 4-competitive for LUP_1. Surprisingly, no algorithm is known to beat *Mtf*, i.e., achieve ratio lower than 4.[1].

Our main result. To address this open problem, we develop an online algorithm *Full-Or-Partial-Move* (*Fpm*) for LUP_1 with competitive ratio $\frac{1}{8}\cdot(23+\sqrt{17}) \approx 3.3904$, significantly improving the previous upper bound of 4.

Our algorithm *Fpm* remains 3.3904-competitive even for the *partial cost* function, where the cost of accessing location ℓ is $\ell - 1$, instead of the *full cost* of ℓ used in the original definition of List Update [26]. Both functions have been used in the literature, depending on context and convenience. For any online algorithm, its partial-cost competitive ratio is at least as large as its full-cost ratio, although the difference typically vanishes when the list size is unbounded. We present our analysis in terms of partial costs.

Fpm remains 3.3904-competitive also in the dynamic scenario of List Update that allows operations of insertions and deletions, as in the original definition in [26] (see Appendix A).

Technical challenges and new ideas. The question whether ratio 3 can be achieved remains open. We also do not know if there is a *simple* algorithm with ratio below 4. We have considered some natural adaptions of *Mtf* and other algorithms that are 2-competitive for LUP_S, but for all we were able to show lower bounds higher than 3 for LUP_1.

As earlier mentioned, *Mtf* is 4-competitive for LUP_1. It is also easy to show that its ratio is not better than 4: repeatedly request the last item in the list. Ignoring additive constants, the algorithm pays twice the length of the list at each step, while any algorithm that just keeps the list in a fixed order, pays only half the length on the average.

The intuition why *Mtf* performs poorly is that it moves the requested items to front "too quickly". For the aforementioned adversarial strategy against *Mtf*, ratio 3 can be obtained by moving the items to front only *every other time* they are requested. This algorithm, called *Dbit*, a deterministic variant of algorithm *Bit* from [25] and a special case of algorithm *Counter* in [25]; it has been also considered in [22]. In Appendix B, we show that *Dbit* is

[1] The authors of [22] claimed an upper bound of 3 for LUP_1, but later discovered that their proof was not correct (personal communication with S. Kamali).

not better than 4-competitive in the partial cost model, and no better than 3.302-competitive in the full cost model.

One can generalize these approaches by considering a more general class of algorithms that either leave the requested item at its current location or move it to the front. In Appendix B, we show that such a strategy cannot achieve ratio better than 3.25, even for just three items. A naive fix would be, for example, to always move the requested item half-way towards front. This algorithm is even worse: its competitive ratio is at least 6 (see Appendix B).

We also show (see Section 5) via a computer-aided argument that, for LUP_1, the work function algorithm's competitive ratio is larger than 3, even for lists of length 5. This is in contrast to the its performance for the standard model, where it achieves optimal ratio 2 [9].

Our algorithm *Fpm* overcomes the difficulties mentioned above by combining a few new ideas. The first idea is a more sophisticated choice of the target location for the requested item. That is, aside from *full moves* that move the requested items to the list front, *Fpm* sometimes performs a *partial move* to a suitably chosen target location in the list. This location roughly corresponds to the front of the list when this item was requested earlier.

The second idea is to keep track, for each pair of items, of the work function for the two-item subsequence consisting of these items. These work functions are used in two ways. First, they roughly indicate which relative order between the items in each pair is "more likely" in an optimal solution. The algorithm uses this information to decide whether to perform a full move or a partial move. Second, the simple sum of all these pair-based work functions is a lower bound on the optimal cost, which is useful in analyzing the competitive ratio of *Fpm*.

Related work. Better bounds are known for randomized algorithms both in standard model (LUP_S) and uniform cost model (LUP_1). For LUP_S, a long line of research culminated in a 1.6-competitive algorithm [20, 25, 2, 6], and 1.5-lower bound [27]. The upper bound of 1.6 is tight in the class of so-called projective algorithms, whose computation is uniquely determined by their behavior on two-item instances [8]. For LUP_1, the ratio is known to be between 1.5 [4] and 2.64 [25].

It is possible to generalize the uniform cost function by distinguishing between the cost of 1 for following a link during search and the cost of $d \geq 1$ for a swap [25, 4] This model is sometimes called P^d model; in this terminology, our LUP_1 corresponds to P^1. While *Mtf* is 4-competitive for P^1, it does not generalize in an obvious way to $d > 1$. Other known algorithms for the P^d model (randomized and deterministic *Counter*, *RandomReset* and *TimeStamp*) have bounds on competitive ratios that monotonically decrease with growing d [25, 4]. In particular, deterministic *Counter* achieves the ratio 4.56 when d tends to infinity [7] and for the same setting ($d \to \infty$) a recent result by Albers and Janke [4] shows a randomized algorithm *TimeStamp* that is 2.24-competitive.

A variety of List Update variants have been investigated in the literature over the last forty years, including models with lookahead [1], locality of reference [10, 5], parameterized approach [18], algorithms with advice [16], prediction [12], or alternative cost models [19, 23, 24]. A particularly interesting model was proposed recently by Azar *et al.* [13], where an online algorithm is allowed to postpone serving some requests, but is either required to serve them by a specified deadline or pay a delay penalty.

In summary, List Update is one of canonical problems in the area of competitive analysis, used to experiment with refined models of competitive analysis or to study the effects of additional features. This underscores the need to fully resolve the remaining open questions regarding its basic variants, including the question whether ratio 3 is attainable for LUP_1.

2 Preliminaries

Model. An algorithm has to maintain a list of items, while a sequence σ of access requests is presented to an algorithm in an online manner. In each step $t \geq 1$, an algorithm is presented an access request σ^t to an item in the list. If this item is in a location ℓ, the algorithm incurs cost $\ell - 1$ to access it. (The locations in the list are indexed $1, 2, ...$) Afterwards, the algorithm may change the list configuration by performing an arbitrary number of swaps of neighboring items, each of cost 1.

For any algorithm A, we denote its cost for processing a sequence σ by $A(\sigma)$. Furthermore, we denote the optimal algorithm by $\textsc{Opt}$.

Notation. Let $\mathcal{P}$ be the set of all unordered pairs of items. For a pair $\{x, y\} \in \mathcal{P}$, we use the notation $x \prec y$ ($x \succ y$) to denote that x is before (after) y in the list of an online algorithm. (The relative order of x, y may change over time, but it will be always clear from context what step of the computation we are referring to.) We use $x \preceq y$ ($x \succeq y$) to denote that $x \prec y$ ($x \succ y$) or $x = y$.

For an input σ and a pair $\{x, y\} \in \mathcal{P}$, σ_{xy} is the subsequence of σ restricted to requests to items x and y only. Whenever we say that an algorithm serves input σ_{xy}, we mean that an algorithm has to maintain a list of two items, x and y.

2.1 Work Functions

Work functions on item pairs. For each prefix σ of the input sequence, an online algorithm may compute a so-called *work function* W^{xy}, where $W^{xy}(xy)$ (or $W^{xy}(yx)$) is the optimal cost of the solution that serves σ_{xy} and ends with the list in configuration xy (or yx). (Function W^{xy} also has prefix σ as an argument. Its value will be always uniquely determined from context.) The values of W for each step can be computed iteratively using straightforward dynamic programming. Note that the values of W are non-negative integers and $|W^{xy}(xy) - W^{xy}(yx)| \leq 1$.

Modes. For a pair $\{x, y\} \in \mathcal{P}$, we define its *mode* depending on the value of the work function W^{xy} in the current step and the mutual relation of x and y in the list of an online algorithm. In the following definition we assume that $y \prec x$.

- Pair $\{x, y\}$ is in mode α if $W^{xy}(yx) + 1 = W^{xy}(xy)$.
- Pair $\{x, y\}$ is in mode β if $W^{xy}(yx) = W^{xy}(xy)$.
- Pair $\{x, y\}$ is in mode γ if $W^{xy}(yx) - 1 = W^{xy}(xy)$.

For an illustration of work function evolution and associated modes, see Figure 1.

If a pair $\{x, y\}$ is in mode α, then the minimum of the work function W^{xy} is at configuration yx, i.e., the one that has y before x. That is, an online algorithm keeps these items in a way that "agrees" with the work function. Note that α is the initial mode of all pairs. Conversely, if a pair $\{x, y\}$ is in state γ, then the minimum of the work function W^{xy} is at configuration xy. In this case, an online algorithm keeps these items in a way that "disagrees" with the work function.

2.2 Lower Bound on OPT

Now we show how to use the changes in the work functions of item pairs to provide a useful lower bound on the cost of an optimal algorithm. The following lemma is a standard and

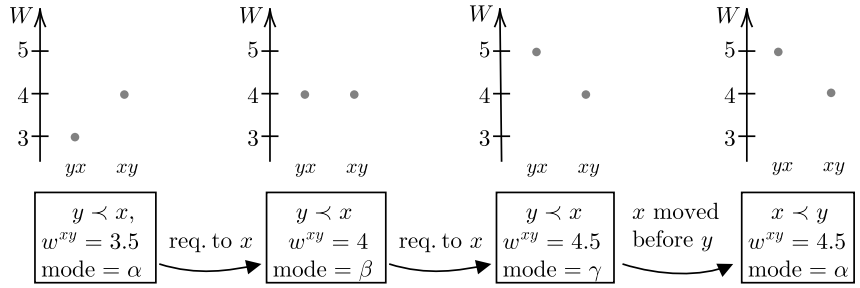


Figure 1 The evolution of the work function W^{xy}. Initially, W^{xy} has its minimum in the state yx and $y \prec x$ (in the list of an online algorithm). Thus, the mode of the pair $\{x, y\}$ is α. Next, because of the two requests to x, the value of $W^{xy}(yx)$ is incremented, while the value at xy remains intact. The mode is thus changed from α to β and then to γ. Finally, when an algorithm moves item x before y, the mode of the pair $\{x, y\}$ changes to α. The value of w^{xy} (the average of $W^{xy}(xy)$ and $W^{xy}(yx)$) increases by $\frac{1}{2}$ whenever the mode changes due to a request.

straightforward result of the list partitioning technique [9].[2]

▶ **Lemma 1.** *For every input sequence σ, it holds that $\sum_{\{x,y\}\in\mathcal{P}} \mathrm{OPT}(\sigma_{xy}) \le \mathrm{OPT}(\sigma)$.*

Averaging work functions. We define the function w^{xy} as the average value of the work function W^{xy}, i.e.,

$$w^{xy} \triangleq \tfrac{1}{2} \cdot \big(W^{xy}(xy) + W^{xy}(yx) \big).$$

We use w_t^{xy} to denote the value of w^{xy} after serving the first t requests of σ, and define $\Delta_t w^{xy} \triangleq w_t^{xy} - w_{t-1}^{xy}$. The growth of w^{xy} can be related to $\mathrm{OPT}(\sigma)$ in the following way.

▶ **Lemma 2.** *For a sequence σ consisting of T requests, it holds that $\sum_{t=1}^{T} \sum_{\{x,y\}\in\mathcal{P}} \Delta_t w^{xy} \le \mathrm{OPT}(\sigma)$.*

Proof. We first fix a pair $\{x, y\} \in \mathcal{P}$. We have $w_0^{xy} = \frac{1}{2}$ and $w_T^{xy} \le \mathrm{OPT}(\sigma_{xy}) + \frac{1}{2}$. Hence, $\sum_{t=1}^{T} \Delta_t w^{xy} = w_T^{xy} - w_0^{xy} \le \mathrm{OPT}(\sigma_{xy})$. The proof follows by summing over all pairs $\{x, y\} \in \mathcal{P}$ and invoking Lemma 1. ◀

Pair-based OPT. Lemma 2 gives us a convenient tool to lower bound $\mathrm{OPT}(\sigma)$. We define the cost of *pair-based* OPT in step t as $\sum_{\{x,y\}\in\mathcal{P}} \Delta_t w^{xy}$. For a given request sequence σ, the sum of these costs over all steps is a lower bound on the actual value of $\mathrm{OPT}(\sigma)$.

On the other hand, we can express $\Delta_t w^{xy}$ (and thus also the pair-based OPT) in terms of the changes of the modes of item pairs. See Figure 1 for an illustration.

▶ **Observation 3.** *If a pair $\{x, y\}$ changes its mode due to the request in step t then $\Delta_t w^{xy} = \frac{1}{2}$, otherwise $\Delta_t w^{xy} = 0$.*

3 Algorithm Full-Or-Partial-Move

For each item x, FPM keeps track of an item denoted θ_x and called the *target* of x. At the beginning, FPM sets $\theta_x = x$ for all x. Furthermore, at each time, FPM ensures that $\theta_x \preceq x$.

[2] There are known input sequences on which the relation of Lemma 1 is not tight [9].

The rest of this section describes the overall strategy of algorithm *FPM*. Our description is top-down, and proceeds in three steps:

- First we describe, in broad terms, what actions are involved in serving a request, including the choice of a move and the principle behind updating target items.
- Next, we define the concept of states associated with item pairs and their potentials.
- Finally, we explain how algorithm *FPM* uses these potential values to decide how to adjust the list after serving the request.

This description will fully specify how *FPM* works, providing that the potential function on the states is given. Thus, for any choice of the potential function the competitive ratio of *FPM* is well-defined. What remains is to choose these potential values to optimize the competitive ratio. This is accomplished by the analysis in Section 4 that follows.

Serving a request. Whenever an item z^* is requested, *FPM* performs the following three operations, in this order:

1. *Target cleanup.* If z^* was a target of another item y (i.e., $\theta_y = z^*$ for $y \neq z^*$), then θ_y is updated to the successor of z^*. This happens for all items y with this property.
2. *Movement of z^*.* *FPM* executes one of the two actions: a *partial move* or a *full move*. We will explain how to choose between them later.
   - In the partial move, item z^* is inserted right before θ_{z^*}. (If $\theta_{z^*} = z^*$, this means that z^* does not change its position.)
   - In the full move, item z^* is moved to the front of the list.
3. *Target reset.* θ_{z^*} is set to the front item of the list.

It is illustrative to note a few properties and corner cases of the algorithm.

- Target cleanup is executed only for items following z^*, and thus the successor of z^* exists then (i.e., *FPM* is well defined).
- If $\theta_{z^*} = z^*$ and a partial move is executed, then z^* is not moved, but the items that targeted z^* now target the successor of z^*.
- For an item x, the items that precede θ_x in the list were requested (each at least once) after the last time x had been requested.

Modes, flavors and states. Fix a pair $\{x, y\}$ such that $y \prec x$, and thus also $\theta_y \preceq y \prec x$. This pair is assigned one of four possible *flavors*, depending on the position of θ_x (cf. Figure 2):

- flavor d (**d**isjoint): if $\theta_y \preceq y \prec \theta_x \preceq x$,
- flavor o (**o**verlapping): if $\theta_y \prec \theta_x \preceq y \prec x$,
- flavor e (**e**qual): if $\theta_y = \theta_x \preceq y \prec x$,
- flavor n (**n**ested): if $\theta_x \prec \theta_y \preceq y \prec x$

For a pair $\{x, y\}$ that is in a mode $\xi \in \{\alpha, \beta, \gamma\}$ and has a flavor $\omega \in \{d, o, e, n\}$, we say that the *state* of $\{x, y\}$ is ξ^ω. Recall that all pairs are initially in mode α, and note that their initial flavor is d. That is, the initial state of all pairs is α^d.

We will sometimes combine the flavors o with e, stating that the pair is in state ξ^{oe} if its mode is ξ and its flavor is o or e. Similarly, we will also combine flavors n and e.

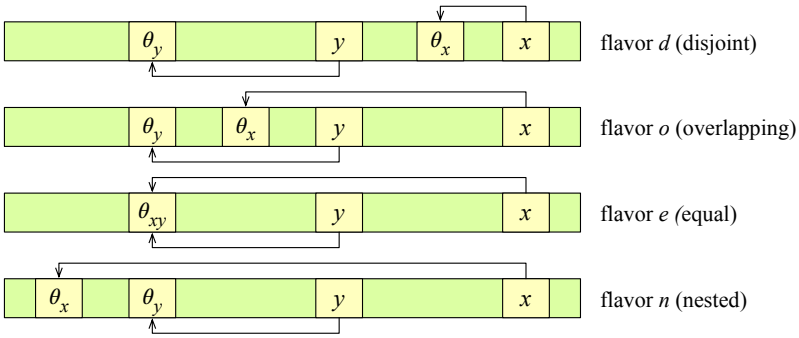


Figure 2 The flavor of a pair $\{x, y\}$ (where $y \prec x$) depends on the position of target θ_x with respect to items $\theta_y \preceq y \prec x$. In the figure for flavor e, we write θ_{xy} for the item $\theta_x = \theta_y$.

Pair potential. To each pair $\{x, y\}$ of items, we assign a non-negative *pair potential* Φ_{xy}. We abuse the notation and use ξ^ω not only to denote the pair state, but also the corresponding values of the pair potential. That is, we assign the potential $\Phi_{xy} = \xi^\omega$ if pair $\{x, y\}$ is in state ξ^ω. We pick the actual values of these potentials only later in Subsection 4.5.

We emphasize that the states of each item pair depend on work functions for this pair that are easily computable in online manner. Thus, FPM may compute the current values of Φ_{xy}, and also compute how their values would change for particular choice of a move.

Choosing the cheaper move. For a step t, let $\Delta_t \mathrm{FPM}$ be the cost of FPM in this step, and for a pair $\{x, y\} \in \mathcal{P}$, let $\Delta_t \Phi_{xy}$ be the change of the potential of pair $\{x, y\}$ in step t. Let z^* denote the requested item. FPM chooses the move (full or partial) with the smaller value of

$$\Delta_t \mathrm{FPM} + \sum_{y \prec z^*} \Delta_t \Phi_{z^* y},$$

breaking ties arbitrarily. Importantly, note that the value that FPM minimizes involves only pairs including the requested item z^* and items currently preceding it.

4 Analysis of Full-Or-Partial-Move

In this section, we show that for a suitable choice of pair potentials, FPM is 3.3904-competitive.

To this end, we first make a few observations concerning how the modes, flavors (and thus states) of item pairs change due to the particular actions of FPM. These are summarized in Table 1, Table 2, and Table 3, and proved in Subsection 4.2 and Subsection 4.3.

Next, in Subsection 4.4 and Subsection 4.5, we show how these changes influence the amortized cost of FPM pertaining to particular pairs. We show that for a suitable choice of pair potentials, we can directly compare the amortized cost of FPM with the cost of OPT.

4.1 Structural Properties

Note that requests to items other than x and y do not affect the mutual relation between x and y. Moreover, to some extent, they preserve the relation between targets θ_x and θ_y, as stated in the following lemma.

▶ **Lemma 4.** *Fix a pair $\{x, y\} \in \mathcal{P}$ and assume that $\theta_y \preceq \theta_x$ (resp. $\theta_y = \theta_x$). If FPM serves a request to an item z^* different than x and y, then these relations are preserved. The relation $\theta_y \prec \theta_x$ changes into $\theta_y = \theta_x$ only when θ_y and θ_x are adjacent and $z^* = \theta_y$.*

Proof. By the definition of FPM, the targets of non-requested items are updated (during the target cleanup) only if they are equal to z^*. In such a case, they are updated to the successor of z^*. We consider several cases.

- If $z^* \notin \{\theta_x, \theta_y\}$, the targets θ_x and θ_y are not updated.
- If $z^* = \theta_y = \theta_x$, both targets are updated to the successor of z^*, and thus they remain equal.
- If $z^* = \theta_y \prec \theta_x$, target θ_y is updated. If θ_y and θ_x were adjacent (θ_y was an immediate predecessor of θ_x), they become equal. In either case, $\theta_y \preceq \theta_x$.
- If $\theta_y \prec \theta_x = z^*$, target θ_x is updated, in which case the relation $\theta_y \prec \theta_x$ is preserved. ◀

4.2 Mode Transitions

Now, we focus on the changes of modes. It is convenient to look first at how they are affected by the request itself (which induces an update of the work function), and subsequently due to the actions of FPM (when some items are swapped). The changes are summarized in the following observation.

▶ **Observation 5.** *Let z^* be the requested item.*

- *Fix an item $y \succ z^*$. The mode transitions of the pair $\{z^*, y\}$ due to request are $\alpha \to \alpha$, $\beta \to \alpha$, and $\gamma \to \beta$. Subsequent movement of z^* does not further change the mode.*
- *Fix an item $y \prec z^*$. Due to the request, pair $\{z^*, y\}$ changes first its mode due to the request in the following way: $\alpha \to \beta$, $\beta \to \gamma$, and $\gamma \to \gamma$. Afterwards, if FPM moves z^* before y, the subsequent mode transitions for pair $\{z^*, y\}$ are $\beta \to \beta$ and $\gamma \to \alpha$.*

4.3 State Transitions

Throughout this section, we fix a step and let z^* be the requested item.

We analyze the potential changes for both types of movements. We split our considerations into three cases corresponding to three types of item pairs. The first two types involve z^* as one pair item, where the second item either initially precedes z^* (cf. Lemma 7) or follows z^* (cf. Lemma 8). The third type involves pairs that do not contain z^* at all (cf. Lemma 9).

While we defined 12 possible states (3 modes × 4 flavors), we will show that α^n, γ^d, and γ^o never occur. This clearly holds at the very beginning as all pairs are then in state α^d.

For succinctness, we also combine some of the remaining states, reducing the number of states to the following six: α^d, β^d, α^{oe}, β^o, β^{ne} and γ^{ne}. (For example, a pair is in state α^{oe} if it is in state α^o or α^e.) In the following we analyze the transitions between them. We start with a simple observation.

▶ **Lemma 6.** *Fix an item $y \neq z^*$. If FPM performs a full move, then the resulting flavor of the pair $\{z^*, y\}$ is d.*

Proof. Item z^* is moved to the front of the list, and its target θ_{z^*} is reset to this item, i.e., $z^* = \theta_{z^*}$. After the movement, we have $z^* \prec \theta_y$. This relation follows trivially if z^* is indeed moved. However, it holds also if z^* was already on the first position: even if $\theta_y = z^*$ before the move, θ_y would be updated to the successor of z^* during the target cleanup. The resulting ordering is thus $\theta_{z^*} = z^* \prec \theta_y \preceq y$, i.e., the flavor of the pair becomes d. ◀

▶ **Lemma 7.** *Fix $y \prec z^*$. The state transitions for pair $\{z^*, y\}$ are given in Table 1.*

Proof. First, suppose FPM performs a full move. The pair $\{z^*, y\}$ is swapped, which changes its mode according to Observation 5 ($\alpha \to \beta$, $\beta \to \alpha$, $\gamma \to \alpha$). By Lemma 6, the flavor of the pair becomes d. This proves the correctness of the transitions in row three of Table 1.

State before move	α^d	β^d	α^{oe}	β^o	β^{ne}	γ^{ne}
State after partial move	β^{ne}	γ^{ne}	β^d or β^o or β^{ne}	α^{oe}	α^d	α^d
State after full move	β^d	α^d	β^d	α^d	α^d	α^d

Table 1 State transitions for pairs $\{z^*, y\}$ where $y \prec z^*$ right before the request to z^*.

State before move	α^d	β^d	α^{oe}	β^o	β^{ne}	γ^{ne}
State after move	α^d	α^d	α^d	α^d	α^d or α^{oe}	β^d or β^o or β^{ne}

Table 2 State transitions for pairs $\{z^*, y\}$ where $z^* \prec y$ right before the request to z^*.

In the rest of the proof, we analyze the case when *Fpm* performs a partial move. We consider three sub-cases depending on the initial flavor of the pair $\{z^*, y\}$. For the analysis of mode changes we will apply Observation 5.

- Before the movement, the flavor of the pair was d, i.e., $\theta_y \preceq y \prec \theta_{z^*} \preceq z^*$.
  The movement of z^* does not swap the pair, i.e., its mode transitions are $\alpha \to \beta$ and $\beta \to \gamma$. As θ_{z^*} is set to the list front, after the movement $\theta_{z^*} \preceq \theta_y \preceq y \prec z^*$, i.e., the flavor of the pair becomes either e or n. This explains the state transitions $\alpha^d \to \beta^{ne}$ and $\beta^d \to \gamma^{ne}$.
- Before the movement, the flavor of the pair was o, i.e., $\theta_y \prec \theta_{z^*} \preceq y \prec z^*$.
  Due to the movement, the pair is swapped, and its mode transitions are $\alpha \to \beta$ and $\beta \to \alpha$. As θ_{z^*} is set to the list front, we have $\theta_{z^*} \preceq \theta_y$. After the movement, $\theta_y \preceq z^* \prec y$, i.e., the flavor of the pair becomes either o or e. This explains the state transitions $\alpha^{oe} \to (\beta^o$ or $\beta^{ne})$ and $\beta^o \to \alpha^{oe}$.
- Before the movement, the flavor of the pair was e or n, i.e., $\theta_{z^*} \preceq \theta_y \preceq y \prec z^*$.
  The movement swaps the pair, and its mode transitions are $\alpha \to \beta$, $\beta \to \alpha$ and $\gamma \to \alpha$. After the movement $z^* \prec \theta_y$, and target θ_{z^*} is set to the list front, which results in $\theta_{z^*} \preceq z^* \prec \theta_y \preceq y$, i.e., the pair flavor becomes d. This explains the state transitions $\alpha^{oe} \to \beta^d$, $\beta^{ne} \to \alpha^d$, and $\gamma^{ne} \to \alpha^d$ ◀

▶ **Lemma 8.** *Fix $y \succ z^*$. The state transitions for pair $\{z^*, y\}$ are given in Table 2.*

Proof. The pair $\{z^*, y\}$ is not swapped due to the request, and thus, by Observation 5, its mode transition is $\alpha \to \alpha$, $\beta \to \alpha$, $\gamma \to \beta$.

If *Fpm* performs a full move, the flavor of the pair becomes d by Lemma 6. This explains the state transitions $\alpha^d \to \alpha^d$, $\beta^d \to \alpha^d$, $\alpha^{oe} \to \alpha^d$, $\beta^o \to \alpha^d$, $\beta^{ne} \to \alpha^d$, and $\gamma^{ne} \to \beta^d$.

In the following, we assume that *Fpm* performs a partial move, and we will identify cases where the resulting pair flavor is different than d. We consider two cases.

- The initial flavor is d, o or e. That is $\theta_{z^*} \preceq z^* \prec y$ and $\theta_{z^*} \preceq \theta_y \preceq y$. During the target cleanup, θ_y may be updated to its successor, but it does not affect these relations, and in particular we still have $\theta_{z^*} \preceq \theta_y$. Thus, when z^* is moved, it gets placed before θ_y. This results in the ordering $\theta_{z^*} \preceq z^* \prec \theta_y \preceq y$, i.e., the resulting flavor is d.
- The initial flavor is n, i.e., $\theta_y \prec \theta_{z^*} \preceq z^* \prec y$. As $\theta_y \neq z^*$, the target θ_y is not updated during the target cleanup. As z^* is moved right before original position of θ_{z^*}, it is placed after θ_y, and the resulting ordering is $\theta_{z^*} \preceq \theta_y \prec z^* \prec y$. That is, the flavor becomes o or e, which explains the state transitions $\beta^{ne} \to \alpha^{oe}$ and $\gamma^{ne} \to (\beta^o$ or $\beta^{ne})$. ◀

State before move	α^d	β^d	α^{oe}	β^o	β^{ne}	γ^{ne}
State after move	α^d	β^d	α^{oe}	β^o or β^{ne}	β^{ne}	γ^{ne}

Table 3 State transitions for pairs $\{x, y\}$ where $x \neq z^*$ and $y \neq z^*$ right before the request to z^*.

▶ **Lemma 9.** *Fix $y \prec x$, such that $x \neq z^*$ and $y \neq z^*$. State transitions for pair $\{x, y\}$ are given in Table 3.*

Proof. The mode of the pair $\{x, y\}$ is not affected by the request to z^*.

The flavor of the pair $\{x, y\}$ depends on mutual relations between x, y, θ_x and θ_y. By Lemma 4, the only possible change is that θ_x and θ_y were different but may become equal: this happens when they were adjacent and the earlier of them is equal to z^*. We consider four cases depending on the initial flavor of the pair.

- The initial flavor was e ($\theta_y = \theta_x \preceq y \prec x$). As θ_y and θ_x are not adjacent, the flavor remains e.
- The initial flavor was d ($\theta_y \preceq y \prec \theta_x \preceq x$). Suppose $\theta_y = z^*$. As $y \neq z^*$, we have $\theta_y = z^* \prec y \prec \theta_x$. That is, θ_x and θ_y are not adjacent, and thus the flavor remains d.
- The initial flavor was o ($\theta_y \prec \theta_x \preceq y \prec x$). In this case it is possible that θ_y and θ_x are adjacent and $z^* = \theta_y$. The flavor may thus change to e or remain o.
- The initial flavor was n ($\theta_x \prec \theta_y \preceq y \prec x$). Similarly to the previous case, it is possible that θ_x and θ_y are adjacent and $z^* = \theta_x$. The flavor may thus change to e or remain n. ◀

4.4 Amortized Analysis

We set $R = \frac{1}{8}(23 + \sqrt{17}) \leq 3.3904$ as our desired competitive ratio.

In the following, we fix a step t in which item z^* is requested. We partition $\mathcal{P}$ into three sets corresponding to the three types of pairs:

- $\mathcal{P}_1^t \triangleq \{\{y, z^*\} : y \prec z^*\}$ (pairs where z^* is the second item, analyzed in Table 1),
- $\mathcal{P}_2^t \triangleq \{\{y, z^*\} : z^* \prec y\}$ (pairs where z^* is the first item, analyzed in Table 2),
- $\mathcal{P}_3^t \triangleq \{\{x, y\} : x \neq z^* \wedge y \neq z^*\}$ (pairs where z^* is not involved, analyzed in Table 3).

For succinctness, wherever it does not lead to ambiguity, we omit subscripts t, i.e., write $\Delta\Phi_{xy}$ and Δw^{xy} instead of $\Delta_t\Phi_{xy}$ and $\Delta_t w^{xy}$. We also omit superscripts t in $\mathcal{P}_1^t$, $\mathcal{P}_2^t$, and $\mathcal{P}_3^t$.

Our goal is to show the following three bounds:

- $\Delta F_{PM} + \sum_{\{x,y\}\in\mathcal{P}_1} \Delta\Phi_{xy} \leq R \cdot \sum_{\{x,y\}\in\mathcal{P}_1} \Delta w^{xy}$,
- $\sum_{\{x,y\}\in\mathcal{P}_2} \Delta\Phi_{xy} \leq R \cdot \sum_{\{x,y\}\in\mathcal{P}_2} \Delta w^{xy}$,
- $\sum_{\{x,y\}\in\mathcal{P}_3} \Delta\Phi_{xy} \leq R \cdot \sum_{\{x,y\}\in\mathcal{P}_3} \Delta w^{xy}$.

Note that the left hand sides of these inequalities correspond to the portions of amortized cost of F_{PM} corresponding to sets $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$, while the right hand sides are equal to R times the corresponding portion of the cost of pair-based O_{PT}. Hence, if we can show the above inequalities for every step t, the competitive ratio of R will follow by simply adding them up.

As we show in the sections that follow, these bounds reduce to some constraints involving state potentials α^d, β^d, α^{oe}, β^o, β^{ne} and γ^{ne}. The bounds for $\mathcal{P}_2$ and $\mathcal{P}_3$, while they involve summations over pairs, can be justified by considering individual pairs and the needed constraints are simple inequalities between state potentials, summarized in the following assumption:

▶ **Assumption 10.** *We assume that*

- $\alpha^d = 0$,
- *all constants* β^d, α^{oe}, β^o, β^{ne} *and* γ^{ne} *are non-negative,*
- $\alpha^{oe} \le \beta^{ne} + \frac{1}{2}R \le \beta^o + \frac{1}{2}R$,
- $\max\{\beta^d, \beta^o\} \le \gamma^{ne} + \frac{1}{2}R$.

The bound for $\mathcal{P}_1$ is most critical (not surprisingly, as it corresponds to requesting the second item of a pair). To analyze this bound, the needed constraints, besides the state potentials also need to involve the numbers of pairs that are in these states. This gives rise to a non-linear optimization problem that we need to solve.

4.4.1 Analyzing pairs from set $\mathcal{P}_1$

The proof of the following lemma is deferred to Subsection 4.5.

▶ **Lemma 11.** *There exist parameters* α^d, β^d, α^{oe}, β^o, β^{ne} *and* γ^{ne}, *satisfying Assumption 10, such that for any step* t, *it holds that* $\Delta F_{PM} + \sum_{\{x,y\}\in\mathcal{P}_1} \Delta\Phi_{xy} \le R \cdot \sum_{\{x,y\}\in\mathcal{P}_1} \Delta w^{xy}$.

4.4.2 Analyzing pairs from set $\mathcal{P}_2$

▶ **Lemma 12.** *For any step* t, *it holds that* $\sum_{\{x,y\}\in\mathcal{P}_2} \Delta\Phi_{xy} \le R \cdot \sum_{\{x,y\}\in\mathcal{P}_2} \Delta w^{xy}$.

Proof. Recall that $\mathcal{P}_2$ contains all pairs $\{z^*, y\}$, such that $z^* \prec y$. Thus, it is sufficient to show that $\Delta\Phi_{z^*y} \le R \cdot \Delta w^{z^*y}$ holds for any such pair $\{z^*, y\}$. The lemma will then follow by summing over all pairs from $\mathcal{P}_2$.

By Assumption 10, we have

$$\alpha^d - \alpha^{oe} \le 0, \tag{1}$$

$$\max\{\alpha^d, \alpha^{oe}\} - \beta^{ne} = \alpha^{oe} - \beta^{ne} \le \tfrac{1}{2}R, \tag{2}$$

$$\max\{\beta^d, \beta^o, \beta^{ne}\} - \gamma^{ne} = \max\{\beta^d, \beta^o\} - \gamma^{ne} \le \tfrac{1}{2}R. \tag{3}$$

We consider two cases depending on the initial mode of the pair $\{z^*, y\}$. In each case, we upper-bound the potential change on the basis of possible state changes of this pair (cf. Table 2).

- The initial mode of $\{z^*, y\}$ is α. By Table 2, this mode remains α, and thus by Observation 3, $\Delta w^{z^*y} = 0$. Then,

$$\begin{aligned} \Delta\Phi_{z^*y} &\le \max\left\{\alpha^d - \alpha^d, \alpha^d - \alpha^{oe}\right\} && \text{(by Table 2)} \\ &= 0 = R \cdot \Delta w^{z^*y}. && \text{(by (1))} \end{aligned}$$

- The initial mode of $\{z^*, y\}$ is β or γ. By Table 2, its mode changes due to the request to z^*, and hence, by Observation 3, $\Delta w^{z^*y} = \frac{1}{2}$. Then,

$$\begin{aligned} \Delta\Phi_{z^*y} &\le \max\{\alpha^d - \alpha^d, \alpha^d - \beta^d, \alpha^d - \alpha^{oe}, \alpha^d - \beta^o, \\ &\qquad \max\{\alpha^d, \alpha^{oe}\} - \beta^{ne}, \\ &\qquad \max\{\beta^d, \beta^o, \beta^{ne}\} - \gamma^{ne}\} && \text{(by Table 2)} \\ &\le \max\{-\beta^d, -\beta^o, \tfrac{1}{2}R, \tfrac{1}{2}R\} && \text{(by } \alpha^d = 0\text{, (1), (2) and (3))} \\ &= \tfrac{1}{2}R = R \cdot \Delta w^{z^*y}. \end{aligned}$$

◀

4.4.3 Analyzing pairs from set $\mathcal{P}_3$

▶ **Lemma 13.** *For any step t, it holds that $\sum_{\{x,y\}\in\mathcal{P}_3} \Delta\Phi_{xy} \le R \cdot \sum_{\{x,y\}\in\mathcal{P}_3} \Delta w^{xy}$.*

Proof. As in the previous lemma, we show that the inequality $\Delta\Phi_{xy} \le R \cdot \Delta w^{xy}$ holds for any pair $\{x,y\} \in \mathcal{P}_3$, i.e., for a pair $\{x,y\}$, such that $x \neq z^*$ and $y \neq z^*$. The lemma will then follow by summing over all pairs $\{x,y\} \in \mathcal{P}_3$.

Possible state transitions of such a pair $\{x,y\}$ are given in Table 3. Hence, such a pair either does not change its state (and then $\Delta\Phi_{xy} = 0$) or it changes it from β^o to β^{ne} (and then $\Delta\Phi_{xy} = \beta^{ne} - \beta^o \le 0$ by Assumption 10). In either case, $\Delta\Phi_{xy} \le 0 \le R \cdot \Delta w^{xy}$. ◀

4.4.4 Proof of R-competitiveness

We now show that the three lemmas above imply that FPM is R-competitive.

▶ **Theorem 14.** *For an appropriate choice of parameters, the competitive ratio of FPM is at most $R = \frac{1}{8}(23 + \sqrt{17}) \le 3.3904$.*

Proof. Fix any sequence σ consisting of T requests. By summing the guarantees of Lemma 11, Lemma 12, and Lemma 13, we obtain that for any step t, it holds that

$$\Delta_t FPM + \sum_{\{x,y\}\in\mathcal{P}} \Delta_t \Phi_{xy} \le R \cdot \sum_{\{x,y\}\in\mathcal{P}} \Delta_t w^{xy}.$$

By summing over all steps, observing that the potentials are non-negative and the initial potential is zero (cf. Assumption 10), we immediately obtain that $FPM(\sigma) \le R \cdot \sum_{t=1}^{T} \sum_{\{x,y\}\in\mathcal{P}} \Delta w^{xy} \le R \cdot OPT(\sigma)$. The second inequality follows by Lemma 2. ◀

4.5 Proof of Lemma 11

Again, we focus on a single step t, in which the requested item is denoted z^*. We let A^d, B^d, A^{oe}, B^o, B^{ne}, C^{ne} be the number of items y preceding z^* such that pairs $\{z^*, y\}$ have states α^d, β^d, α^{oe}, β^o, β^{ne} and γ^{ne}, respectively. Let

$$V \triangleq [A^d, B^d, A^{oe}, B^o, B^{ne}, C^{ne}].$$

Note that $\|V\|_1$ is the number of items preceding z^*, and thus also the access cost FPM pays for the request. Moreover, $A^d + B^d$ is the number of items that precede θ_{z^*}. We use $\odot$ to denote scalar product (point-wise multiplication) of two vectors.

We define three row vectors G_{OPT}, G_{PM}, and G_{FM}, such that

$$\begin{pmatrix} G_{\mathrm{PM}} \\ G_{\mathrm{FM}} \\ G_{\mathrm{OPT}} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix} + \begin{pmatrix} \beta^{ne} & (\gamma^{ne} - \beta^d) & (\max\{\beta^d, \beta^o\} - \alpha^{oe}) & (\alpha^{oe} - \beta^o) & -\beta^{ne} & -\gamma^{ne} \\ \beta^d & -\beta^d & (\beta^d - \alpha^{oe}) & -\beta^o & -\beta^{ne} & -\gamma^{ne} \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Expressing costs as vector products. Recall that to prove Lemma 11, we need to relate the $\mathcal{P}_1$ portion of the amortized cost of FPM, i.e., $\Delta FPM + \sum_{\{x,y\}\in\mathcal{P}_1} \Delta\Phi_{xy}$ and the corresponding portion of the cost of pair-based OPT, i.e., $\sum_{\{x,y\}\in\mathcal{P}_1} \Delta w^{xy}$. In the following two lemmas, we show how to express both terms as vector products.

▶ **Lemma 15.** *It holds that* $\sum_{y \prec z^*} \Delta w^{z^* y} = G_{\mathrm{OPT}} \odot V$.

Proof. The right-hand side of the lemma relation is equal to $\frac{1}{2}(A^d + B^d + A^{oe} + B^o + B^{ne})$. By Observation 5, due to request to z^*:

- $A^d + A^{oe}$ pairs change mode from α to β, and
- $B^d + B^o + B^{ne}$ pairs change mode from β to γ.

By Observation 3, each such mode change contributes $\frac{1}{2}$ to the left hand side of the lemma equation. Note that C^{ne} pairs of mode γ do not change their mode due to the request. ◀

▶ **Lemma 16.** *It holds that* $\Delta F_{PM} + \sum_{y \prec z^*} \Delta \Phi_{z^* y} = G \odot V$, *where*

- $G = G_{\mathrm{PM}}$ *if* F_{PM} *performs a partial move, and*
- $G = G_{\mathrm{FM}}$ *if* F_{PM} *performs a full move.*

Proof. First, assume that F_{PM} performs a partial move. By the definitions of A^d, B^d, A^{oe}, B^o, B^{ne}, and C^{ne}, $\Delta F_{PM} = [1, 1, 2, 2, 2, 2] \odot V$. By Table 1,

$$\begin{aligned}\sum_{y \prec z^*} \Delta \Phi_{z^* y} &\\ &\leq \left[\beta^{ne} - \alpha^d, \gamma^{ne} - \beta^d, \max\{\beta^d, \beta^o, \beta^{ne}\} - \alpha^{oe}, \alpha^{oe} - \beta^o, \alpha^d - \beta^{ne}, \alpha^d - \gamma^{ne}\right] \odot V \\ &= \left[\beta^{ne}, \gamma^{ne} - \beta^d, \max\{\beta^d, \beta^o\} - \alpha^{oe}, \alpha^{oe} - \beta^o, -\beta^{ne}, -\gamma^{ne}\right] \odot V,\end{aligned}$$

where the second equality follows as $\alpha^d = 0$ and $\beta^o \geq \beta^{ne}$ (by Assumption 10). Thus, the lemma holds for a partial move.

Next, assume F_{PM} performs a full move. Then, $\Delta F_{PM} = [2, 2, 2, 2, 2, 2] \odot V$. By Table 1,

$$\begin{aligned}\sum_{y \prec z^*} \Delta \Phi_{z^* y} &= \left[\beta^d - \alpha^d, \alpha^d - \beta^d, \beta^d - \alpha^{oe}, \alpha^d - \beta^o, \alpha^d - \beta^{ne}, \alpha^d - \gamma^{ne}\right] \odot V \\ &= \left[\beta^d, -\beta^d, \beta^d - \alpha^{oe}, -\beta^o, -\beta^{ne}, -\gamma^{ne}\right] \odot V,\end{aligned}$$

where in the second equality we used $\alpha^d = 0$ (by Assumption 10). Thus, the lemma holds for a full move as well. ◀

Recall that F_{PM} is defined to choose the move that minimizes $\Delta F_{PM} + \sum_{y \prec z^*} \Delta \Phi_{z^* y}$.

▶ **Corollary 17.** *It holds that* $\Delta F_{PM} + \sum_{y \prec z^*} \Delta \Phi_{z^* y} = \min\{G_{\mathrm{PM}} \odot V, G_{\mathrm{FM}} \odot V\}$.

Finding Parameters. We may now prove Lemma 11, restated below for convenience.

▶ **Lemma 11.** *There exist parameters* α^d, β^d, α^{oe}, β^o, β^{ne} *and* γ^{ne}, *satisfying Assumption 10, such that for any step* t, *it holds that* $\Delta F_{PM} + \sum_{\{x,y\} \in \mathcal{P}_1} \Delta \Phi_{xy} \leq R \cdot \sum_{\{x,y\} \in \mathcal{P}_1} \Delta w^{xy}$.

Proof. We choose the following values of the parameters:

$$\begin{aligned}&\alpha^d = 0 && \beta^d = \tfrac{1}{16}(5 + 3\sqrt{17}) \approx 1.086 && \alpha^{oe} = 2 \\ &\beta^o = \tfrac{1}{16}(1 + 7\sqrt{17}) \approx 1.866 && \beta^{ne} = \tfrac{1}{16}(9 - \sqrt{17}) \approx 0.305 && \gamma^{ne} = 2\end{aligned}$$

It is straightforward to verify that these values satisfy the conditions of Assumption 10. We note that relation $\alpha^{oe} \leq \beta^{ne} + \frac{1}{2}R$ holds with equality.

By Corollary 17, $\Delta F_{PM} + \sum_{\{x,y\} \in \mathcal{P}_1} \Delta \Phi_{xy} = \Delta F_{PM} + \sum_{y \prec z^*} \Delta \Phi_{z^* y} = \min\{G_{\mathrm{PM}} \odot V, G_{\mathrm{FM}} \odot V\}$. On the other hand, by Lemma 15, $\sum_{\{x,y\} \in \mathcal{P}_1} \Delta w^{xy} = \sum_{y \prec z^*} \Delta w^{z^* y} = G_{\mathrm{OPT}} \odot V$. Hence, it remains to show that $\min\{G_{\mathrm{PM}} \odot V, G_{\mathrm{FM}} \odot V\} \leq R \cdot G_{\mathrm{OPT}} \odot V$.

We observe that

$$G_{\mathrm{PM}} = \tfrac{1}{16} \cdot \left[25 - \sqrt{17},\ \ 43 - 3\sqrt{17},\ 1 + 7\sqrt{17},\ 63 - 7\sqrt{17},\ 23 + \sqrt{17},\ 0\right],$$
$$G_{\mathrm{FM}} = \tfrac{1}{16} \cdot \left[37 + 3\sqrt{17},\ 27 - 3\sqrt{17},\ 5 + 3\sqrt{17},\ 31 - 7\sqrt{17},\ 23 + \sqrt{17},\ 0\right].$$

Let $c = \frac{1}{4}(\sqrt{17} - 1)$ and let $G_{\mathrm{COMB}} = c \cdot G_{\mathrm{PM}} + (1 - c) \cdot G_{\mathrm{FM}}$. Then,

$$G_{\mathrm{COMB}} = \tfrac{1}{16} \cdot \left[23 + \sqrt{17},\ 23 + \sqrt{17},\ 23 + \sqrt{17},\ 23 + \sqrt{17},\ 23 + \sqrt{17},\ 0\right].$$

Now for any vector V,

$$\begin{aligned} \min\{G_{\mathrm{PM}} \odot V, G_{\mathrm{FM}} \odot V\} &\leq c \cdot G_{\mathrm{PM}} \odot V + (1 - c) \cdot G_{\mathrm{FM}} \odot V \\ &= (c \cdot G_{\mathrm{PM}} + (1 - c) \cdot G_{\mathrm{FM}}) \odot V \\ &= G_{\mathrm{COMB}} \odot V = R \cdot G_{\mathrm{OPT}} \odot V, \end{aligned}$$

completing the proof. ◀

5 Final Remarks

The most intriguing question left open in our work is whether competitive ratio of 3 can be achieved. We have shown computationally (see Appendix E) that 3-competitive algorithms exist for lists with up to 6 items.

However, even for short lists the definition of such 3-competitive algorithm remains elusive. For many online problems, the most natural candidate is the generic work function algorithm. This algorithm is 2-competitive in the $\mathsf{LUP}_{\mathrm{S}}$ model [9]. However, our computer-aided calculation of its performance shows that its ratio is larger than 3 already for 5 items (see Appendix E). It is 3-competitive for lists of length up to 4, though.

We do not know whether the analysis of *FPM* is tight. For the specific choice of parameters used in the paper, we verified that *FPM* is 3-competitive for lists of length 3 (see Appendix C), but not better than 3.04-competitive for lists of length 5 (see Appendix D).

The focus of this paper is on the LUP_1 model (also known as P^1); we believe that the setting of $d = 1$ captures the essence and hardness of the deterministic variant. That said, extending the definition and analysis of *FPM* to the P^d model (for arbitrary d) is an interesting open problem that deserves further investigation.

References

1 Susanne Albers. A competitive analysis of the list update problem with lookahead. *Theoretical Computer Science*, 197(1):95–109, 1998. doi:10.1016/S0304-3975(97)00026-1.

2 Susanne Albers. Improved randomized on-line algorithms for the list update problem. *SIAM Journal on Computing*, 27(3):682–693, 1998. doi:10.1137/S0097539794277858.

3 Susanne Albers. Online list update. In *Encyclopedia of Algorithms*, pages 598–601. Springer, 2008. doi:10.1007/978-0-387-30162-4_266.

4 Susanne Albers and Maximilian Janke. New bounds for randomized list update in the paid exchange model. In *37th Int. Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 12:1–12:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.STACS.2020.12.

5 Susanne Albers and Sonja Lauer. On list update with locality of reference. *Journal of Computer and System Sciences*, 82(5):627–653, 2016. doi:10.1016/J.JCSS.2015.11.005.

6 Susanne Albers, Bernhard Von Stengel, and Ralph Werchner. A combined BIT and TIMESTAMP algorithm for the list update problem. *Information Processing Letters*, 56(3):135–139, 1995. doi:10.1016/0020-0190(95)00142-Y.

7 Susanne Albers and Jeffery Westbrook. Self-organizing data structures. In *Online Algorithms, The State of the Art*, pages 13–51. Springer, 1996. doi:10.1007/BFb0029563.

8 Christoph Ambühl, Bernd Gärtner, and Bernhard von Stengel. Optimal lower bounds for projective list update algorithms. *ACM Transactions on Algorithms*, 9(4):1–18, 2013. doi:10.1145/2500120.

9 Eric J. Anderson, Kirsten Hildrum, Anna R. Karlin, April Rasala, and Michael E. Saks. On list update and work function algorithms. In *Proc. 7th Annual European Symposium on Algorithms (ESA)*, pages 289–300. Springer, 1999. doi:10.1007/3-540-48481-7_26.

10 Spyros Angelopoulos, Reza Dorrigiv, and Alejandro López-Ortiz. List update with locality of reference. In *Proc. 8th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 399–410. Springer, 2008. doi:10.1007/978-3-540-78773-0_35.

11 Anonymous Authors. Online list update with uniform costs: Code repository. https://anonymous.4open.science/r/listupdate-C0E6/, 2025.

12 Yossi Azar, Shahar Lewkowicz, and Varun Suriyanarayana. List update with prediction. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 15436–15444. AAAI Press, 2025. doi:10.1609/AAAI.V39I15.33694.

13 Yossi Azar, Shahar Lewkowicz, and Danny Vainstein. List update with delays or time windows. In *Proc. 51st Int. Colloqium on Automata, Languages, and Programming (ICALP)*, pages 15:1–15:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.15.

14 Ran Bachrach, Ran El-Yaniv, and M. Reinstadtler. On the competitive theory and practice of online list accessing algorithms. *Algorithmica*, 32(2):201–245, 2002. doi:10.1007/S00453-001-0069-8.

15 Jon L. Bentley and Catherine C. McGeoch. Amortized analyses of self-organizing sequential search heuristics. *Communications of the ACM*, 28(4):404–411, 1985. doi:10.1145/3341.3349.

16 Joan Boyar, Shahin Kamali, Kim S. Larsen, and Alejandro López-Ortiz. On the list update problem with advice. *Information and Computation*, 253:411–423, 2017. doi:10.1016/J.IC.2016.06.007.

17 Marek Chrobak and Lawrence L. Larmore. The server problem and on-line games. In *On-Line Algorithms, Proceedings of a DIMACS Workshop*, pages 11–64. DIMACS/AMS, 1991. doi:10.1090/DIMACS/007/02.

18 Reza Dorrigiv, Martin R Ehmsen, and Alejandro López-Ortiz. Parameterized analysis of paging and list update algorithms. *Algorithmica*, 71(2):330–353, 2015. doi:10.1007/S00453-013-9800-5.

19 Alexander Golynski and Alejandro López-Ortiz. Optimal strategies for the list update problem under the MRM alternative cost model. *Information Processing Letters*, 112(6):218–222, 2012. doi:10.1016/J.IPL.2011.12.001.

20 Sandy Irani. Two results on the list update problem. *Information Processing Letters*, 38(6):301–306, 1991. doi:10.1016/0020-0190(91)90086-W.

21 Shahin Kamali. Online list update. In *Encyclopedia of Algorithms*, pages 1448–1451. Springer, 2016. doi:10.1007/978-1-4939-2864-4_266.

22 Shahin Kamali and Alejandro López-Ortiz. *A Survey of Algorithms and Models for List Update*, pages 251–266. Springer, 2013. doi:10.1007/978-3-642-40273-9_17.

23 Conrado Martınez and Salvador Roura. On the competitiveness of the move-to-front rule. *Theoretical Computer Science*, 242(1-2):313–325, 2000. doi:10.1016/S0304-3975(98)00264-3.

24 J. Ian Munro. On the competitiveness of linear search. In *Proc. 8th Annual European Symposium on Algorithms (ESA)*, pages 338–345. Springer, 2000. doi:10.1007/3-540-45253-2_31.

25 Nick Reingold, Jeffery Westbrook, and Daniel D Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11(1):15–32, 1994. doi:10.1007/BF01294261.

26 Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985. doi:10.1145/2786.2793.

27 Boris Teia. A lower bound for randomized list update algorithms. *Information Processing Letters*, 47(1):5–9, 1993. doi:10.1016/0020-0190(93)90150-8.

A Handling insertions and deletions

In the "dynamic" variant of List Update, except from access requests, the input σ may contain insertions and deletions of items. By the variant definition (see, e.g., [3]):

- inserting z^* places it at the end of the list, and its cost is defined to be equal to the current length of the list;
- deleting z^* involves accessing z^* (at a cost equal to its position), and then removing it.

Right after the insertion, an algorithm may reorganize the list paying the usual cost of swaps. Similarly, such reorganization is possible right after accessing the item to be deleted, and before its actual removal.

In this section, we demonstrate that FPM can handle insertions and deletions while retaining its competitive ratio. We start with a few definitions and clarifications, showing how the arguments for lower-bounding OPT need to be adapted for the dynamic variant. Later, we we specify what swaps FPM performs due to insertions and deletions.

Adjusted cost. Let *adjusted cost* of an algorithm be the cost that omits the cost of serving insertions of items. (It does involve the cost of an optional list reorganization). As these costs are the same for both FPM and OPT, if we can show that an algorithm is R-competitive with respect to the adjusted cost, then it is also R-competitive with respect to the actual cost. From this point on, we use only adjusted cost in our analysis.

Lifetimes of pairs. Without loss of generality, we assume that when an item is deleted, it is never inserted back to the list (we simply rename its next occurrences). In the main part of the paper, we used $\mathcal{P}$ to denote the set of all pairs of items. For the dynamic variant, we will use $\mathcal{Q}$ to denote the set of all pairs of items that are ever together in the list. We call a pair $\{x, y\} \in \mathcal{Q}$

- *dormant* before x and y are both present in the list;
- *active* when both x and y are present in the list;
- *inactive* after either x or y is deleted from the list.

Each pair of $\mathcal{Q}$ is active within a certain period. This period may be preceded by the dormant one, and may be followed by the inactive one. (In the static case, all pairs are always active.)

Pair-based OPT. Previously, we related both FPM and OPT to optimal algorithms for sequences σ_{xy} operating on two-item lists containing x and y. In the dynamic variant, we define σ_{xy} to contain requests only from the period when the pair $\{x, y\}$ is active. The corresponding two-item list is also defined only within this period. The initial order of items on this two-item list matches the order of x and y on the actual list at the beginning of the active period. This implies that each pair starts its active period in mode α.

With these definitions in place, Lemma 1 extends in a straightforward way to the dynamic case, i.e., $\sum_{\{x,y\}\in\mathcal{Q}} \mathrm{OPT}(\sigma_{xy}) \leq \mathrm{OPT}(\sigma)$ for each input sequence σ. This also implies a counterpart of Lemma 2 for the dynamic variant: for any input sequence σ consisting of T requests, it holds that $\sum_{t=1}^{T} \sum_{\{x,y\}\in\mathcal{Q}} \Delta_t w^{xy} \leq \mathrm{OPT}(\sigma)$.

Potential function. Recall that the potential function Φ_{xy} for a pair $\{x, y\}$ depends on the corresponding work function W^{xy}, which is defined only for active pairs. Therefore, for completeness, we set the potential function for a dormant pair to be zero, and assume that the value of the potential function is frozen once a pair stops being active and becomes inactive.

Specifying the behavior of the algorithm. The behavior of *FPM* for access requests remains the same as in the static case. For insertions, *FPM* performs no list reorganization. Finally, suppose that an element z^* is deleted. Upon such request, *FPM* performs exactly the same list reorganization as if z^* was accessed (i.e., it moves z^* fully or partially towards the front of the list). Once this action is performed, due to the target cleanup operation, z^* is not a target for any other item in the list.

Admittedly, the behavior of *FPM* for deletions is wasteful as it moves the element which is about to be removed from the list. However, this definition streamlines the analysis, as we can reuse the arguments for access requests.

Competitive ratio. We first argue about potential changes due to insertions and deletions. Later, we use these arguments to show that *FPM* retains its competitive ratio also in the dynamic variant.

We split the action of deleting an item z^* into two parts: accessing the item (with induced list reorganization) and *actual removal* ot this item from the list.

▶ **Lemma 18.** *Fix a pair $\{x, y\} \in \mathcal{Q}$. Due to an insertion or an actual removal, $\Delta\Phi_{xy} = 0$.*

Proof. Let z^* be the item either inserted or actually removed; we consider two cases.

In the first case, pair $\{x, y\}$ does not contain z^*. If pair $\{x, y\}$ remains dormant or inactive, then Φ_{xy} is trivially unaffected. Otherwise, the pair must be and remain active. It suffices to note that neither the insertion of z^* nor the actual removal of z^* does affect θ_x or θ_y. Thus, the mode and flavor of $\{x, y\}$ remains unchanged, and hence $\Delta\Phi_{xy} = 0$.

In the second case, one of pair elements, say x, is equal to z^*.

- For insertions, this means that pair $\{z^*, y\}$ switches from the dormant state (of zero potential) to the active one. As noted above, $\{z^*, y\}$ starts in mode α. Since *FPM* sets $\theta_{z^*} = z^*$, it holds that $\theta_y \preceq y \prec \theta_{z^*} = z^*$, i.e., pair $\{z^*, y\}$ starts in flavor d. Hence, the state of $\{z^*, y\}$ right after the insertion is α^d which corresponds to the zero potential. Thus, $\Delta\Phi_{xy} = 0$.
- For actual removals, pair $\{z^*, y\}$ switches from the active state to the inactive one. The pair potential becomes frozen, and thus $\Delta\Phi_{xy} = 0$. ◀

▶ **Theorem 19.** *The competitive ratio of FPM in the dynamic variant is at most $R = \frac{1}{8}(23 + \sqrt{17}) \leq 3.3904$.*

Proof. Fix any sequence σ consisting of T requests. By Theorem 14, for any step t containing an access request, we have that

$$\Delta_t FPM + \sum_{\{x,y\} \in \mathcal{Q}} \Delta_t \Phi_{xy} \leq R \cdot \sum_{\{x,y\} \in \mathcal{Q}} \Delta_t w^{xy}. \tag{4}$$

We observe that the initial potentials for all pairs of $\mathcal{Q}$ (including the dormant ones) are zero. Thus, to complete the proof, it remains to show that (4) holds for insertions and deletions. The competitive ratio of *FPM* then follows by summing (4) over all steps t, using the non-negativity

of pair potentials, and finally applying the relation $\sum_{t=1}^{T} \sum_{\{x,y\}\in\mathcal{Q}} \Delta w^{xy} \leq \mathit{OPT}(\sigma)$ as in the proof of Theorem 14.

For insertions, by the definition of adjusted cost, we have that $\Delta_t \mathit{FPM} = \Delta_t \mathit{OPT} = 0$. By Lemma 18, $\Delta_t \Phi_{xy} = 0$ for any pair $\{x, y\} \in \mathcal{Q}$, and thus (4) holds.

For deletions, we can split the action into two parts: accessing the item (and the resulting list reorganization) and actual removal of the item. For the first part, (4) holds by Theorem 14. For the actual removal, $\Delta_t \mathit{FPM} = \Delta_t \mathit{OPT} = 0$. Furthermore, Lemma 18 implies that $\Delta_t \Phi_{xy} = 0$ for any pair $\{x, y\} \in \mathcal{Q}$, and thus (4) holds. ◀

B Lower Bounds on MTF and its Modifications

In this section, we show the lower bounds greater than 3 for several natural algorithms, most of which have previously been considered for this problem. All these bounds (unless explicitly stated) hold for the full cost model, and thus also for the partial one.

B.1 Specific Algorithms

Deterministic Bit. Algorithm Deterministic *Bit* (*DBit*) starts with all items unmarked. On a request to item r, if it is marked, it moves r to the front and unmarks it. Otherwise, it marks r. We show that the competitive ratio of *DBit* is at least 3.302.

To this end, consider a list of length n and denote its items as $x_0, x_1, \ldots, x_{n-1}$, counting from the front of the list. We divide the list into two parts, A and B, where part A has length cn and part B has length $(1-c)n$. The value of parameter c will be determined later. Formally $A = (x_0, x_1, x_2, \ldots, x_{cn-1})$ and $B = (x_{cn}, x_{cn+1}, \ldots, x_{n-2}, x_{n-1})$. For succinctness, we denote such order of the list as A, B. We define input

$$\sigma = (x_{cn-1}, x_{cn-2}, \ldots, x_1, x_0, \\ x_{n-1}, x_{n-1}, x_{n-2}, x_{n-2}, x_{n-3}, x_{n-3}, \ldots, x_{cn}, x_{cn})^2.$$

In this sequence, all items of A are requested first, starting from the one furthest in the list. Then every item of B is requested twice, also starting from the furthest one. It results in items of B being transported to the front, thus after the first half of σ, the order of the list of *DBit* is B, A. Note that order of items within B does not change.

Then the same sequence of requests is repeated. The difference is that now requesting any item of A results in moving that item to the front. Then items of B are again requested twice, resulting in final list of algorithm being B, A. Then,

$$\begin{aligned} \mathit{DBit}(\sigma) &= \frac{cn(cn+1)}{2} && \text{(access to } A\text{)} \\ &\quad + (1-c)n \cdot (3n-1) && \text{(access to } B \text{ and movement)} \\ &\quad + cn \cdot (2n-1) && \text{(access to } A \text{ and movement)} \\ &\quad + (1-c)n \cdot (3n-1) && \text{(access to } B \text{ and movement)} \\ &= n^2 \left(6 - 4c + \frac{c^2}{2}\right) + n\left(\frac{3c}{2} - 2\right). \end{aligned}$$

Now we define a better algorithm for this sequence, which upper-bounds the cost of *Opt*. After receiving the first batch of requests to the items of A, it moves all the items of B before

the items of A. It does not performs any further swaps. Thus,

$$\begin{aligned}
\mathit{OPT}(\sigma) \le{}& \frac{cn(cn+1)}{2} && \text{(access to } A\text{)}\\
&+ cn\cdot(1-c)n && \text{(swap } A \text{ with } B\text{)}\\
&+ 2\cdot\frac{1+(1-c)n}{2}\cdot(1-c)n && \text{(access to } B\text{)}\\
&+ \frac{(1-c)n+1+n}{2}\cdot cn && \text{(access to } A\text{)}\\
&+ 2\cdot\frac{1+(1-c)n}{2}\cdot(1-c)n && \text{(access to } B\text{)}\\
={}& n^2\left(2-2c+c^2\right)+n\left(2-c\right).
\end{aligned}$$

Note that the final order of the list in both algorithms is the same and equal to B, A. Moreover, every item is requested an even number of times, so no item is marked after σ. Thus, this request sequence can be repeated infinitely many times, with sets A, B redefined after each sequence.

With a long enough list, n becomes insignificant compared to n^2, and therefore the ratio is $\mathit{DBIT}(\sigma)/\mathit{OPT}(\sigma) \ge (6-4c+\frac{1}{2}c^2)/(2-2c+c^2)$. For $c=\frac{1}{3}(5-\sqrt{13})$, we have $\mathit{DBIT}(\sigma)/\mathit{OPT}(\sigma) \ge \frac{1}{2}(3+\sqrt{13}) \ge 3.302$.

Deterministic Bit in partial cost model. We now show that the competitive ratio of *DBIT* is at least 4 in the partial cost model. We recursively construct a sequence that achieves this ratio for every length of the list $n \ge 2$. For a list of length n, the created sequence will be denoted σ_n. For reasons that will become apparent soon, we index items from the end of the list, namely the algorithm's list at the start of the sequence is $x_{n-1}, x_{n-2}, \ldots, x_1, x_0$.

First, assume $n = 2$. Set

$$\sigma_2 = x_1, x_0, x_0, x_1, x_0, x_0.$$

Clearly, $\mathit{DBIT}(\sigma_2) = 8$. The cost of 2 can be achieved for that sequence by starting with the list order x_1, x_0 and swapping items only after the first request. Note that such an algorithm would have the same order of items as algorithm *DBIT* after serving σ_2.

Now let $n > 2$. Set

$$\sigma_n \;=\; x_{n-1}, \sigma_{n-1}, x_{n-1}, \sigma'_{n-1}.$$

where by σ'_{n-1} above we mean the sequence obtained from σ_{n-1} by permuting the items $x_{n-2}, \ldots, x_0$ according to their position in the list after executing σ_{n-1}. Specifically, if $x_{\pi(n-2)}, x_{\pi(n-3)}, \ldots, x_{\pi(0)}$ is the list after executing σ_{n-1} then σ'_{n-1} is obtained from σ_{n-1} by replacing each item x_i by $x_{\pi(i)}$. For example, for $n = 3$,

$$\sigma_3 \;=\; x_2, x_1, x_0, x_0, x_1, x_0, x_0, x_2, x_0, x_1, x_1, x_0, x_1, x_1.$$

We now express $\mathit{DBIT}(\sigma_n)$ and $\mathit{OPT}(\sigma_n)$ as functions of $\mathit{DBIT}(\sigma_{n-1})$ and $\mathit{OPT}(\sigma_{n-1})$, respectively.

At the start of σ_n, item x_{n-1} is at the front, marked because of the first request. Within σ_{n-1} every item other than x_{n-1} is requested even number of times (at least twice). For the first pair of such requests, x_{n-1} contributes 3 to the cost (one per each request and one for the swap when the requested item is moved to the front). Thus, the additional cost due to

x_{n-1} is at least $3(n-1)$. Then, item x_{n-1} is requested and moved to the front incurring cost $2(n-1)$. Similarly, x_{n-1} contributes an additional cost of $3(n-1)$ to σ'_{n-1}. Thus,

$$\textsc{Dbit}(\sigma_n) \;\geq\; 2 \cdot \textsc{Dbit}(\sigma_{n-1}) + 8 \cdot (n-1). \tag{5}$$

The optimal algorithm can serve the first request to x_{n-1} at cost 0 and then move it immediately to the end, paying $2(n-1)$ for the second request to x_{n-1}. In the final list of $\textsc{Opt}$, x_{n-1} is at the end, just like in the list of $\textsc{Dbit}$. This implies that serving σ_{n-1} and σ'_{n-1} will cost at most $2 \cdot \textsc{Opt}(\sigma_{n-1})$, i.e.,

$$\textsc{Opt}(\sigma_n) \;\leq\; 2 \cdot \textsc{Opt}(\sigma_{n-1}) + 2 \cdot (n-1). \tag{6}$$

Recurrence relations (5) and (6) immediately imply that $\textsc{Dbit}(\sigma_n)/\textsc{Opt}(\sigma_n) \geq 4$.

Half-Move. Finally, we consider the algorithm Half-Move, which moves the requested item to the middle point between its position and the front of the list. We show that its competitive ratio is at least 6.

Consider a list of even length n. Denote the items as $x_0, x_1, \ldots, x_{n-1}$, according to their positions in the initial list of the algorithm. The request sequence is defined as

$$\sigma = \left(x_{n-1}, x_{n-2}, x_{n-3}, \ldots, x_{n/2}\right)^k$$

for some $k \geq 1$. At the time of the request, the requested item is at the end of the list and it is moved to the middle of the list. Since in the list of even length there are two middle positions, we will assume that the item is always moved to the one further from the front of the list. (If it is moved to the other one, the proof only needs to be slightly modified.) The cost of each request for the algorithm is $n + \frac{1}{2}n - 1 = \frac{3}{2}n - 1$. Moreover, after $\frac{1}{2}n$ requests, the list returns to its original order. Therefore, the total cost of the sequence is $k \cdot \frac{1}{2}n \cdot (\frac{3}{2}n - 1)$.

It is possible to change the order of the list to $x_{n/2}, x_{n/2+1}, \ldots, x_{n-1}, x_0, x_1, x_2, \ldots, x_{n/2-1}$ before any requests arrive, at cost less than n^2. Thus, the total cost of the request sequence is $k \cdot \binom{n/2}{2}$. Thus, $\textsc{Opt}(\sigma) \leq n^2 + \frac{1}{8}kn^2$. The ratio approaches 6 for sufficiently large n, as $k \to \infty$.

B.2 Stay-or-MTF Class

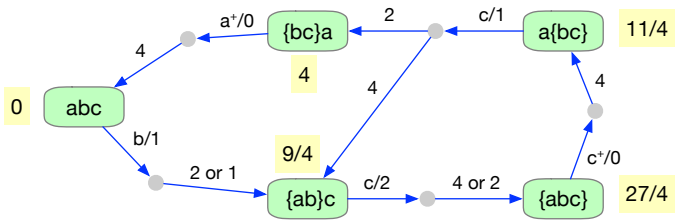
We now consider a class of online algorithms that never make partial moves. That is, when an item x is requested, an algorithm either does not change the position of x or moves x to the front. We refer to such algorithms as *Stay-or-MTF* algorithms.

We show that if A is a Stay-or-MTF algorithm, then the competitive ratio of A is at least 3.25, even for a list of length 3.

The adversary strategy is illustrated in the figure below. The list has three items named a, b and c. The vertices represent states of the game. In each state, we assume that the A's list is abc. When A executes a move, the items are appropriately renamed. Each state is specified by the current *offset function*, equal to the current work function minus the minimum of this work function. At each move, the increase of the minimum of the work function is charged towards the $\textsc{Opt}$'s cost, and the offset function is updated. Each offset function that appears in this game is specified by the set of permutations of a, b, c where its value is 0. (Its value at each other list is equal to the minimum swap distance to some offset function with value 0.) To specify this set, we indicate, using braces, which items are

allowed to be swapped. For example, $a\{bc\}$ represents two permutations, abc and acb, and $\{abc\}$ represents all permutations.

The adversary strategy is represented by the transitions between the states. For each state the adversary specifies the generated request when this state is reached. Notation r/ξ on an arrow means that the request is x and the adversary cost is ξ. Then, the algorithm decides to make a move. The decision points of the algorithm are indicated by small circles. The type of move is uniquely specified by the cost, specified on the edges from these circles. For example, if the request is on c, the algorithm either stays, which costs 2, or moves to front, which costs 4.



The numbers next to each state are the values of potential Ψ. By routine verification of the transition graph, we can check that for each move we have

$$\Delta A + \Delta\Psi \;\geq\; \tfrac{13}{4} \cdot \Delta OPT,$$

where ΔA, $\Delta\Psi$ and ΔOPT represent the cost of A, the potential change, and the optimum cost for this move. This shows that the competitive ratio of A is at least $\frac{13}{4} = 3.25$.

Our construction is tight for $n = 3$, i.e., it can be shown that there exists a Stay-or-MTF algorithm with the competitive ratio of 3.25 for $n = 3$. For $n = 4$, we can improve the lower bound to 3.3, using an approach that is essentially the same, but requires many more states.

C A 3-Competitive Algorithm for Lists of Length 3

The algorithm FPM is 3-competitive for lists of length 3. This is true even against our lower bound $\sum_{\{x,y\}\in\mathcal{P}} w^{xy}$ on the optimum cost.

To prove that, we consider possible states of FPM. There are 15 of them. We assign a potential Ψ to every state such that for any state and for any request, it holds that $\Delta FPM + \Delta\Psi \leq 3 \cdot \sum_{\{x,y\}\in\mathcal{P}} \Delta w^{xy}$. Note that Ψ is introduced only to simplify the proof: it is not related to the potential used internally by the algorithm FPM.

The states are shown in Table 4. In each state we assume that the list is abc; the items are renamed after a transition if needed. Arrows represent modes: an arrow pointing from an item x towards an item y indicates that the pair $\{x, y\}$ is in mode α (if x is before y in the list) or γ (otherwise). If there is no arrow between x and y, the pair is in mode β. The column "move" specifies what FPM does after the request, while the column "dest." denotes the index of the destination state after movement. State changes correspond to the moves of FPM. Cost of the pair-based OPT is denoted $\Delta OPT_P = \sum_{\{x,y\}\in\mathcal{P}} \Delta w^{xy}$.

There exists an easier 3-competitive algorithm for 3 items. This algorithm uses 6 states to keep track of the past history. The algorithm is illustrated below as a graph whose vertices represent the current state of the algorithm and edges represent its transitions. Again, items are renamed after each transition so that the list is always abc. The modes of each pair are represented the same way as in Table 4. Each edge is labeled by the requested item, the algorithm's cost, and the optimum cost ($\frac{1}{2}$ for each mode change). The algorithm's move is

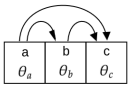
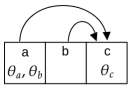
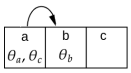
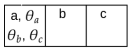
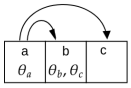
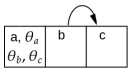
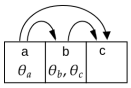
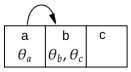
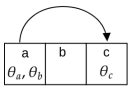
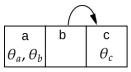
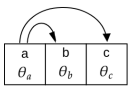
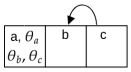
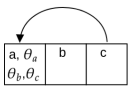
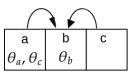
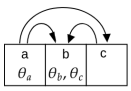
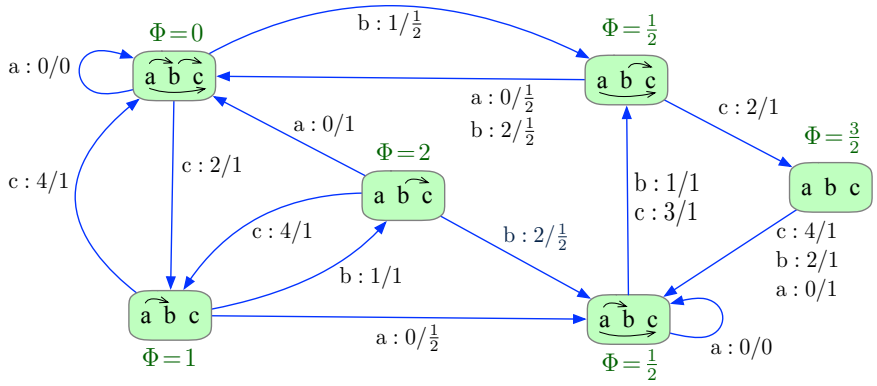
state	idx	Ψ	req.	move	dest.	ΔFPM	$\Delta\Psi$	ΔOPT_P
	0	0	a	FRONT	0	0	0	0
			b	Θ_b	1	1	$\frac{1}{2}$	$\frac{1}{2}$
			c	Θ_c	2	2	1	1
	1	$\frac{1}{2}$	a	FRONT	0	0	$-\frac{1}{2}$	$\frac{1}{2}$
			b	FRONT	0	2	$-\frac{1}{2}$	$\frac{1}{2}$
			c	Θ_c	3	2	1	1
	2	1	a	FRONT	4	0	$-\frac{1}{2}$	$\frac{1}{2}$
			b	Θ_b	5	1	1	1
			c	FRONT	0	4	-1	1
	3	$\frac{3}{2}$	a	FRONT	4	0	-1	1
			b	FRONT	4	2	-1	1
			c	FRONT	4	4	-1	1
	4	$\frac{1}{2}$	a	FRONT	4	0	0	0
			b	Θ_b	1	1	0	1
			c	Θ_c	1	3	0	1
	5	2	a	FRONT	6	0	0	1
			b	FRONT	4	2	$-\frac{3}{2}$	$\frac{1}{2}$
			c	FRONT	7	4	-1	1
	6	2	a	FRONT	6	0	0	0
			b	Θ_b	1	1	$-\frac{3}{2}$	$\frac{1}{2}$
			c	Θ_c	8	3	0	1
	7	1	a	FRONT	4	0	$-\frac{1}{2}$	$\frac{1}{2}$
			b	Θ_b	9	1	1	1
			c	FRONT	0	4	-1	1
	8	2	a	FRONT	10	0	$-\frac{1}{2}$	$\frac{1}{2}$
			b	FRONT	0	2	-2	1
			c	Θ_c	11	2	1	1
	9	2	a	FRONT	0	0	-2	1
			b	FRONT	10	2	$-\frac{1}{2}$	$\frac{1}{2}$
			c	Θ_c	12	2	1	1
	10	$\frac{3}{2}$	a	FRONT	10	0	0	0
			b	Θ_b	1	1	-1	1
			c	Θ_c	13	2	1	1
	11	3	a	FRONT	14	0	-1	1
			b	FRONT	7	2	-2	1
			c	FRONT	4	4	$-\frac{5}{2}$	$\frac{1}{2}$
	12	3	a	FRONT	7	0	-2	1
			b	FRONT	14	2	-1	1
			c	FRONT	4	4	$-\frac{5}{2}$	$\frac{1}{2}$
	13	$\frac{5}{2}$	a	FRONT	14	0	$-\frac{1}{2}$	$\frac{1}{2}$
			b	Θ_b	3	1	-1	1
			c	FRONT	0	4	$-\frac{5}{2}$	$\frac{1}{2}$
	14	2	a	FRONT	14	0	0	0
			b	Θ_b	8	1	0	1
			c	Θ_c	1	3	$-\frac{3}{2}$	$\frac{1}{2}$

Table 4 *FPM* is 3-competitive for lists of length 3

implied by the cost value. For example, if the request is c and the cost is 3, it means that the algorithm paid 2 to access c and then swapped c with the preceding b.



The diagram also shows the potential values Φ for each state. By routine verification, for each transition we have that the amortized cost (that is, the cost of the algorithm plus the potential change) is at most 3 times the optimum cost.

D Lower bound for the Full-Or-Partial-Move algorithm

In this section, we show that for the choice of parameters from Lemma 11, the competitive ratio of FPM is at least 3.04 in the partial cost model.

The sequence achieving this ratio is shown in Table 5. It uses five items (on a longer list one may simply use only five initial items). Graphs in the column "work function on pairs" depict modes of all the pairs. There, an arrow from x to y means that $W^{xy}(xy)+1 = W^{xy}(yx)$ (which corresponds to mode α or γ, depending on the order of the list). Some arrows are omitted for clarity: the actual graph is the transitive closure of the graph shown.

Note that the final state of the list after the sequence is the same as the state before the sequence, therefore it can be repeated indefinitely. The first state described in the table is accessible from the starting state described in Section 3 by requesting item d and then item a.

Table 5 compares cost of FPM to the cost of *pair-based* OPT defined as $P_{OPT} = \sum_{\{x,y\}\in\mathcal{P}} \Delta w^{xy}$, which for this sequence is equal to 25. We now show the behavior of an actual offline algorithm OPT that achieves this cost. Before the start of the sequence, the algorithm changes its list to *cdeab*. This incurs an additional additive cost, which becomes negligible if the sequence is repeated sufficiently many times. During the entire sequence, OPT performs two moves: after serving the first request, it moves c to the third position in the list, and after the ninth request, it brings c back to the front. Thus, it executes 4 swaps. It is easy to check that the cost of requests for paid by OPT is 21, and hence its total cost is 25.

E Computational Results

This section contains the description of our computational study of the LUP_1 problem. Our source code for reproducing the results is available at [11].

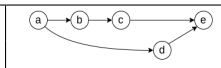
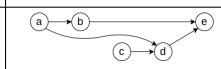
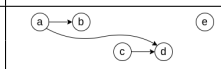
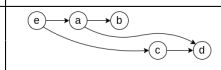
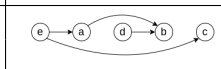
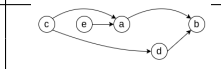
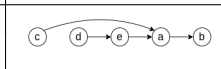
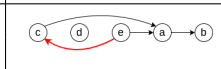
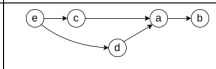
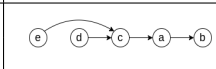
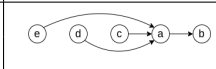
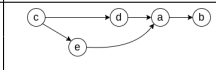
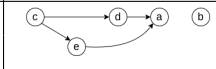
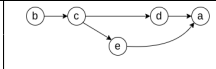
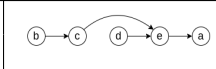
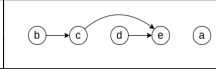
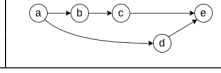
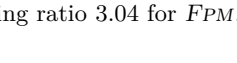
list of FPM	work function on pairs	req.	move of FPM	cost of FPM	cost of OPT_P
a θ_a | b θ_b θ_d | c θ_c | d | e θ_e					
a θ_a θ_c | b θ_b θ_d | c | d | e θ_e		c	θ_c	2	$1\frac{1}{2}$
a θ_a | b θ_b | c | d | e θ_e		e	θ_e	6	$3\frac{1}{2}$
e θ_e | a θ_a θ_c | b θ_b θ_d | c | d		e	$FRONT$	14	$5\frac{1}{2}$
e θ_e θ_d | a θ_a θ_c | d | b θ_b | c		d	θ_d	20	$7\frac{1}{2}$
c θ_c | e θ_d θ_e | a θ_a | d | b θ_b		c	$FRONT$	28	$9\frac{1}{2}$
c θ_c θ_d | d | e θ_e | a θ_a | b θ_b		d	θ_d	33	11
c θ_c | d | e θ_e | a θ_a | b θ_b		e	θ_e	35	12
e θ_e | c θ_c θ_d | d | a θ_a | b θ_b		e	$FRONT$	39	$12\frac{1}{2}$
e θ_e θ_d | d | c θ_c | a θ_a | b θ_b		d	θ_d	42	$13\frac{1}{2}$
e θ_e | d | c θ_c | a θ_a | b θ_b		c	θ_c	44	$14\frac{1}{2}$
e θ_e θ_d | d | c | a θ_a | b θ_b		c	$FRONT$	48	$15\frac{1}{2}$
c θ_c | e θ_d θ_e | d | a θ_a | b θ_b		b	θ_b	52	$17\frac{1}{2}$
c θ_b θ_c | e θ_d θ_e | d | a θ_a | b		b	$FRONT$	60	$19\frac{1}{2}$
b θ_b | c θ_c | e θ_d θ_e | d | a θ_a		d	θ_d	64	21
b θ_b θ_d | c θ_c | d | e θ_e | a θ_a		a	θ_a	68	23
b θ_a θ_b θ_d | c θ_c | d | e θ_e | a		a	$FRONT$	76	25
a θ_a | b θ_b θ_d | c θ_c | d | e θ_e					

Table 5 Sequence achieving ratio 3.04 for FPM. The red arrow indicates the pair in mode γ.

Note on the accuracy of our results. Our computational results serve as sources of insight into the LUP_1 problem, and our code produces graphs which can be checked for correctness, although the size of the graph may be an obstacle.

We therefore list our results as *computational results*, not as theorems, and advise the reader to be similarly cautious when interpreting them or citing them in future work.

E.1 Upper bound of 3 for list length up to 6

We start by discussing our results of a exhaustive computer search for the optimal competitive ratio for the LUP_1 model for small list lengths.

▶ **Computational Result 20.** *There exists a 3-competitive algorithm for LUP_1 for list size $n \leq 6$.*

Our computational method starts with a fixed competitive ratio 3 and creates a bipartite graph. In one part (we call it the *Opt* part), we store all pairs (σ, W), where σ is the current list of the algorithm and W is a work function for this state. (Recall that the work function W is the optimal cost of serving the input σ and ending in a given list state.) Each vertex in this part has n edges, corresponding to the new request r. Each edge leads to a vertex in the *Alg* part of the graph, which stores a triple (σ, W', r), where W' is the new work function after including the new request r into the sequence. In turn, each *Alg* vertex has $n!$ edges, each leading to a different permutation σ', which we can interpret as the new list of the algorithm after serving the request r.

Edges of the graph have an associated cost with them, where the cost of edges leaving *Alg* correspond to the cost of the algorithm serving the request r and the cost of edges leaving *Opt* correspond to the increase of the minimum of the work function.

Each vertex in this graph stores a number, which we call the *potential* of the *Alg* or *Opt* vertex. We initialize the potentials to all-zero values and then run an iterative procedure, inspired by [17], which iteratively computes a new minimum potential in each *Opt* vertex and a new maximum potential in each *Alg* vertex.

If this procedure terminates, we have a valid potential value for each vertex in the graph, and by restricting our algorithm to pick any tight edge leaving an *Alg* vertex, we obtain an algorithm that is 3-competitive. The data generated after this termination are similar in nature to Table 4.

Our computational results correspond to our procedure terminating for list sizes $4 \leq n \leq 6$.

For list size $n = 6$, it is no longer feasible to create a graph of all reachable work functions, as our computations indicate that even listing all of these would require more memory than our current computational resources allow. We therefore switch to the model of pair-based *Opt* of Section 2. As explained in that section, a c-competitive algorithm for pair-based *Opt* is also c-competitive against regular *Opt*. However, the space of reachable pair-based work functions is much smaller, so the setting of $n = 6$ can be explored computationally as well.

Note that we can also switch the results of $n = 4$ and $n = 5$ into the pair based setting, which improves the running time. The computational results against the pair-based OPT in those cases are consistent with our results against general *Opt*.

E.2 Lower bound for the work function algorithm class

We can translate the algorithms for LUP_S from [9] to the LUP_1 model. For input σ to LUP_1 and a list π, the work function $W^\sigma(\pi)$ defines the optimal cost of serving the input σ and ending with the list π. Then, a work function algorithm, given its current state of the list μ

and a new request r, it first serves the request and then reorders the list to end in a state π being a minimizer of $W^{\sigma,r}(\pi) + d(\pi, \mu)$, where $d(\pi, \mu)$ is the number of pairwise swaps needed to switch from μ to π. The minimizer state is not necessarily unique, hence we talk about the *work function algorithm class*.

Recall that some algorithms from the work function algorithm class are 2-competitive for $\mathsf{LUP}_{\mathrm{S}}$ [9], which means they are optimal for that setting. In contrast, we show the following.

▶ **Computational Result 21.** *The competitive ratio of all algorithms in the work function algorithm class for LUP_1 is at least* 3.1 *(for the partial cost model).*

To show this claim, we construct a lower bound instance graph for list length 5 where all choices of the work function algorithm class are being considered. This graph is equivalent to the graph construction of Subsection E.1, with the out-edges of the $\textsc{Alg}$ vertices restricted only to all valid choices of the work function algorithm class.

However, since the result of an iterative algorithm is that the potential does not stabilize, we constructed a tailored approach for producing a lower bound graph from the unstable potentials. A result of this code is then a standard lower bound using an adversary/algorithm graph, with the adversary presenting a single item in each of its vertices and the competitive ratio of any cycle in the graph being at least 3.1.

We remark that this approach yields a 3-competitive work function algorithm for LUP_1 for list lengths 3 and 4.