Competitive Algorithms for Multilevel Caching and Relaxed List Update (Extended Abstract)

Marek Chrobak* John Noga†

Abstract

We study the Relaxed List Update Problem (RLUP), in which access requests are made to items stored in a list. The cost to access the jth item x_j is c_j, where $c_i \leq c_{i+1}$ for all i. After the access, x_j can be repeatedly swapped, at no cost, with any item that precedes it in the list. This problem was introduced by Aggarwal et al [1] as a model for the management of hierarchical memory that consists of a number of caches of increasing size and access time. They also proved that a version of LRU is C-competitive, for some C, for a restricted class of cost functions.

(1) We give an efficient offline algorithm that computes the optimal strategy for RLUP. We also show an elegant characterization of work functions for RLUP. (2) We prove that Move-to-Front (MTF) is optimally competitive for RLUP with any cost function. An interesting feature of the proof is that it does not involve any estimates on the competitive ratio. (3) We give a lower bound on the competitive ratio of online algorithms for RLUP.

1 Introduction:

Caching. In a two-level memory system a processor has a cache that stores data items that are likely to be accessed in the future. Given an access request to a memory address r, if r is in the cache, it can be accessed at little cost. If r is not in the cache – an event referred to as a "miss" or "fault" – the processor fetches r from the main memory, and may replace some other item x in the cache by r. Our goal is to design a replacement strategy that will minimize the number of misses.

We distinguish two versions of the caching problem. In the *required-fetch* model, when the request r is a miss, r must be put in the cache, while in the *optional-fetch* model the request can be served without fetching r.

In the offline version of the problem, we are given the initial cache configuration X and a request sequence ϱ, and we wish to compute the optimal replacement sequence ϱ when starting from X. Belady [2] proposed a simple and efficient algorithm that computes an optimal service. (The Belady algorithm was formulated for the required-fetch model, but it can be easily adapted to the optional-fetch model.) In practice, however, the replacement decisions need to be made *online*, that is, without the knowledge of future requests. One of the most popular and most studied online algorithms is LRU, that at each miss evicts from the cache the least recently accessed item. Another well-known strategy is FIFO, that evicts the item that was first fetched into the cache. It is easy to show that no online replacement strategy can minimize the number of misses on each request sequence.

Competitive algorithms. In this abstract we adopt the competitive ratio as the performance measure of online algorithms. Given a constant $C \geq 1$, we say that an online algorithm $\mathcal{A}$ is *C-competitive* if, for each request sequence ϱ,

$$(1.1) \qquad cost_{\mathcal{A}}(X,\varrho) \quad \leq \quad C \cdot opt(X,\varrho) + b_X,$$

where $cost_{\mathcal{A}}(X,\varrho)$ is the cost of $\mathcal{A}$ on ϱ when starting with X in the cache, $opt(X,\varrho)$ is the corresponding optimal cost, and b_X is a constant independent of ϱ. In our applications, constant b_X will always be zero. For simplicity, unless otherwise stated, we assume that the computation starts from a fixed initial configuration X_{init}, and then we simplify notation to $cost_{\mathcal{A}}(\varrho)$ and $opt(\varrho)$. The *competitive ratio* of $\mathcal{A}$, denoted by $C_{\mathcal{A}}$, is the smallest C for which $\mathcal{A}$ is C-competitive. Note that the definition (1.1) is not specific to the caching problem; it can be used to measure the performance of online algorithms for a variety of optimization problems.

Caching is probably the most studied problem in the literature on competitive online algorithms. Sleator and Tarjan [3] proved that, in the required-fetch model and with a cache of size s, the competitive ratio of LRU and FIFO is s. (In the optional-fetch model both competitive ratios are $s+1$.) A randomized version of the problem was studied in [4, 5, 6]. Some research has been done on relaxing the worst-case approach to competitive analysis and on refining the competitive measure in order to take into account temporal and

*Department of Computer Science, University of California, Riverside, CA 92521, USA. Email: marek@cs.ucr.edu. Research supported by the NSF grant CCR-9503498.

†Department of Mathematics, University of California, Riverside, CA 92521, USA. Email: jnoga@cs.ucr.edu. Research supported by the NSF grant CCR-9503498.

spatial locality of memory references, see [7, 8, 9, 10, 11].

Relaxed List Update. In the List Update Problem (LUP), we wish to maintain a linear list in response to access requests. If the current list is $x_1x_2\ldots x_k$, the cost of accessing item x_p is p. After being accessed, item x_p can be moved at no cost to any location earlier in the list. Sleator and Tarjan [3] proved that Move-To-Front (MTF) is 2-competitive for LUP, and Irani [12] showed that the exact competitive ratio of MTF is $2k/(k+1)$. The status of the randomized version of the problem remains open. The best known randomized algorithm, by Albers et al [13], has competitive ratio 1.6, while the best lower bound, by Teia [14], is 1.5.

We concentrate only on the access operations on a linear list. Thus, at each step, the list is simply a permutation of the item set. The problem considered by Sleator and Tarjan [3] is actually slightly more general, since they also allow insertions and deletions. These operations, however, do not affect the competitive ratio.

Relaxed List Update Problem (RLUP) is a variant of LUP, in which the cost to access x_p is c_p, where $0 \le c_1 \le \ldots \le c_k$. The accessed item x_p can be repeatedly swapped, at no cost, with arbitrary items earlier in the list. We call this problem "relaxed", since in LUP the requested item x_p can be repeatedly swapped only with the immediately preceding item. Clearly, this additional flexibility of RLUP can reduce the cost of an optimal service sequence.

Multilevel caching. Some modern computer architectures achieve improved memory performance by implementing a *hierarchy* of caches. Such a multilevel cache system consists of m caches of increasing size and access time, where the level-1 cache is smallest and fastest, and the main memory is viewed as the largest and slowest cache at level m. (See [15, 16, 17].)

Aggarwal et al [1] introduced RLUP as a model for the management of hierarchical memory [1]. They also proved that a version of LRU is C-competitive, for some C, if there is an α such that $c_{2i} \le \alpha c_i$ for all i. In addition to online algorithms, they investigated offline algorithms for searching, FFT and matrix multiplication in the hierarchical memory model. See also [18, 19] for other related work on hierarchical memory.

If the multilevel cache system consists of m fully associative caches, where cache i has size s_i and access time f_i, it can be modeled as RLUP in the following way: we let $k = \sum_{\ell=1}^{m} s_\ell$, and the cost function is $c_i = f_j$, where j is smallest for which $\sum_{\ell=1}^{j} s_\ell \ge i$. Sometimes such multilevel systems satisfy the *inclusion property*: if an item is in the cache at level j then it also belongs to all caches at levels greater than j. The implementation of the inclusion property facilitates maintaining memory consistency. For such a system, we let $k = s_m$, and $c_i = f_j$, where j is smallest for which $s_i \ge j$.

The reader needs to keep in mind that RLUP is only an abstraction of the actual caching problem tailored specifically towards the analysis of replacement strategies. The systems used in practice are usually not fully associative at all levels. Instead, combinations of fully associative, set associative and direct-map caches are used. There are other complicating factors that impact the overall cache performance that are not taken into account in the RLUP model: the difference between the read and write requests, bus congestion, etc.

Our results. We provide a number of results on offline and online algorithms for RLUP. First, we give a simple and efficient optimal offline strategy for RLUP. Our strategy can be viewed as a generalization of Belady's algorithm. In addition we also provide an elegant characterization of work functions for RLUP.

Our next result is that MTF is optimally competitive for RLUP for an arbitrary choice of the cost function c_i. The interesting feature of our proof is that it does not involve any estimates on the competitive ratio of MTF. In fact, there appears to be no simple formula describing the competitive ratio of MTF for arbitrary costs. It turns out that for the special case when the cost function corresponds to the caching problem, MTF is equivalent to LRU. For cache systems with more levels, MTF can be thought of as an algorithm that applies LRU independently to each level. In addition, we provide a lower bound on the competitive constant of MTF (and thus for any online algorithm) for any cost function, and we conjecture that MTF achieves this competitive ratio. For the special case when the cost function is $c_i = i$, we show that the competitive ratio of MTF for RLUP is $2k/(k+1)$, the same as for LUP.

Our results reveal an interesting connection between the paging and list update problems, showing that MTF and LRU are, in essence, two special instances of the same algorithm. We believe that exploring this connection further can yield better understanding of the work functions for LUP.

2 Optimal Offline Algorithm for RLUP

We assume that the item set is $\{1, 2, \ldots, k\}$. We use letters $r, x, y, \ldots$, possibly with subscripts, to denote items, and boldface letters, $\mathbf{x}, \mathbf{y}, \ldots$ to denote lists of items, say $\mathbf{x} = x_1x_2\ldots x_k$, etc. Thus, a list is simply a permutation of $\{1, \ldots, k\}$. By *swap*(x_i, x_j) we will mean

[1] The model in [1] is named differently, and the treatment of this topic is rather informal, but careful reading of the proofs reveals that the model the authors had in mind is in fact equivalent to RLUP.

the operation which interchanges the positions of x_i and x_j in $\mathbf{x}$. Any $swap(x_i, x_j)$ which brings the current request to an earlier position in the list will be called free (or unpaid) and have cost 0. Any other $swap(x_i, x_j)$ will be called spontaneous (or paid) and have cost $|c_j - c_i|$. Without loss of generality, we assume that the initial list is $\mathbf{x}_{init} = 12 \ldots k$. We use Greek letters ϱ, σ, τ to denote sequences of requests.

A *service* for $\mathbf{x}$ and $\varrho = r_1 r_2 \ldots r_n$ is a sequence $\mathcal{A} = \mathcal{A}_1 \mathcal{A}_2 \ldots \mathcal{A}_n$, where $\mathcal{A}_i$ is the sequence of swaps executed when r_i is issued. In the proofs we will use the concept of the *length of* $\mathcal{A}$, which is defined as the number of all swaps in $\mathcal{A}$ plus n.

Given a request sequence ϱ and an item x, $\mathit{first}_\varrho(x)$ is the index of the first occurrence of x in ϱ. (If x does not appear in ϱ, we assume $\mathit{first}_\varrho(x) = \infty$.)

Algorithm OPT. Suppose that the current list is $\mathbf{x} = x_1 x_2 \ldots x_k$, and the remaining request sequence is $\varrho = x_p \sigma$, that is, x_p is the first request in ϱ. Pick $j \in \{1, \ldots, p\}$ that maximizes $\mathit{first}_\sigma(x_j)$. If $j = p$, do not make any swaps on x_p, and apply OPT to $\mathbf{x}$ and σ. If $j < p$, execute $swap(x_j, x_p)$ and apply OPT to the new list $\mathbf{x}'$ and ϱ.

In other words, when serving x_p, OPT repeatedly swaps x_p with items x_j that precede x_p and maximize $\mathit{first}_\sigma(x_j)$, as long as $\mathit{first}_\sigma(x_j) > \mathit{first}_\sigma(x_p)$.

In this section we will prove that OPT computes an optimal service. We first formulate the following *continuation principle*: Let $\mathcal{A}$ be an optimal service for $\mathbf{x}$ and $\varrho = x_p \sigma$.

(i) Suppose that $\mathcal{A}$ does not execute any swaps on x_p. Then $\mathcal{A}$ (ignoring the empty service sequence for x_p) is also optimal for $\mathbf{x}$ and σ, and $opt(\mathbf{x}, \varrho) = opt(\mathbf{x}, \sigma) + c_p$.

(ii) Suppose that $\mathcal{A}$ executes at least one swap on x_p, starting with $swap(x_i, x_j)$, for $i < j$, and let $\mathcal{A}'$ be the remainder of this service (excluding only $swap(x_i, x_j)$). Let also $\mathbf{x}'$ be the result of applying $swap(x_i, x_j)$ to $\mathbf{x}$. Then $\mathcal{A}'$ is an optimal service for $\mathbf{x}'$ and ϱ, and

  (a) if $j = p$ (free swap) then $opt(\mathbf{x}, \varrho) = opt(\mathbf{x}', \varrho) + c_p - c_i$.

  (b) if $i = p$ (spontaneous swap) then $opt(\mathbf{x}, \varrho) = opt(\mathbf{x}', \varrho)$.

  (c) if $p \neq i$ and $p \neq j$ (spontaneous swap) then $opt(\mathbf{x}, \varrho) = opt(\mathbf{x}', \varrho) + c_j - c_i$.

Note that the formulas in (a) and (c) are the same, but for a different reason: the $(c_j - c_i)$ term in (a) accounts for the difference in access cost, while in (c) it is the cost of the paid swap. An easy proof of the continuation principle (which is, in essence, an instance of the principle of optimality in dynamic programming) is left to the reader.

LEMMA 2.1. *For all* $\mathbf{x}$ *and* ϱ *there is an optimal service that contains only free swaps.*

Proof. It is sufficient to show the following claim: Let $\varrho = x_p \sigma$ and let $\mathcal{A}$ be an optimal service for $\mathbf{x}$ and ϱ that first executes a spontaneous swap on x_p, say $swap(x_i, x_j)$ for $i < j \neq p$, and only free swaps afterwards. Then there is another optimal service $\mathcal{A}'$ of the same length (as defined earlier in this section) for $\mathbf{x}$ and ϱ that contains only free swaps.

The proof of this claim is by induction on the length of $\mathcal{A}$. If the length of $\mathcal{A}$ is 2 (σ is empty, and $\mathcal{A}$ executes only one swap on x_p), the claim is trivial, since we can simply replace $\mathcal{A}$ by an empty service. So suppose that $\mathcal{A}$ has length at least 3.

If $\mathcal{A}$ executes only one swap on x_p then consider a service $\mathcal{B}$ for $\mathbf{x}$ and σ that is the same as $\mathcal{A}$ except that $swap(x_i, x_j)$ is executed on the next request after x_p. Since $\mathcal{B}$ is shorter $\mathcal{A}$, by induction, there is an optimal service $\mathcal{B}'$ for $\mathbf{x}$ and σ that only has free swaps. By adding an empty swap sequence for x_p to $\mathcal{B}'$ we obtain an optimal service $\mathcal{A}'$ for $\mathbf{x}$ and ϱ without paid swaps.

So we can assume from now on that $\mathcal{A}$ executes at least two swaps on x_p. By the assumption, the second swap is free, say $swap(x_a, x_p)$, where x_a precedes x_p after executing $swap(x_i, x_j)$.

We first claim that the case $j = a$ cannot occur in $\mathcal{A}$. Suppose, towards contradiction, that $j = a$. If $i = p$, then remove the first two swaps. If $i < p < j$ then replace the first two swaps by $swap(x_i, x_p)$ and $swap(x_i, x_j)$. If $p > j$ then replace the first two swaps by $swap(x_j, x_p)$ and $swap(x_i, x_p)$. In each case we reduce the cost, contradicting the optimality of $\mathcal{A}$. Thus we can assume that $j \notin \{a, p\}$.

Next, suppose that $i \notin \{a, p\}$, that is, the first two swaps are independent. Then we can exchange the order of the first two swaps and apply the inductive assumption, as follows: Let $\mathbf{x}'$ be the list obtained from $\mathbf{x}$ by applying $swap(x_a, x_p)$. Then the service $\mathcal{B}$ obtained from $\mathcal{A}$ by removing the second swap, $swap(x_a, x_p)$, is optimal for $\mathbf{x}'$ and ϱ. Since $\mathcal{B}$ is shorter than $\mathcal{A}$, by induction, there is an optimal service $\mathcal{B}'$ for $\mathbf{x}'$ and ϱ without paid swaps. We obtain $\mathcal{A}'$ by adding $swap(x_a, x_p)$ at the beginning of $\mathcal{B}'$.

When $i \in \{a, p\}$ and $j \notin \{a, b\}$ we will construct a service $\mathcal{D}$ which differs from $\mathcal{A}$ only in the first two swaps, has the same cost as $\mathcal{A}$, and begins with a free swap. The lemma will follow by induction, similarly to the previous case.

If $i = a < j < p$ then $\mathcal{D}$'s first two swaps are

$\mathit{swap}(x_a, x_p)$ and $\mathit{swap}(x_p, x_j)$. If $a < i = p < j$ then $\mathcal{D}$'s first two swaps are $\mathit{swap}(x_a, x_p)$ and $\mathit{swap}(x_j, x_a)$.

The final case is when $i = p < a < j$. The first two swaps are $\mathit{swap}(x_a, x_j)$ and $\mathit{swap}(x_p, x_j)$. Unlike the last two cases, however, after applying the inductive assumption, we still have a service of the same length which begins with a *paid* swap. If this new service satisfies one of the previous cases, we are done. If this new service belongs to this same case, then the swapped items are now closer to each other. Thus after at most $j - i$ applications of this case the resulting service must belong to one of the other cases.

LEMMA 2.2. *Suppose we have two lists* $\mathbf{x} = x_1 x_2 \ldots x_k$, $\mathbf{y} = y_1 y_2 \ldots y_k$, *and* $a < b$, *such that* $x_i = y_i$ *for* $i \neq a, b$ *and* $x_a = y_b$, $x_b = y_a$. *In other words,* $\mathbf{x}$ *and* $\mathbf{y}$ *are identical except that the items at locations* a, b *are interchanged. If* $\mathit{first}_\varrho(x_a) \leq \mathit{first}_\varrho(x_b)$ *then* $\mathit{opt}(\mathbf{x}, \varrho) \leq \mathit{opt}(\mathbf{y}, \varrho)$.

Proof. It is sufficient to show the following claim: if ϱ does not contain x_a or x_b, and if $\mathcal{B}$ is an optimal service for $\mathbf{y}$ and ϱx_a, then there is a service $\mathcal{A}$ for $\mathbf{x}$ and ϱx_a such that $\mathit{cost}_\mathcal{A}(\mathbf{x}, \varrho x_a) \leq \mathit{cost}_\mathcal{B}(\mathbf{y}, \varrho x_a)$, and after ϱx_a the lists of $\mathcal{A}$ and $\mathcal{B}$ become equal.

We prove this claim by induction on the minimum length of an optimal service $\mathcal{B}$ for ϱx_a. Without loss of generality, $\mathcal{B}$ does not contain any spontaneous swaps. Let the first request in ϱ be x_p, where $p \neq b$. The claim is trivially true when the length of $\mathcal{B}$ is 1 ($p = a$ and $\mathcal{B}$ is empty). So we can assume that $\mathcal{B}$ has length at least 2.

Assume first that $p = a$. $\mathcal{B}$ makes at least one swap, say $\mathit{swap}(x_i, x_a)$. If $i = b$, $\mathcal{A}$ does nothing (the lists become equal), and from then on $\mathcal{A}$ makes the same swaps as $\mathcal{B}$. If $i < a$, $\mathcal{A}$ executes $\mathit{swap}(x_i, x_a)$ and $\mathit{swap}(x_i, x_b)$, a spontaneous swap after which the two lists now become equal, and then makes the same swaps as $\mathcal{B}$. The final subcase is when $a < i < b$. Then $\mathcal{A}$ does $\mathit{swap}(x_i, x_b)$, a spontaneous swap, and the claim now follows by induction.

Assume now that $p \neq a$. If the first swap in $\mathcal{B}$ does not reorder x_a and x_b, $\mathcal{A}$ mimics $\mathcal{B}$ with x_b and x_a interchanged. (Note that $\mathcal{A}$ may need to execute a spontaneous swap.) The cost of $\mathcal{A}$ does not exceed the cost of $\mathcal{B}$, and the lemma follows by induction. If $\mathcal{B}$ reorders x_a and x_b, then $p > b$ and $\mathcal{B}$ makes $\mathit{swap}(x_b, x_p)$. In this case $\mathcal{A}$ executes $\mathit{swap}(x_b, x_p)$ and $\mathit{swap}(x_a, x_p)$, after which the lists become equal.

THEOREM 2.1. *Algorithm* OPT *computes an optimal service sequence.*

Proof. We use notation from the description of OPT. It is sufficient to prove the following claim: There is an optimal service for $\mathbf{x}$ and $\varrho = x_p \sigma$ that begins with $\mathit{swap}(x_j, x_p)$, if $j < p$, and makes no swaps if $j = p$.

We prove this claim by induction on the minimum length of an optimal service $\mathcal{B}$ for ϱ starting from $\mathbf{x}$. If the length of $\mathcal{B}$ is 1 ($\varrho = x_p$ and $\mathcal{B}$ does not make any swaps), the claim is trivial. So we can assume that $\mathcal{B}$ has length at least two.

Suppose first that $\mathcal{B}$ does not make any swaps on x_p. If $j = p$, we are done. If $j < p$, let $\mathcal{A}$ be a service that executes only $\mathit{swap}(x_j, x_p)$ on x_p and then proceeds optimally on σ. Since $\mathit{first}_\sigma(x_j) \geq \mathit{first}_\sigma(x_p)$, the claim follows from Lemma 2.2.

Suppose now that $\mathit{swap}(x_i, x_p)$ is $\mathcal{B}$'s first swap on x_p and let $\mathbf{y}$ be the new list. By induction, after this first swap, $\mathcal{B}$ makes the same moves as OPT would have made on ϱ when started on $\mathbf{y}$. If $i = j$, we are done. Otherwise we have two subcases.

If $i < j$, then let $\mathcal{A}$ be a service that executes first $\mathit{swap}(x_j, x_p)$ and $\mathit{swap}(x_i, x_p)$ (instead of $\mathit{swap}(x_i, x_p)$) and then proceeds optimally. The access cost for x_p is the same for $\mathcal{A}$ and $\mathcal{B}$. If $\mathbf{y}'$ is the new list of $\mathcal{A}$ then, since $\mathit{first}_\sigma(x_i) \leq \mathit{first}_\sigma(x_j)$, Lemma 2.2 implies that $\mathit{opt}(\mathbf{y}', \varrho) \leq \mathit{opt}(\mathbf{y}, \varrho)$. Thus $\mathcal{A}$ is also optimal for $\mathbf{x}$ and ϱ.

If $i > j$, by induction, the next swap of $\mathcal{B}$ is $\mathit{swap}(x_p, x_j)$. Let $\mathbf{z}$ be the list of $\mathcal{B}$ after these two swaps. $\mathcal{A}$ makes $\mathit{swap}(x_p, x_j)$, obtaining a list $\mathbf{z}'$, and then proceeds optimally. The claim follows again from Lemma 2.2.

OPT can be implemented efficiently by precomputing the list of requests to each item, and by maintaining a sequence $H = h_1 h_2 \ldots h_\ell$, where x_{h_a} maximizes $\mathit{first}(x_i)$ for $1 \leq i < h_{a+1}$ (we assume $h_{\ell+1} = k + 1$). By storing H in a balanced binary search tree one can achieve time complexity $O(m + n \log k)$, where n is the number of requests, and m is the number of swaps in the service computed by OPT. We leave the details to the reader.

3 Characterization of RLUP Work Functions

We use capitals $X, Y, \ldots$ to denote sets of items. We will simplify notation by identifying an item x with a singleton set $\{x\}$, and writing $X + x$ instead of $X \cup \{x\}$ and $X - x$ instead of $X - \{x\}$. If $\mathbf{x}$ is a list, $\mathbf{x} = x_1 x_2 \ldots x_k$, then we use $\mathbf{x}^j$ to denote the *set* consisting of the first j items in $\mathbf{x}$, $\mathbf{x}^j = \{x_1, \ldots, x_j\}$. Greek letters ω, μ will denote functions that assign real numbers to lists of items. Similarly, ω^j, μ^j are functions defined on j-element sets. $\mathbf{S}^j, \mathbf{R}^j, \ldots$, are length-$j$ vectors whose elements are sets of items, say $\mathbf{S}^j = (S^j_1, \ldots, S^j_j)$. $\mathbf{S}, \mathbf{R}, \ldots$ are $k \times k$ lower-triangular matrices whose elements are sets of items, $\mathbf{S} = \left[S^j_i\right]_{1 \leq i \leq j \leq k}$ (j is the row index, and i is the column index), etc.

Nested vectors. Fix $j \in \{1, 2, \dots, k\}$. Let $\mathbf{S}^j = (S_1^j, \dots, S_j^j)$ be a vector whose elements are sets of items. $\mathbf{S}^j$ is called *nested* if $S_i^j \subsetneq S_{i+1}^j$ for $i = 1, \dots, j-1$. We say that a j-element set X is *represented* by $\mathbf{S}^j$ iff $\left|X \cap S_i^j\right| \geq i$, for all $1 \leq i \leq j$. To simplify notation, we will treat $\mathbf{S}^j$ as the name of the family of j-element sets represented by $\mathbf{S}^j$, and thus we can write $X \in \mathbf{S}^j$ when X is represented by $\mathbf{S}^j$.

Let $\mathbf{S}^j = \left(S_1^j, S_2^j, \dots, S_j^j\right)$. The *update operator* transforms $\mathbf{S}^j$ into another vector, denoted $\mathbf{S}^j \wedge r$, and defined as follows:

$$\mathbf{S}^j \wedge r = \begin{cases} (r, S_1^j + r, \dots, S_{i-1}^j + r, S_{i+1}^j, \dots, S_j^j) \\ \qquad\qquad \text{if } r \in S_i^j - S_{i-1}^j \\ (S_1^j + r, S_2^j + r, \dots, S_j^j + r) \\ \qquad\qquad \text{if } r \notin S_j^j \end{cases}$$

where we assume that $S_0^j = \emptyset$.

Example. $\mathbf{S}^4 = (\{1\}, \{1,2,3\}, \{1,2,3,4\}, \{1,2,3,4,5\})$ is a nested vector that represents the following 4-element sets:

$$\begin{aligned} \mathbf{S}^4 &= \{\{1,2,3,4\}, \{1,2,3,5\}, \{1,2,4,5\}\} \\ \mathbf{S}^4 \wedge 4 &= (\{4\}, \{1,4\}, \{1,2,3,4\}, \{1,2,3,4,5\}) \\ &= \{\{1,2,3,4\}, \{1,2,4,5\}, \{1,3,4,5\}\} \\ \mathbf{S}^4 \wedge 6 &= (\{1,6\}, \{1,2,3,6\}, \{1,2,3,4,6\}, \\ &\quad \{1,2,3,4,5,6\}) \end{aligned}$$

$\mathbf{S}^4 \wedge 6$ contains 9 4-element sets: all sets in $\mathbf{S}^4$, and all sets obtained from the ones in $\mathbf{S}^4$ by replacing one item by 6 in all possible ways.

LEMMA 3.1. *Let $\mathbf{S}^j$ be a nested vector. Then,*

(a) *If $X \in \mathbf{S}^j$ and $y \in S_j^j - X$, then there is $x \in X$ such that $X + y - x \in \mathbf{S}^j$.*

(b) *$\mathbf{S}^j \wedge r$ is nested.*

(c) *If $r \in S_j^j$ then $X \in \mathbf{S}^j \wedge r$ iff $r \in X \in \mathbf{S}^j$.*

(d) *If $r \notin S_j^j$ then $X \in \mathbf{S}^j \wedge r$ iff either $X \in \mathbf{S}^j$ or $r \in X$ and $X - r + x \in \mathbf{S}^j$ for some $x \notin X$.*

Proof. Parts (b), (c), and (d) follow directly from the definitions by straightforward verification. To show part (a), order $X = \{x_1, x_2, \dots, x_j\}$, so that $\left|\{x_1, \dots, x_j\} \cap S_i^j\right| \geq i$ for $i = 1, \dots, j$. If $y \in S_p^j - S_{p-1}^j$ then $X + y - x_p \in \mathbf{S}^j$.

Induced caching problems. Define a *j-miss* to be a request to an item in location $i > j$. For j-element sets X, Y, by $\omega_{Y,\varrho}^j(X)$ we denote the minimum number of j-misses when we start from Y, serve ϱ, and end in X. Function $\omega_{Y,\varrho}^j(X)$ is called a *work function* for the *induced caching problem* with cache size j. We will simply write ω_ϱ^j when Y is the initial set $\mathbf{x}_{init}^j = \{1, 2, \dots, j\}$. The optimal cost for ϱ, equal to the minimum number of j-misses for ϱ, is $\min_X(\omega_\varrho^j(X))$. We will also omit subscript ϱ and write ω^j when the ϱ is clear from context.

If ω^j is a work function for the induced caching problem and r is a request, then the new work function after r is denoted by $\omega^j \wedge r$, and can be computed by dynamic programming:

$$\omega^j \wedge r(X) = \begin{cases} \omega^j(X) & \text{if } r \in X \\ \min_{x \in X+r} \omega^j(X + r - x) + 1 \\ & \text{if } r \notin X \end{cases}$$

For simplicity, we use the same symbol "$\wedge$" for updating vectors and work functions. Obviously, $\omega^j \wedge r(X) \geq \omega^j(X)$ for all X. It is also easy to see that work functions satisfy the following Lipschitz property: $\omega^j(X) - \omega^j(Y) \leq XY$, where XY denotes the "distance" from X to Y, $XY = |X - Y| = |Y - X|$.

We say that ω^j is *supported* by a family of j-element sets $\mathcal{X}$ if

$$\omega^j(X) = \min(\omega^j) + \min_{Y \in \mathcal{X}} XY.$$

Note that $\omega^j(X) = \min(\omega)$ iff $X \in \mathcal{X}$. A work function ω^j supported by $\mathcal{X}$ is uniquely determined by $\min(\omega^j)$ and $\mathcal{X}$.

We will first show a characterization of work functions for the induced caching problem. This characterization and its proof are similar to the one from [11] (the difference is that the [11] uses the required-fetch model).

THEOREM 3.1. *Each work function ω^j for the induced caching problem is supported by some nested vector of sets $\mathbf{S}^j$.*

Proof. The proof is by induction on the number of requests. The initial work function $\omega_{init}^j(X) = X\mathbf{x}_{init}^j$ is supported by $\mathbf{S}_{init}^j$ where $S_{init,i}^j = \mathbf{x}_{init}^i$ for each $i = 1, \dots, j$.

For the inductive step, assume that $\mathbf{S}^j$ is a nested vector that supports ω^j. Without loss of generality, assume $\min(\omega^j) = 0$. Then $\omega^j(X) = \min_{Z \in \mathbf{S}^j} XZ$ for all X. Let $\mu^j = \omega^j \wedge r$ and $\mathbf{T}^j = \mathbf{S}^j \wedge r$, and denote $\tau(X) = \min_{Z \in \mathbf{T}^j} XZ$. We will prove that $\mathbf{T}^j$ supports μ^j.

We distinguish two cases.

Case A: $r \in S^j_j$. In this case $\min(\mu^j) = 0$ and we need to show $\mu^j(X) = \tau(X)$ for all X. By the definition of the update operator and Lemma 3.1.c, $\mu^j(X) = 0$ iff $X \in \mathbf{T}^j$. From the inductive assumption, Lemma 3.1.a, and the Lipschitz property, we get

$$(3.2)\ \tau(X) - 1 \ \le\ \omega^j(X)\ \le\ \mu^j(X)\ \le\ \tau(X).$$

It now suffices to show that $\tau(X) \le \mu^j(X)$. We have three subcases.

First suppose $r \in X$. Pick $Y \in \mathbf{S}^j$ such that $\omega^j(X) = XY$. If $r \in Y$ then $Y \in \mathbf{T}^j$, so $\mu^j(X) = \omega^j(X) \ge \tau(X)$. If $r \notin Y$ then, by Lemma 3.1.a, there is $y \in Y$ for which $Y + r - y \in \mathbf{T}^j$. Since $X(Y + r - y) \le XY$, we get $\tau(X) \le X(Y + r - y) = \omega^j(X)$.

Suppose $r \notin X$ and $\omega^j(X + r - x) < \omega(X)$ for some $x \in X$. Then $\omega^j(X + r - x) = \omega^j(X) - 1$, and thus $\mu^j(X) = \omega^j(X)$. By the argument from the previous subcase, there is $Y \in \mathbf{T}^j$ such that $\omega^j(X + r - x) = (X + r - x)Y$. Then $\tau(X) \le (X + r - x)Y + 1 = \omega^j(X)$.

The last subcase is when $r \notin X$ and $\omega^j(X + r - x) \ge \omega(X)$ for all $x \in X$. Then $\mu^j(X) = \omega^j(X) + 1$. From the inequalities (3.2), we get $\tau(X) = \mu^j(X)$.

Case B: $r \notin S^j_j$. In this case $\mu^j(X) \ge 1$ for all X, and $\mu^j(X) = 1$ iff either $r \in X$ and $\omega^j(X) = 1$ or $r \notin X$ and $\omega^j(X) = 0$. From Lemma 3.1.d, we obtain $\mu^j(X) = 1$ iff $X \in \mathbf{T}^j$. By the inductive assumption, the Lipschitz property and by $\mathbf{S}^j \subseteq \mathbf{T}^j$, we have

$$(3.3)\ \tau(X)\ \le\ \omega^j(X)\ \le\ \mu^j(X)\ \le\ \tau(X) + 1.$$

Thus it suffices to show that $\mu^j(X) \ge 1 + \tau(X)$. We have two subcases.

If $r \in X$ then $\mu^j(X) = \omega^j(X)$. Pick $Y \in \mathbf{S}^j$ such that $\omega^j(X) = XY$ and pick any $y \in Y - X$. Then $Y - y + r \in \mathbf{T}^j$ and $\tau(X) \le X(Y - y + r) = XY - 1 = \omega^j(X) - 1 = \mu^j(X) - 1$.

Now suppose that $r \notin X$. Then $\omega^j(X + r - x) \ge \omega^j(X)$ for all $x \in X$, and therefore $\mu^j(X) = \omega^j(X) + 1 \ge \tau(X) + 1$.

Nested matrices. Consider a lower triangular matrix $\mathbf{S} = \left[S^j_i\right]_{1\le i\le j\le k}$, where each S^j_i is a set of items. We call $\mathbf{S}$ *nested* if it satisfies the following two conditions:

(nm1) each row of $\mathbf{S}$ is a nested vector, and

(nm2) for each $j = 1, \ldots, k-1$ there is $a \in \{0, 1, \ldots, j\}$ such that $S^j_i = S^{j+1}_i$ for $i = 1, \ldots, a$ and $S^j_i = S^{j+1}_{i+1}$ for $i = a+1, \ldots, j$.

A list $\mathbf{x} = x_1 x_2 \ldots x_k$ is *represented* by $\mathbf{S}$ iff $\left|\mathbf{x}^j \cap S^j_i\right| \ge i$ for all $1 \le i \le j \le k$. (Recall that $\mathbf{x}^j = \{x_1, \ldots, x_j\}$.) As with vectors, we will identify the matrices with the sets of lists that they represent, and we will write $\mathbf{x} \in \mathbf{S}$ when $\mathbf{x}$ is represented by $\mathbf{S}$.

We extend the update operator from vectors to matrices, by setting $\mathbf{S}\wedge r = (\mathbf{S}^j \wedge r)^k_{j=1}$, that is, by applying the update operator to each row separately.

Example. We show lists represented by the following nested matrices.

$$\begin{aligned}
\mathbf{S} &= \begin{bmatrix} \{1,2\} & & & \\ \{2\} & \{1,2\} & & \\ \{2\} & \{1,2\} & \{1,2,3,4\} & \\ \{2\} & \{1,2\} & \{1,2,4\} & \{1,2,3,4\} \end{bmatrix} \\
&= \{1234, 1243, 2134, 2143\} \\
\mathbf{S}\wedge 1 &= \begin{bmatrix} \{1\} & & & \\ \{1\} & \{1,2\} & & \\ \{1\} & \{1,2\} & \{1,2,3,4\} & \\ \{1\} & \{1,2\} & \{1,2,4\} & \{1,2,3,4\} \end{bmatrix} \\
&= \{1234, 1243\} \\
\mathbf{S}\wedge 4 &= \begin{bmatrix} \{1,2,4\} & & & \\ \{2,4\} & \{1,2,4\} & & \\ \{4\} & \{2,4\} & \{1,2,4\} & \\ \{4\} & \{2,4\} & \{1,2,4\} & \{1,2,3,4\} \end{bmatrix} \\
&= \{1243, 1423, 2143, 2413, 4123, 4213\}
\end{aligned}$$

LEMMA 3.2. *If $\mathbf{S}$ is nested then so is $\mathbf{T} = \mathbf{S}\wedge r$.*

Proof. From Lemma 3.1, each row of $\mathbf{T}$ is a nested vector. We need to show that $\mathbf{T}$ satisfies property (nm2) of nested matrices.

Fix any $1 \le j \le k-1$, and let a be the index from condition (nm2). We will show b such that $T^j_i = T^{j+1}_i$ for $i = 1, \ldots, b$ and $T^j_i = T^{j+1}_{i+1}$ for $i = b+1, \ldots, j$. We distinguish three cases.

Case A: $r \notin S^{j+1}_{j+1}$. Then

$$\begin{aligned}
\mathbf{T}^j &= (S^j_1 + r, \ldots, S^j_j + r) \\
\mathbf{T}^{j+1} &= (S^{j+1}_1 + r, \ldots, S^{j+1}_j + r, S^{j+1}_{j+1} + r)
\end{aligned}$$

so $b = a$.

Case B: $r \in S^{j+1}_{j+1} - S^j_j$. This implies that $a = j$, that is $S^j_i = S^{j+1}_i$ for $i = 1, \ldots, j$. We have

$$\begin{aligned}
\mathbf{T}^j &= (S^j_1 + r, \ldots, S^j_j + r) \\
\mathbf{T}^{j+1} &= (r, S^{j+1}_1 + r, \ldots, S^{j+1}_j + r)
\end{aligned}$$

and thus $b = 0$.

Case C: $r \in S^j_i - S^j_{i-1}$ for some $1 \le i \le j$. Let $r \in S^{j+1}_{i'} - S^{j+1}_{i'-1}$. Then

$$\begin{aligned}
\mathbf{T}^j &= (r, S^j_1 + r, \ldots, S^j_{i-1} + r, S^j_{i+1}, \ldots, S^j_j) \\
\mathbf{T}^{j+1} &= (r, S^{j+1}_1 + r, \ldots, S^{j+1}_{i'-1} + r, S^{j+1}_{i'+1}, \ldots, S^{j+1}_{j+1})
\end{aligned}$$

If $i \le a$ then $i' = i$ and $b = a$. If $i > a+1$ then $i' = i+1$ and $b = a+1$. For $i = a+1$ we have two possibilities, either $i' = i$ or $i+1$, but in both cases we get $b = a+1$.

Work functions for RLUP. A work function $\omega_{\mathbf{y},\varrho}(\mathbf{x})$ gives the minimum cost of serving the request sequence ϱ starting from list $\mathbf{y}$ and ending in $\mathbf{x}$. We will omit subscript $\mathbf{y}$ when $\mathbf{y} = \mathbf{x}_{init}$ (the initial list). We will also omit ϱ whenever it is clear from context. The optimal cost of serving ϱ is $opt(\varrho) = \min_{\mathbf{x}} \omega_\varrho(\mathbf{x})$.

Let $swapcost(\mathbf{x}, \mathbf{y})$ denote the minimum cost of paid swaps which transform $\mathbf{x}$ into $\mathbf{y}$, and let $swapcost_r(\mathbf{x}, \mathbf{y})$ be the minimum cost of transforming $\mathbf{x}$ into $\mathbf{y}$ on request r. Note that $swapcost_r(\mathbf{x}, \mathbf{y})$ could be smaller than $swapcost(\mathbf{x}, \mathbf{y})$ because of possible free swaps. We will also use notation $acost_r(\mathbf{x})$ to denote the cost of accessing r when the current list is $\mathbf{x}$. Thus $acost_r(\mathbf{x}) = c_i$ when $r = x_i$. If the current work function is ω, then after the request r the new work function $\omega \wedge r$ is determined by dynamic programming:

$$\omega \wedge r(\mathbf{y}) = \min_{\mathbf{x}} \{\omega(\mathbf{x}) + acost_r(\mathbf{x}) + swapcost_r(\mathbf{x}, \mathbf{y})\}.$$

Thus, for any ϱ and r, $\omega_\varrho \wedge r = \omega_{\varrho r}$. We say that a collection of lists $\mathcal{L}$ *supports* ω if

$$\omega(\mathbf{y}) \quad = \quad \min(\omega) + \min_{\mathbf{x} \in \mathcal{L}} swapcost(\mathbf{x}, \mathbf{y})$$

for all lists $\mathbf{y}$. If ω is supported by $\mathcal{L}$ then its values are completely determined by $\mathcal{L}$ and $\min(\omega)$.

LEMMA 3.3. *Let $d_j = c_j - c_{j-1}$ for $j = 1, \ldots, k$, where for $j = 1$ we assume that $c_0 = 0$. For all lists $\mathbf{x} = x_1 x_2 \ldots x_k$ and $\mathbf{y} = y_1 y_2 \ldots y_k$ we have*

(a) $swapcost(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^k d_j \left|\mathbf{y}^j - \mathbf{x}^j\right|$,

(b) $swapcost_r(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^k d_j \left|\mathbf{y}^j - \mathbf{x}^j - r\right|$.

Proof. We first show the "$\ge$" inequalities. Fix a sequence of swaps $\mathcal{A}$ that transforms $\mathbf{x}$ into $\mathbf{y}$. Suppose that at some step $\mathcal{A}$ executes $swap(z_i, z_j)$, for $i < j$ on a current list $\mathbf{z}$. We can think of this swap as a sequence of shifts, each moving z_j from location $\ell+1$ to location ℓ for $\ell = j-1, j-2, \ldots, i$, followed by putting z_i in location j. We will call the shifting of an item from location $\ell+1$ to location ℓ an *ℓ-promotion*. Typically, we will charge d_ℓ for the ℓ-promotion of an item; this will cause the cost of a $swap(z_i, z_j)$ to be $c_j - c_i$, as desired. However, when considering $swapcost_r(\mathbf{x}, \mathbf{y})$ with $z_j = r$, we will charge 0, because we want the cost of $swap(z_i, z_j)$ to be 0.

For any j and $y \in \mathbf{y}^j - \mathbf{x}^j$ there must be a swap in $\mathcal{A}$ that moves y from a location $j' \ge j+1$ to a location $j'' \le j$. To each such j and y we assign a j-promotion in this swap that moves y from location $j+1$ to j. Each j-promotion in $\mathcal{A}$ will be assigned to at most one pair j, y, implying the "$\ge$" inequalities.

Now we need to show the "$\le$" inequalities. For given $\mathbf{x}, \mathbf{y}$ we show a sequence $\mathcal{A}$ of swaps that transform $\mathbf{x}$ into $\mathbf{y}$ in which no item x is ℓ-promoted more than once and is ℓ-promoted once only if $x_a = x = y_b$ where $a > \ell \ge b$.

We define $\mathcal{A}$ recursively. If $\mathbf{x} = \mathbf{y}$ then $\mathcal{A}$ is empty. Otherwise, there are indices $e \le a < b \le f$ such that $y_e = x_b$ and $y_f = x_a$. Execute $swap(x_a, x_b)$, obtaining a new list $\mathbf{x}'$ with which we continue recursively. Each item moves only towards its destination in $\mathbf{y}$. Thus, (a) and (b) hold.

Now we can state the main result of this section, the characterization of work functions for RLUP.

THEOREM 3.2. *Each work function ω is supported by some nested matrix $\mathbf{S}$. Furthermore, ω can be written as $\omega(\mathbf{x}) = \sum_{j=1}^k d_j \omega^j(\mathbf{x}^j)$.*

Proof. The proof is by induction on the number of requests. The initial work function is

$$\begin{aligned} \omega_{init}(\mathbf{x}) \quad &= \quad swapcost(\mathbf{x}_{init}, \mathbf{x}) \; = \; \sum_{j=1}^k d_j \left|\mathbf{x}^j - \mathbf{x}^j_{init}\right| \\ &= \quad \sum_{j=1}^k d_j \omega^j(\mathbf{x}^j), \end{aligned}$$

for all $\mathbf{x}$. This function is supported by the nested matrix $\mathbf{S}_{init}$, where $\mathbf{S}^j_{init,i} = \mathbf{x}^i_{init}$ for $1 \le i \le j \le k$.

For the inductive step, assume that ω and $\mathbf{S}$ satisfy the theorem. Let $r \in S^\ell_\ell - S^{\ell-1}_{\ell-1}$, $\mu = \omega \wedge r$ and $\mathbf{T} = \mathbf{S} \wedge r$. By the definition of the update operator, the assumption that $\mathbf{S}$ supports ω, and the previous lemma, we get

$$\begin{aligned} \mu(\mathbf{x}) \quad &= \quad \min_{\mathbf{y}} \{\omega(\mathbf{y}) + acost_r(\mathbf{y}) + swapcost_r(\mathbf{y}, \mathbf{x})\} \\ &= \quad \min_{\mathbf{z} \in \mathbf{S}} \min_{\mathbf{y}} \{\min(\omega) + swapcost(\mathbf{z}, \mathbf{y}) \\ &\qquad + acost_r(\mathbf{y}) + swapcost_r(\mathbf{y}, \mathbf{x})\} \\ &= \quad \min_{\mathbf{z} \in \mathbf{S}} \{\min(\omega) + acost_r(\mathbf{z}) + swapcost_{\mathbf{r}}(\mathbf{z}, \mathbf{x})\} \\ &= \quad \min(\omega) + c_\ell + \min_{\mathbf{z} \in \mathbf{S}} \left\{ \sum_{j=1}^k d_j \left|\mathbf{x}^j - \mathbf{z}^{\mathbf{j}} - \mathbf{r}\right| \right\} \\ &= \quad \min(\mu) + \min_{\mathbf{z} \in \mathbf{T}} \left\{ \sum_{j=1}^k d_j \left|\mathbf{x}^j - \mathbf{z}^{\mathbf{j}}\right| \right\} \\ &= \quad \min(\mu) + \min_{\mathbf{z} \in \mathbf{T}} swapcost(\mathbf{z}, \mathbf{x}). \end{aligned}$$

Therefore μ is supported by $\mathbf{T}$. Also, we have

$$\begin{aligned}\mu(\mathbf{x}) &= \min(\mu) + \min_{\mathbf{z}\in\mathbf{T}} \mathit{swapcost}(\mathbf{z},\mathbf{x}) \\ &= \sum_{j=1}^{k} d_j \min(\omega^j) + \sum_{j=1}^{\ell-1} d_j + \min_{\mathbf{z}\in\mathbf{T}} \sum_{j=1}^{k} d_j |\mathbf{x}^j - \mathbf{z}^{\mathbf{j}}| \\ &= \sum_{j=1}^{k} d_j \mu^j(\mathbf{x}^j),\end{aligned}$$

and the proof is complete.

Compact representation. The definition of nested matrices implies the existence of a more compact representation of work functions, since it is sufficient to specify, for each row j, the index a from condition (nm2). Here we propose another simple representation for work functions. (Since this representation is not essential for the rest of the paper, we omit the correctness proofs.)

Given $\mathbf{S}$, let $S_i^k - S_{i-1}^k = x_i$, for $i = 1, \ldots, k$. $\mathbf{S}$ can be represented as $\mathbf{S} = x_1^{j_1} x_2^{j_2} \ldots x_k^{j_k}$, where each j_i is the smallest index such that $\{x_1, \ldots, x_i\} = S_l^{j_i}$ for some $l \in \{1, \ldots, j_i\}$. To define the update operator, let x_p be the requested item and $j_q = \min_{p\le i\le k} j_i$. Find the sequence $p = p_1 < p_2 < \ldots < p_t = q$ such that, for $s = 1, \ldots, t-1$, $j_{p_{s+1}}$ is the first superscript after j_{p_s} smaller than j_{p_s}. Then $\mathbf{S}\wedge x_p$ is obtained from $\mathbf{S}$ by cyclically shifting forward this sequence of superscripts and then moving x_p (with its new superscript j_q) to the beginning. More specifically,

$$\mathbf{S}\wedge x_p = x_p^{e_p} x_1^{e_1} \ldots x_{p-1}^{e_{p-1}} x_{p+1}^{e_{p+1}} \ldots x_k^{e_k}$$

where $e_p = j_q$, $e_{a_u} = j_{a_{u-1}}$ for $2 \le u \le t$, and $e_i = j_i$ otherwise.

Example. The matrices from the previous example can be expressed by $\mathbf{S} = 2^2 1^1 4^4 3^3$, $\mathbf{S}\wedge 1 = 1^1 2^2 4^4 3^3$, $\mathbf{S}\wedge 4 = 4^3 2^2 1^1 3^4$. For better illustration, consider the following larger example:

$$1^5 2^1 3^8 4^9 5^6 6^7 7^4 8^2 9^3 \wedge 3 = 3^2 1^5 2^1 4^9 5^8 6^7 7^6 8^4 9^3.$$

4 Analysis of Move-to-Front

Given a list $\mathbf{x} = x_1 \ldots x_k$ and a permutation π of the item set, by $\pi(\mathbf{x})$ we denote the list obtained by applying π to each item in $\mathbf{x}$: $\pi(\mathbf{x}) = \pi(x_1)\pi(x_2)\ldots\pi(x_k)$. If ω, μ are work functions, then we will write $\omega \approx_\pi \mu$ (resp. $\omega \preceq_\pi \mu$) if $\omega(\mathbf{x}) = \mu(\pi(\mathbf{x}))$ (resp. $\omega(\mathbf{x}) \le \mu(\pi(\mathbf{x}))$) for all lists $\mathbf{x}$. We will also write $\omega \approx \mu$ (resp. $\omega \preceq \mu$) if $\omega \approx_\pi \mu$ (resp. $\omega \preceq_\pi \mu$) for some π. Note that $\approx$ is an equivalence relation, and that $\preceq$ is a partial order that satisfies the following *update monotonicity property:* if $\omega \preceq_\pi \mu$ then $\omega\wedge r \preceq_\pi \mu\wedge(\pi(r))$.

LEMMA 4.1. *Suppose that $\omega = \omega_\varrho$, and that r precedes s in MTF's list after serving ϱ. Then $\omega\wedge r \preceq \omega\wedge s$.*

Proof. We will show that $\omega\wedge r \preceq_\pi \omega\wedge s$, where π is the permutation which interchanges r and s. Given an optimal service $\mathcal{B}$ for ϱs ending in $\mathbf{x}$, we construct a service $\mathcal{A}$ of ϱr ending in $\pi(\mathbf{x})$ with no greater cost. By Theorem 3.2, it is sufficient to consider only lists $\mathbf{x}$ in the support of the final work function, and for these lists we can assume that $\mathcal{B}$ makes only free swaps.

If r occurs before s in $\mathbf{x}$ let $\mathcal{A}$ consist of the same swaps as $\mathcal{B}$ followed by a paid swap of r and s. Any unpaid swaps $\mathcal{B}$ makes on the final request of s will be paid swaps by $\mathcal{A}$. However, this cost together with the cost of the paid swap of r and s is the same as the amount saved by serving r instead of s. Therefore, $(\omega\wedge r)(\pi(\mathbf{x})) \le (\omega\wedge s)(\mathbf{x})$.

Assume now that s occurs before r in $\mathbf{x}$. Let $\varrho = \sigma\tau$ where τ is the longest suffix of ϱ which does not contain r. Note that τ does not contain s and that σ cannot be empty and must end with r. Let $\mathcal{A}$ make the same swaps on σ as $\mathcal{B}$. If, after serving σ, r occurs after s in $\mathcal{B}$'s list, let $\mathcal{A}$ make an additional unpaid swap of r and s. From this point onward we will keep $\mathcal{A}$'s list identical to $\mathcal{B}$'s list, except that the positions of r and s may be switched. The positions of r and s will be switched iff s occurs before r in $\mathcal{B}$'s list. For any swap performed by $\mathcal{B}$ in serving τ which does not change the relative order of r and s, let $\mathcal{A}$ mimic $\mathcal{B}$. Any swap performed by $\mathcal{B}$ in serving τ which changes the relative order of r and s must include an item x and either r or s. Let $\mathcal{A}$ instead perform two swaps, first x with s then x with r.

Let $\mathbf{y} = y_1 \ldots y_k$ be $\mathcal{B}$'s list after serving ϱ. If s precedes r in $\mathbf{y}$ then in $\mathcal{A}$'s list they are interchanged. In this case, when serving r, $\mathcal{A}$ makes the same swaps as $\mathcal{B}$, but with r instead of s.

Suppose now that r precedes s in $\mathbf{y}$. In this case, $\mathcal{A}$'s list after serving ϱ is also $\mathbf{y}$. If $\mathcal{B}$ makes no swaps when serving s, then let $\mathcal{A}$ serve r by making no swaps. If $\mathcal{B}$ makes swaps with items $y_{i_a}, y_{i_{a-1}}, \ldots, y_{i_1}$, then let $\mathcal{A}$ serve $r = y_p$ by making unpaid swaps with $y_{i_b}, y_{i_{b-1}}, \ldots, y_{i_1}$, where $i_b \le p < i_{b+1}$, followed by paid swaps of s with $y_{i_a}, y_{i_{a-1}}, \ldots, y_{i_{b+1}}$. Finally, if $r \ne y_{i_b}$, then $\mathcal{A}$ makes a paid swap of s with y_{i_b}. Since the cost of the paid swaps is no more than the amount saved by serving r instead of s, $(\omega\wedge r)(\pi(\mathbf{x})) \le (\omega\wedge s)(\mathbf{x})$.

THEOREM 4.1. *MTF is optimally competitive. More specifically, if $\mathcal{A}$ is a deterministic online algorithm for RLUP then $C_\mathcal{A} \ge C_{\text{MTF}}$.*

Proof. For any sequence $\varrho = r_1 \ldots r_n$, we will show that there is a sequence $\sigma = s_1 \ldots s_n$ such that

$$(4.4)\quad \mathit{cost}_{\text{MTF}}(\varrho) \le \mathit{cost}_\mathcal{A}(\sigma) \quad\text{and}\quad \omega_\sigma \preceq \omega_\varrho.$$

If $n = 0$ then $cost_{\text{MTF}}(\varrho) = 0 = cost_{\mathcal{A}}(\sigma)$. Since $\omega_\sigma = \omega_\varrho$, we have $\omega_\sigma \preceq_{id} \omega_\varrho$, where id is the identity permutation.

Assume that (4.4) is true, and let $\varrho' = \varrho r_{n+1}$. Thus there is π such that $\omega_\sigma \preceq_\pi \omega_\varrho$. Consider the situation after MTF served ϱ and after $\mathcal{A}$ served σ. Suppose that r_{n+1} is in location j in MTF's list. There must be an item s_{n+1} which $\mathcal{A}$ does not have in the first $j-1$ positions, but MTF has $\pi(s_{n+1})$ in the first j positions. Let $\sigma' = \sigma s_{n+1}$. MTF's cost on r_{n+1} is c_j, which is no more than $\mathcal{A}$'s cost on s_{n+1}. Thus $cost_{\text{MTF}}(\varrho') \le cost_{\mathcal{A}}(\sigma')$.

By induction, by the update monotonicity, and by Lemma 4.1 we have

$$\begin{aligned} \omega_{\sigma'} &= \omega_\sigma \wedge s_{n+1} \preceq_\pi \omega_\varrho \wedge \pi(s_{n+1}) \\ &\preceq \omega_\varrho \wedge r_{n+1} = \omega_{\varrho'}. \end{aligned}$$

This implies the second inequality in (4.4), because "$\preceq$" is a partial order.

Since $\mathcal{A}$ is $C_{\mathcal{A}}$-competitive, there is a constant b such that $cost_{\mathcal{A}}(\sigma) \le C_{\mathcal{A}} \cdot opt(\sigma) + b$ for any σ. For this b and any ϱ

$$\begin{aligned} cost_{\text{MTF}}(\varrho) &\le cost_{\mathcal{A}}(\sigma) \le C_{\mathcal{A}} \cdot opt(\sigma) + b \\ &\le C_{\mathcal{A}} \cdot opt(\varrho) + b. \end{aligned}$$

Therefore, MTF is $C_{\mathcal{A}}$-competitive.

Let J be a set of indices, $J = \{j_1, \ldots, j_m\}$, where $1 = j_0 < j_1 < j_2 < \ldots < j_m \le k$. Define

$$(4.5)\; R_J = \frac{c_{j_m} + \sum_{a=1}^{m} (j_a - j_{a-1}) c_{j_a} \prod_{b=a}^{m-1} (j_{b+1} - j_b + 1)}{c_{j_m} + \sum_{a=1}^{m} \left[\prod_{b=a}^{m-1} (j_{b+1} - j_b + 1)\right] \sum_{i=j_{a-1}}^{j_a - 1} c_i}.$$

Theorem 4.2. $C_{\text{MTF}} \ge \max_J R_J$.

Proof. For each set of indices J we will construct a sequence of requests ϱ_J and compare the cost of MTF on ϱ_J to the cost of an algorithm $\mathcal{A}$ which does not modify its list.

We first define recursively a request sequence σ_J. If $J = \{j_1\}$ then let $\sigma_J = x_{j_1 - 1} x_{j_1 - 2} \ldots x_1$. If $J = \{j_1, \ldots, j_m\}$ and $m > 1$ then let $I = \{j_1, \ldots, j_{m-1}\}$ and

$$\sigma_J = \sigma_I x_{j_m - 1} \sigma_I x_{j_m - 2} \sigma_I \ldots x_{j_{m-1}} \sigma_I.$$

Then $\varrho_J = x_{j_m} \sigma_J$.

A more intuitive explanation of this request sequence is like this: Assume that the original list is $\mathbf{x}_{init} = x_1 \ldots x_k$. If MTF's list is equal $\mathbf{x}_{init}$, request x_{j_m}. Otherwise, pick the smallest i such that $j_a \le i < j_{a+1}$ and in the list of MTF x_i appears in location j_{a+1}. Note that after MTF serves ϱ_J, its list is $\mathbf{x}_{init}$, so we can repeat this process arbitrarily many times.

Fix $J = \{j_1 < j_2 < \ldots < j_m\}$. Let $J_i = \{j_1, \ldots, j_i\}$. Define $M_i = cost_{\text{MTF}}(\varrho_{J_i})$ and $A_i = cost_{\mathcal{A}}(\varrho_{J_i})$. We have $M_1 = j_1 c_{j_1}$, $A_1 = \sum_{\ell=1}^{j_1} c_\ell$, and the following recurrences hold:

$$\begin{aligned} M_{i+1} &= [j_{i+1} - j_i + 1]\left[M_i + c_{j_{i+1}} - c_{j_i}\right], \quad \text{and} \\ A_{i+1} &= \sum_{\ell=j_i}^{j_{i+1}} [A_i + c_\ell - c_{j_i}]. \end{aligned}$$

Using the above, we obtain

$$\begin{aligned} M_m &= \sum_{a=1}^{m} \left[\prod_{b=a}^{m} (j_b - j_{b-1} + 1)\right] (c_{j_a} - c_{j_{a-1}}) \\ &= c_{j_m} + \sum_{a=1}^{m} (j_a - j_{a-1}) c_{j_a} \prod_{b=a}^{m-1} (j_{b+1} - j_b + 1) \end{aligned}$$

and

$$\begin{aligned} A_m &= \sum_{a=1}^{m} \left[\prod_{b=a}^{m-1} (j_{b+1} - j_b + 1)\right] \sum_{i=j_{a-1}}^{j_a} c_i - \\ &\quad \sum_{a=1}^{m-1} \left[\prod_{b=a}^{m-1} (j_{b+1} - j_b + 1)\right] c_{j_a} \\ &= c_{j_m} + \sum_{a=1}^{m} \left[\prod_{b=a}^{m-1} (j_{b+1} - j_b + 1)\right] \sum_{i=j_{a-1}}^{j_a - 1} c_i. \end{aligned}$$

Since the optimal offline cost is no more than that of $\mathcal{A}$, we get $C_{\text{MTF}} \ge M_m / A_m = R_J$.

Conjecture 4.1. *The lower bound from Theorem 4.2 is tight, that is* $C_{\text{MTF}} = \max_J R_J$.

Sleator and Tarjan [3] proved that MTF is 2-competitive for LUP, and Irani [12] proved that the exact competitive ratio of MTF is $2k/(k+1)$. The theorem below shows that MTF has the same ratio for RLUP when the cost function is $c_i = i$.

Theorem 4.3. *If the cost function is $c_i = i$, for $i = 1, \ldots, k$, then MTF is $2k/(k+1)$-competitive.*

Proof. Since $R_J = 2k/(k+1)$ when $J = \{k\}$, we get $C_{\text{MTF}} \ge 2k/(k+1)$, by Theorem 4.2.

Fix a request sequence $\varrho = r_1 \ldots r_n$. We will call an ordered pair of items (x, y) an inversion if x occurs before y in OPT's list and x occurs after y in MTF's list. For $t = 1, \ldots, n$, let N_t be the number of inversions after $r_1, \ldots r_t$ have been served and $\Phi_t = N_t k/(k+1)$.

Fix $1 \le t \le n$. We consider the situation after $r_1 \ldots r_{t-1}$ have been served, and MTF and OPT are about to serve r_t. Suppose that r_t occurs in location a in MTF's list and in location b in OPT's list. When MTF places r_t at the head of the list at least $\max(0, a-b)$ pairs are no longer inversions and at most $b-1$ pairs become inversions. We now claim that any swap performed by OPT decreases the number of inversions by at least one. Suppose that OPT's list is $x_1 \ldots x_k$ and OPT swaps x_e with $x_b = r_t$. This swap will remove the inversions (x_ℓ, x_b) for $\ell = e, \ldots, b-1$. Therefore, even if every pair (x_ℓ, x_e) for $\ell = e+1, \ldots, b-1$ becomes an inversion, the number of inversions will decrease by one. Using these facts,

$$\begin{aligned} (4.6) \quad a + \Phi_t - \Phi_{t-1} &\leq a + \frac{k}{k+1}(2b - a - 1) \\ &\leq \frac{2k}{k+1} b. \end{aligned}$$

By summing (4.6) over all requests in ϱ, we obtain $cost_{\mathrm{MTF}}(\varrho) \leq (2k/(k+1))\, opt(\varrho)$.

5 Final Comments

We presented several results on offline and online algorithms for the relaxed list update problem (RLUP). In the offline case, we gave an efficient offline algorithm for computing the optimal cost and a simple characterization of work functions. In the online case, we proved that Move-to-Front (MTF) is optimally competitive for arbitrary cost function, and that its competitive ratio for the cost function $c_i = i$ is $2k/(k+1)$, the same as for the list update problem (LUP).

Our results may have applications to LUP. The optimal algorithm and work functions for LUP are poorly understood. For instance, the fastest known algorithm for computing an optimal service for LUP has time complexity $O(2^k k! n)$ [20], where n is the number of requests. The lack of a good characterization of work functions for LUP makes it difficult to design optimally competitive randomized algorithms. Note that the optimally competitive algorithms for paging in [5, 6] rely critically on the characterization of work functions for paging.

When the cost function is $c_i = i$, it can be shown that there are arbitrarily long request sequences ϱ, for which $opt_{\mathrm{LUP}}(\varrho) \geq (2 - \epsilon_k) \cdot opt_{\mathrm{RLUP}}(\varrho)$, where $\epsilon_k \to 0$ for $k \to \infty$ (we use subscripts to distinguish between the optimal service costs in LUP and RLUP). Nevertheless, as we have shown, MTF remains $2k/(k+1)$-competitive for RLUP in this case, indicating that work functions for RLUP provide a good approximation to work functions for LUP, and can be used for the competitive analysis, since if an algorithm for LUP is C-competitive for RLUP, it is also C-competitive for LUP.

References

[1] A. Aggarwal, B. Alpern, A.K. Chandra, and M. Snir, *A model for hierarchical memory*, Proc. 19th Symp. of Theory of Computing, 305-313 (1987).

[2] L. Belady, *A study of replacement algorithms for virtual storage computers*, IBM Systems Journal, 5:78-101 (1966).

[3] D. Sleator and R. Tarjan, *Amortized efficiency of list update and paging rules*, Communications of the ACM, 28:202-208 (1985).

[4] A. Fiat, R. Karp, M. Luby, L. McGeoch, D. Sleator, and N. Young, *Competitive paging algorithms*, Journal of Algorithms, 12:685-699 (1991).

[5] L. McGeoch and D. Sleator, *A strongly competitive randomized paging algorithm*, Journal of Algorithms, 6:816-825 (1991).

[6] D. Achlioptas, M. Chrobak, and J. Noga, *Competitive Analysis of Randomized Paging Algorithms*, Proc. 4th European Symp. on Algorithms, LNCS 1136:419-430 (1996).

[7] A. Borodin, S. Irani, P. Raghavan, and B. Schieber, *Competitive paging with locality of reference*, Journal of Computer and System Sciences, 50(2):244-258 (1995).

[8] M. Chrobak and J. Noga, LRU *is better than* FIFO, Proc. 9th Symp. on Discrete Algorithms (1998).

[9] S. Irani, A. Karlin, and S. Phillips, *Strongly competitive algorithms for paging with locality of reference*, Proc. 3rd Symp. on Discrete Algorithms, 228-236 (1992).

[10] A. Karlin, S. Phillips, and P. Raghavan, *Markov paging*, Proc. 33rd Symp. on Foundations of Computer Science, 208-217 (1992).

[11] E. Koutsoupias and C. Papadimitriou, *Beyond competitive analysis*, Proc. 35th Symp. on Foundations of Computer Science, 394-400 (1994).

[12] S. Irani, *Two results on the list update problem*, Information Processing Letters, 39:301-306 (1991).

[13] S. Albers, B. von Stengel, and R. Werchner, *A combined* BIT *and* TIMESTAMP *algorithm for the list update problem*, Information Processing Letters, 56:135-139 (1995).

[14] B. Teia, *A lower bound for randomized list update algorithms*, Information Processing Letters, 47:5-9 (1993).

[15] S. Przybylski, Cache and Memory Hierarchy Design: A Performance-Directed Approach, Morgan Kaufmann (1990).

[16] A. Smith, *Cache memories*, ACM Computing Surveys, 14(3):6-27 (1982).

[17] A. Smith, *Bibliography and Readings on* CPU *Cache memories and related topics*, Computer Architecture News, 14(1):22-42 (1986).

[18] A. Aggarwal and A. Chandra, *Virtual memory algorithms*, Proc. 20th Symp. of Theory of Computing, 173-185 (1988).

[19] A. Aggarwal, A.K. Chandra, and M. Snir, *Hierarchical memory with block transfer*, Proc. 28th Symp. on Foundations of Computer Science, 204-216 (1987).

[20] N. Reingold and J. Westbrook, *Optimum off-line algorithms for the list update problem*, Technical Report, Yale University, YALEU/DCS/TR-805 (1990).