Paid Exchanges are Worth the Price

Alejandro López-Ortiz[1], **Marc P. Renault**[*†2], **and Adi Rosén**[3]

1 **University of Waterloo, Canada**
`alopez-o@uwaterloo.ca`
2 **Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France**
`marc.renault@lip6.fr`
3 **CNRS and Université Paris Diderot, France**
`adiro@liafa.univ-paris-diderot.fr`

Abstract

We consider the list update problem as defined in the seminal work on competitive analysis by Sleator and Tarjan [12]. In this problem, a sequence of requests, consisting of items to access in a linked list, is given. After an item is accessed it can be moved to any position forward in the list at no cost (free exchange), and, at any time, any two adjacent items can be swapped at a cost of 1 (paid exchange). The cost to access an item is its current position in the list. The goal is to dynamically rearrange the list so as to minimize the total cost (accrued from accesses and exchanges) over the request sequence.

We show a lower bound of 12/11 on the worst-case ratio between the performance of an (offline) optimal algorithm that can only perform free exchanges and that of an (offline) optimal algorithm that can perform both paid and free exchanges. This answers an outstanding question that has been open since 1996 [10].

1998 ACM Subject Classification F.2.2 Analysis of Algorithms and Problem Complexity (sequencing and scheduling)

Keywords and phrases list update problem, online computation, online algorithms, competitive analysis, lower bounds

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.636

1 Introduction

The *list update problem* consists of a linked list of ℓ items and a finite request sequence. Each request is to access an item of the list. Each item access begins at the head of the list and follows the list item by item until the requested item is reached. The cost to access the i-th item in the list is thus i. Then, the requested item can be moved forward in the list at no cost and such a move is called a *free exchange*. At any time, two adjacent items may be swapped at a cost of 1 and such swaps are called *paid exchanges*. The goal is to dynamically rearrange the list over the request sequence so as to minimize the total cost of accesses and paid exchanges over the request sequence.

The list update problem (also called the list access problem) was one of the two problems studied in the seminal work on competitive analysis of Sleator and Tarjan [12] (the other being the paging problem). It is a fundamental problem in the area of algorithms that has

[*] Research supported in part by ANR project NeTOC.
[†] Work performed while the author was at LIAFA, Université Paris Diderot – Paris 7, Sorbonne Paris-Cité, France.

been intensely studied, particularly, due to its importance for compression algorithms [5]. For a recent survey on the list update problem, see [8].

In [12], Sleator and Tarjan present a 2-competitive online deterministic algorithm called MOVE TO FRONT (MTF) that Irani showed later to be an optimal online deterministic algorithm [7]. As its name implies, MTF moves every requested item to the front, using a free exchange. Also, in [7], Irani presented the first online randomized algorithm for the list update problem; it has a competitive ratio of 15/8. Reingold and Westbrook presented the first barely random online algorithm called BIT that has a competitive ratio of 7/4 [11]. The best known randomized online algorithm, COMB, of Albers et al. [2] has a competitive ratio of 1.6 and only uses free exchanges. The COMB algorithm randomly uses the barely random online algorithm BIT with a probability of 4/5 and the non-parameterized, deterministic online algorithm TIMESTAMP [1] with a probability of 1/5. The currently best randomized online lower bound is 1.50115 [4]. It should noted that all the best known online algorithms use only free exchanges [8].

The offline problem is known to NP-hard [3]. It is not known if this holds if only free exchanges are permitted. In [10], an algorithm that computes the optimal schedule that uses only paid exchanges is shown to have a running time of $O(2^\ell(\ell-1)!n)$, where ℓ is the length of the list and n is the number of requests.[1] Based on the work of [10], an alternative algorithm that computes the optimal schedule, with a running time of $O(2^\ell \ell! f(\ell) + n + \ell n)$, where $f(\ell) \leq \ell! 3^{\ell!}$, is presented in [6].

Free vs. Paid Exchanges

In [12], Sleator and Tarjan claim that an algorithm that uses paid exchanges and free exchanges can be converted to an algorithm that uses only free exchanges without increasing the cost. This claim turns out not to be true as Reingold and Westbrook gave the counterexample of the request sequence $\langle 3, 2, 2, 3 \rangle$ for a list of length 3 with a starting configuration of $1, 2, 3$ [10]. An optimal algorithm serves this sequence at a cost of 8 by moving item 1 to the back of list with paid exchanges at a cost of 2, and then serving the sequence at a cost of 6. From an enumeration of all possible schedules that use only free moves, it can be seen that an algorithm using only free exchanges serves this sequence for a cost of at least 9, implying that, in the worst case, there is at least an additive constant in the difference between the performance of an optimal algorithm that uses only free exchanges and an unrestricted optimal algorithm. Further, Reingold and Westbrook show that the opposite is true: they show that an algorithm can replace the free exchanges by paid exchanges without increasing the cost [10]. They also show that the permitted paid exchanges can be further restricted, without increasing the cost, to allow only "subset transfers" (see Definition 2 below).

The competitive ratio of 1.6 for the COMB algorithm [1] (as described above) implies an upper bound of 1.6 on the worst case ratio between the cost of an optimal algorithm restricted to free exchanges and the cost of an unrestricted optimal algorithm, over all finite request sequences.

Our Contribution

We compare the cost of an optimal algorithm that can only perform free exchanges, denoted by OPT_FREE, and an optimal algorithm that can use both paid and free exchanges, denoted

[1] As we indicate later, one can assume without loss of generality that the optimal schedule uses only paid exchanges.

by OPT. We show that there is a multiplicative gap of at least 12/11 on the worst-case ratio, over all possible finite request sequences, between the performance of OPT_FREE and OPT. Until now, it was not known if there is such a gap in an asymptotic sense. We answer this question in the affirmative, thus solving a question that has been open for almost 20 years since Reingold and Westbrook [10] gave the counterexample to the claim of Sleator and Tarjan.

As all online algorithms with currently best known competitive ratios use only free exchanges [8], our result suggests that, in order to achieve better upper bounds, it may be useful to consider online algorithms that make use of paid exchanges.

2 Preliminaries

The *list update problem* consists of a linked list of ℓ items and a finite request sequence of accesses. Each request is to access an item of the list. Each item access begins at the head of the list and there is a cost of 1 to the algorithm for each item accessed until the requested item is found. That is, the cost to access the i-th item in the list is i. Then, the requested item can be moved forward to any position in the list at no cost and such a move is called a *free exchange*. At any time, two adjacent items may be swapped at a cost of 1 and these swaps are called *paid exchanges*. The goal is to dynamically rearrange the list over the request sequence so as to minimize the total cost of accesses and paid exchanges over the request sequence.

Note that, in the offline version of the list update problem (as defined above), the input is still a request sequence that must be served in order. The difference between the offline and online versions is that the offline algorithm has knowledge of the entire request sequence whereas, in the online version, a request is not revealed until all prior requests in the sequence have been served.

For an algorithm ALG and a request sequence σ, we denote the cost to ALG to serve σ by ALG(σ).

We will use OPT to denote an unrestricted optimal (offline) algorithm, and we will use OPT_FREE to denote an optimal (offline) algorithm restricted to using only free exchanges. For the request sequences, we will denote multiple requests in a row to the same item by using exponents, e.g. x^k means that x is requested k times in a row.

In [9, 10], Reingold and Westbrook consider the offline version of the list update problem and show several properties of an offline optimum that uses both paid and free exchanges such as the following lemma.

▶ **Lemma 1.** [9][Cor. 3.2] *If an item x is requested* 3 *or more times consecutively, then an optimal offline algorithm must move it to the front before the second access.*

In [9, 10], Reingold and Westbrook also define the notion of a subset transfer and show that there exists an optimal algorithm that only performs such moves.

▶ **Definition 2** (Subset Transfer). Let x be a requested item. A *subset transfer* is a move, performed just before x is accessed, of a subset of the items ahead of x in the list to the position immediately after x such that the relative order of the items in the subset is maintained.

▶ **Theorem 3.** [10][Thm. 2] *There is an optimal offline algorithm that does only subset transfers.*

Using Lemma 1 and Theorem 3, we get the following theorem that states that, for any sequence consisting of at least 3 consecutive requests to every item, MTF is OPT_FREE.

▶ **Theorem 4.** *Let $\sigma = \left\langle x_1^{k_1}, \ldots, x_j^{k_j} \right\rangle$, where, for all i, $k_i \geq 3$ and, for $i < j$, $x_i \neq x_{i+1}$. For any initial list configuration, there exists an* OPT_FREE *that moves each x_i, $1 \leq i \leq j$, to the front of the list immediately after the first access to x_i of $x_i^{k_i}$ in σ.*

Proof. By Lemma 1, an (unrestricted) optimal algorithm must move each x_i, $1 \leq i \leq j$, of σ to the front before the second request to that item. Furthermore, by Theorem 3, there exists such optimal algorithm that only performs subset transfers; denote this optimal algorithm by OPT. Observe that if OPT does not move x_i to the front immediately before the first request to x_i, but does move x_i to the front immediately before the second request to x_i, then it cannot be optimal, since smaller cost could be achieved by moving x_i to the front immediately before the first request to x_i. We conclude that OPT is an optimal, subset-transfer-only, algorithm, that moves each x_i, $1 \leq i \leq j$, of σ to the front immediately before the first request to x_i. Observe now that since OPT is a subset-transfer-only algorithm, then OPT does not perform any other rearrangements in the list while processing σ.

The action by OPT of moving x_i to the front by subset transfer immediately before the first request to x_i, and then accessing x_i k_i times, can be accomplished for the same cost by an algorithm restricted to free exchanges. This is done by first accessing x_i (on the first request to x_i), then moving x_i to the front by a free exchange, and then accessing x_i for the remaining $k_i - 1$ times. It follows that there exists an algorithm restricted to free moves, that on σ moves every x_i, $1 \leq i \leq j$, to the front immediately after the first request to x_i, and its cost is equal to the cost of the optimal unrestricted algorithm for σ. This algorithm must therefore be OPT_FREE for σ. ◀

Informally, the next theorem shows that, on a series of sequential requests, it is not to the advantage of ALG_FREE to delay moving the requested item forward. That is, for an arbitrary algorithm that only performs free exchanges, denoted by ALG_FREE, and, for a sequence of consecutive requests to an item x, such that β is the position closest to the head of the list to which x is moved by the end of these consecutive requests, if ALG_FREE would move x to β immediately after the first request, it would not increase its cost. This holds for both offline and online algorithms, but online algorithms generally are not able take advantage of this fact given that they do not in general know the subsequent requests.

▶ **Theorem 5.** *Let $\sigma = \langle \sigma_1, \nu, \sigma_2 \rangle$, where ν is at least two consecutive requests to the same item x. Let β be the position of x immediately after ν for an arbitrary algorithm* ALG_FREE. *There exists an algorithm* ALG_FREE′ *that moves x to β immediately after the first request of ν such that* ALG_FREE′$(\sigma) \leq$ ALG_FREE(σ), *and* ALG_FREE′ *serves σ_1 and σ_2 exactly as* ALG_FREE.

Proof. The algorithm ALG_FREE′ is defined to serve σ_1 in the same manner as ALG_FREE, to then move x to position β immediately after the first request of ν, and to serve σ_2 in the same manner as ALG_FREE. Note that the list configurations of ALG_FREE′ and ALG_FREE match prior to and after serving ν. Therefore, the cost to both algorithms is the same for σ_1 and σ_2.

Since ALG_FREE uses only free moves, i.e. moves of items towards the head of the list, it follows that the cost of ALG_FREE′ for all requests in ν is no more than the cost of ALG_FREE for those requests. Therefore, ALG_FREE′$(\sigma) \leq$ ALG_FREE(σ). ◀

3 Lower Bound for OPT_FREE

In this section, we give a lower bound for the free move optimal offline algorithm as compared to the unrestricted optimal offline algorithm. That is, we are comparing the power of paid exchanges and free exchanges versus only free exchanges. We show that, for the case of a list of length at least 3, the ratio between the performance of OPT_FREE and that of OPT is at least $12/11 > 1.09$ in the worst case. More formally, we show that there exists an infinite family of finite request sequences σ_r, $r > 0$, such that the cost of an offline algorithm that can use paid exchanges, PAID, increases with r, and such that $\frac{\text{OPT\_FREE}(\sigma_r)}{\text{OPT}(\sigma_r)} \geq \frac{\text{OPT\_FREE}(\sigma_r)}{\text{PAID}(\sigma_r)} \geq 12/11$. This implies that, for any $\varepsilon > 0$ and any additive constant η that does not depend on the request sequence, there does not exist a free exchange algorithm, ALG_FREE, such that ALG_FREE$(\sigma) \leq \left(\frac{12}{11} - \varepsilon\right)$ OPT$(\sigma) + \eta$ for all σ.

To prove the claim, we use a list of length 3 and we begin by defining a request sequence $R(L)$. For a given initial list configuration L, we define the request sequence $R(L)$ and a deterministic offline algorithm PAID that uses paid exchanges. By relabelling the list of PAID after having served $R(L)$ to match L, we can define an arbitrarily long request sequence σ_r consisting of repeated requests to $R(L)$ based on a relabelling of the list state of PAID after each $R(L)$. Our result applies to a list of length at least 3: If the list has a length greater than 3, we can ignore all but 3 items. Hence, without loss of generality, we only consider lists of length 3.

Line of Proof

As indicated above, our proof uses arbitrarily long request sequences, σ_r, $r \geq 1$ that are built by a repeated concatenation of r short request sequence $R(L)$, defined using a relabelling of the list state of PAID after each $R(L)$. We first prove two claims related to a single short request sequence $R(L)$. Namely, that PAID serves $R(L)$ starting with list configuration L at cost of 11; and that any OPT_FREE that serves $R(L)$ starting with list configuration L has cost at least 12. This however only repeats the claim of Reingold and Westbrook as to the existence of a request sequence with an additive difference between the optimal performance with free exchanges only and the optimal performance with both free and paid exchanges. We then concatenate these short request sequences to create a long request sequence. Observe that a multiplicative gap does not follow from such a concatenation. Indeed, an optimal algorithm that uses only free exchanges could potentially pay more than 12 for a given request sequence $R(L)$, reach a different list configuration, and then be able to serve the next $R(L)$ with cost less than 12, thus paying in total no more than 23 for the two sequences (or have such a phenomenon over a sequence of more than two sequences $R(L)$). To overcome this difficulty we prove that, for the long sequences that we consider, σ_r, any OPT_FREE must reach the same configuration as PAID does at the end of each $R(L)$. We can then conclude that for σ_r the cost of PAID is $11r$ and the cost of any OPT_FREE is at least $12r$.

Offline Paid Exchange Algorithm

For a list of length 3 with a starting list configuration $L = y, x_1, x_2$, we define the request sequence $R(L) = \langle x_2, x_1^3, x_2^3 \rangle$.

Let PAID be an unrestricted offline algorithm for $R(L)$ defined as follows. Before the first request of $R(L)$, using two paid exchanges, x_1 and x_2 are moved to the front of the list. Then, immediately before the second request to any x_i, $1 \leq i \leq 2$, PAID moves x_i to the front.

Immediately from the definition of PAID, we have the following facts.

▶ **Fact 6.** *Given a starting list configuration of $L = y, x_1, x_2$, after serving $R(L)$, the list configuration of* PAID *is x_2, x_1, y.*

▶ **Fact 7.** *Given a starting list configuration of $L = y, x_1, x_2$, the cost of* PAID *to serve $R(L)$ is* 11.

Proof. The cost to bring x_1, x_2 to the front by paid exchanges is 2 and the list configuration is now x_1, x_2, y. The cost of the first access to x_2 is 2, the cost to the next three requests of x_1 is 3. The second access to x_2 costs 2 and then x_2 is brought to the front and the remaining two accesses cost 2. Overall, the cost to PAID is 11. ◀

Arbitrarily Long Request Sequences

For an initial list configuration of $L = y, x_1, x_2$, from Fact 6, the configuration of the list of PAID after serving $R(L)$ is x_2, x_1, y. Therefore, after serving $R(L)$, with a relabelling of the list of PAID to that of L, $R(L)$ can subsequently be requested again, and this can be repeated to create arbitrarily long request sequences. That is, if $L' = x_2, x_1, y$ (as is the list configuration PAID after serving $R(L)$ for $L = y, x_1, x_2$), then $R(L') = \langle y, x_1^3, y^3 \rangle$.

Let $\sigma_r = \langle R_1(L_1), R_2(L_2), \ldots, R_r(L_r) \rangle$ such that $R_j(L_j)$ is based on L_j, where L_j is the configuration of the list of PAID after serving $R_1, \ldots, R_{j-1}$ for $1 < j \leq r$ and $L_1 = L$ is the initial configuration of the list. We will use the term round to signify a subsequence $R(L)$ in σ_r.

Optimal (Offline) Free Exchange Algorithm

Let MTF be the algorithm that moves every requested item to the front. Immediately from the definition of MTF, we have the following fact.

▶ **Fact 8.** *Given a starting list configuration of $L = y, x_1, x_2$, after serving $R(L)$, the list configuration of* MTF *is x_2, x_1, y.*

Note that, when starting from the same initial list configuration and serving $R(L)$, the list configuration of MTF is exactly that of PAID after serving $R(L)$.

For an initial configuration $L = y, x_1, x_2$ and $R(L) = \langle x_2, x_1^3, x_2^3 \rangle$, the following lemma shows that MTF is an optimal free move algorithm for $R(L)$.

▶ **Lemma 9.** *For an initial list configuration $L = y, x_1, x_2$,* MTF$(R(L)) = 12$ *and* OPT_FREE$(R(L)) = 12$.

Proof. Irrespective of the specific free exchange algorithm, the access cost for the first request is 3 and there are 3 possible list configurations after the access. They are y, x_1, x_2; y, x_2, x_1; and x_2, y, x_1 (this last configuration corresponds to that of MTF). By Theorem 4, applied to the suffix of $R(L)$, after serving the first request of $R(L)$, $\langle x_1^3, x_2^3 \rangle$,every OPT_FREE moves x_1 and x_2 to the front of the list on the next request to each item. Table 1 summarizes the costs of the 3 possible ways to serve $R(L)$, as a function of the list configuration after the first request. The actions of MTF on $R(L)$ correspond to the x_2, y, x_1 column which is a minimum. ◀

We note that our proof will go through also if instead of using MTF we would use the algorithm that results in the list configuration as defined in the first configuration in the table.

Table 1 For an initial list configuration of $L = y, x_1, x_2$, this table summarizes the potential optimal free exchange algorithms for $R(L) = \left\langle x_2, x_1^3, x_2^3 \right\rangle$. From Theorem 4, we know that after the first request every OPT_FREE moves all the items to the front of the list for the remaining requests. Therefore, the only variable is the configuration of the list immediately after the first request. Columns $3-5$ represent the three possible list configurations. Column 1 is the index in $R(L)$ of the request listed in column 2. From the table, the first and third list configurations are optimal, and MTF corresponds to the third list configuration.

		List Configuration		
Request		y, x_1, x_2	y, x_2, x_1	x_2, y, x_1
1	x_2	3	3	3
2	x_1	2	3	3
3	x_1^2	2	2	2
6	x_2	3	3	2
7	x_2^2	2	2	2
Total:		12	13	12

The Last Round of σ_r

In the following lemma, we show that any OPT_FREE moves any item x to the front of the list immediately after the first access of three consecutive requests to x in $R_r(L_r)$ of σ_r, i.e. in the last round of σ_r.

▶ **Lemma 10.** *For $\sigma_r = \langle R_1(L_1), \ldots, R_r(L_r)\rangle$, every* OPT_FREE *moves any item x to the front of the list immediately after the first access of three consecutive requests to x in $R_r(L_r)$, where $L_1 = y, x_1, x_2$ and L_j, $1 < j \leq r$, is the list configuration of* PAID *after serving $\langle R_1(L_1), \ldots, R_{j-1}(L_{j-1})\rangle$.*

Proof. Let $L_r = y, x_1, x_2$ and let A be an arbitrary OPT_FREE algorithm. By way of contradiction, assume that A does not move some $x_i \in R_r(L_r)$ to the front immediately after the first request to $x_i^3 \in R_r(L_r)$.

Let $\sigma' = \langle R_1(L_1), \ldots, R_{r-1}(L_{r-1}), x_2\rangle$ and $\sigma'' = \left\langle x_1^3, x_2^3 \right\rangle$ (note that $\sigma_r = \langle \sigma', \sigma''\rangle$). Define $\hat{A}$ to be a free move algorithm that serves σ' exactly as A and moves all $x_j \in \sigma''$ to the front immediately after the first request to each item in σ''.

Since A and $\hat{A}$ serve σ' in the same manner, $\hat{A}(\sigma') = A(\sigma')$ and the list configurations of A and $\hat{A}$ are the same immediately after σ'. From Theorem 4, $\hat{A}$ is OPT_FREE over the remainder of the sequence. But, given the list configuration of both A and $\hat{A}$ after serving σ', starting with that list configuration, $\hat{A}(\sigma'') = \text{OPT\_FREE}(\sigma'') < A(\sigma'')$. Therefore, $\hat{A}(\sigma_r) = A(\sigma') + \text{OPT\_FREE}(\sigma'') < A(\sigma_r)$ which contradicts the fact that A is an optimal free exchange algorithm. ◀

The Rest of σ_r

In the next lemma, we show that the property proved in Lemma 10 for the last round of σ_r can be proved for all of σ_r. Namely, we show that for σ_r there exists an OPT_FREE that moves any item x to the front after the first access of any three consecutive requests to the same item. We note that Theorem 4 holds only for the specific type of sequence defined in the statement of that theorem, and that the property proved in the next lemma does not hold in general for an arbitrary sequence. For example, it can be verified that the sequence $\langle 5, 5, 5, 4, 3, 2, 1, 4, 3, 2, 1, 4, 3, 2, 1\rangle$ (starting with list configuration $1, 2, 3, 4, 5$) can be served by OPT__FREE at cost of 44, while if ALG__FREE moves item 5 to the front immediately after the first request to item 5, then the cost of ALG__FREE is at least 45.

▶ **Lemma 11.** *For $\sigma_r = \langle R_1(L_1), \ldots, R_r(L_r)\rangle$, there exists an* OPT__FREE *that moves any item x to the front of the list immediately after the first access of three consecutive requests to x, where $L_1 = y, x_1, x_2$ and L_j, $1 < j \leq r$, is the list configuration of* PAID *after serving $\langle R_1(L_1), \ldots, R_{j-1}(L_{j-1})\rangle$.*

Proof. In this proof, for σ_r, we consider an arbitrary OPT__FREE algorithm A and show that, if the property does not hold for A, then there exists another OPT__FREE algorithm A' that does move every item x to the front of the list immediately after the first access of three consecutive requests to x, and such that $A'(\sigma_r) \leq A(\sigma_r)$. This will be done by defining a sequence of algorithms A_q, starting with $A_0 = A$, and by reverse induction on i and j over all the $x_i^3 \in R_j(L_j) \in \sigma_r$. That is, we consider the rounds from $R_r(L_r)$ to $R_1(L_1)$ and the consecutive three requests in each round from the last consecutive three requests to the first. For each x^3, if A_q does not move x to the front immediately after the first request, we define A_{q+1}, based on A_q, such that the desired property holds for x^3 and all subsequent consecutive three requests, and we show that the cost does not increase.

In the proof, we use the following notations. Let x^3 be three consecutive requests in $R_j(L_j)$ for which A_q does not have the desired property. We will denote all the requests in σ_r before x^3 by σ_1. The requests after x^3 will be denoted by σ_2. Note that σ_2 could be an empty sequence. For the analysis, we will often (Case 2 and Case 3 below) further partition σ_2 into disjoint subsequences $\langle\sigma_3, \ldots, \sigma_p\rangle$ such that $\sigma_r = \left\langle\sigma_1, x^3, \sigma_3, \ldots, \sigma_p\right\rangle$. At a risk of a slight abuse of notation, we will denote the cost of a subsequence of an arbitrary σ_r to an algorithm, ALG, that serves all of σ_r, as $\text{ALG}(r_i, \ldots, r_j) = \text{ALG}(r_1, \ldots, r_j) - \text{ALG}(r_1, \ldots, r_{i-1})$, where the prefix and the suffix are understood implicitly. That is, $\text{ALG}(r_i, \ldots, r_j)$ is the cost accrued by ALG over the requests $r_i, \ldots, r_j$ of σ_r given that ALG has served the prefix $r_1, \ldots, r_{i-1}$ and will serve the remaining requests. Therefore, we have that $\text{ALG}(\sigma_r) = \text{ALG}(\sigma_1) + \text{ALG}(x^3) + \text{ALG}(\sigma_3) + \cdots + \text{ALG}(\sigma_p)$. Further note that by Theorem 5, we can assume without loss of generality that A_q does not move x further ahead in the list on the second or third requests of x^3.

Definition of $\hat{A}_q$

We first define an algorithm $\hat{A}_q$ that we use extensively in the proof. For $\sigma = \left\langle\sigma_1, x_i^3, \sigma_2\right\rangle$ and algorithm A_q as defined previously, let $\hat{A}_q$ be an algorithm that serves σ_1 in the same manner as A_q and then moves x_i from position $\alpha > 1$ to the front of the list at the first request of x_i. Immediately after serving x_i, the configuration of the list of A_q is some B, x_i, C and the configuration of the list of $\hat{A}_q$ is x_i, B, C, where B is the set of items ahead of x_i in the configuration of A_q at this point and C is the set of items behind x_i in the configuration of A_q. As long as the list configurations of A_q and $\hat{A}_q$ differ, for each $x_j \in \sigma_2$, if A_q moves x_j to the front, $\hat{A}_q$ moves x_j to front. Otherwise, $\hat{A}_q$ does not move x_j forward at all. Once the list configurations of A_q and $\hat{A}_q$ match, $\hat{A}_q$ will serve the remaining requests exactly as A_q. Note that it is possible that the list configuration of $\hat{A}_q$ may never match that of A_q (see Case 1 below).

From the definition of $\hat{A}_q$, and the fact that the list has length of 3, we have the following useful properties.

$$\hat{A}_q(\sigma_1) = A_q(\sigma_1)\ , \quad (1) \qquad |C| = 2 - |B|\ , \quad (3)$$

$$|B| \geq 1\ , \quad (2) \qquad \beta = |B| + 1\ , \quad (4)$$

where β is the position to which x_i is moved by A_q. Further, given that A_q moves x_i from α to β, $1 < \beta \leq \alpha$, and $\hat{A}_q$ moves x_i from α to the front of the list, we have the following properties.

$$A_q(x_i^3) = \alpha + 2\beta \ , \tag{5}$$

$$\hat{A}_q(x_i^3) = \alpha + 2 \tag{6}$$

$$= A_q(x_i^3) - 2\beta + 2 \ , \tag{7}$$

where (7) follows by replacing α in (6) by the value of α in (5).

We now turn to the inductive proof. For a list of length 3, there are two alternating list configurations for PAID (i.e. values for L_j) before each $R_j(L_j)$: y, x_1, x_2 and x_2, x_1, y. Therefore, x_1 is requested in every $R_j(L_j)$, and x_2 and y are requested in alternating $R_j(L_j)$'s.

For $x_i \in R_j(L_j)$, which is the last point in σ_r for which A_q does not move x_i to the front immediately after the first request of three consecutive requests, we can distinguish between three cases: (1) x_i is never requested again in σ_r; (2) x_i is requested again in $R_{j+1}(L_{j+1})$, i.e. in the next round; (3) x_i is requested again in $R_{j+2}(L_{j+2})$, i.e. in the round after the next round. Note that this partitioning is exhaustive.

At each inductive step such that A_q does not have the desired property, we define an algorithm A_{q+1} based on A_q, for $q \geq 0$, and show that $A_{q+1}(\sigma_r) \leq A_q(\sigma_r)$. This is done by case analysis over the three cases defined above.

Case 1: $x_i \in R_j(L_j)$ is never requested again in σ_r.

Recall that $\sigma_r = \langle \sigma_1, x_i^3, \sigma_2 \rangle$. When $j = r$, this case follows immediately from Lemma 10 by defining A_{q+1} to be the algorithm defined in the proof of Lemma 10.

When $j < r$, we define A_{q+1} to be $\hat{A}_q$ as defined above. For a list of length 3, this only occurs when $j = r - 1$ and $i = 2$, where $L_{r-1} = y, x_1, x_2$ and, hence, $R_{r-1}(L_{r-1}) = \langle x_2, x_1^3, x_2^3 \rangle$. For $L_{r-1} = y, x_1, x_2$, $R_r(L_r) = \langle y, x_1^3, y^3 \rangle$ and x_2 is not requested.
Denote $\sigma_1 = \langle R_1(L_1), \ldots, R_{r-2}(L_{r-2}), x_2, x_1^3 \rangle$,
$\sigma_2 = \langle y, x_1^3, y^3 \rangle$.

Cost for $\sigma_2 = \langle y, x_1^3, y^3 \rangle$. By the induction hypothesis we know that A_q moves x_1 and y to the front of the list on the first request to x_1 and on the second request to y. It follows that the configurations of the lists of $\hat{A}_q$ and A_q will match before the the third request to y is processed.

If y is in B, then the total cost to access y for $\hat{A}_q$ over σ_2 is at most 2 more than that of A_q over σ_2. This follows from the fact that there are two requests to y before A_q must move y to the front, according to the induction hypothesis and, if y is in B, then $\hat{A}_q$ has x_1 in front of y, whereas A_q does not.

If y is in C, the total cost to access y for $\hat{A}_q$ over σ_2 is at most 1 more than that of A_q over σ_2. This can occur if, on the first access to y in σ_2, A_q were to move y between x_1 and x_2 in its list. Then, on the second access, y is one item closer to the front in the list of A_q as compared to the list of $\hat{A}_q$.

By the induction hypothesis, A_q must move x_1 to the front on the first request to x_1 in σ_2. Therefore, if x_1 is in B, the first access costs 1 more to $\hat{A}_q$ as compared to A_q and, if x_1 is in C, the cost for the first access is the same for both $\hat{A}_q$ and A_q.

This gives that for σ_2,

$$\begin{aligned}\hat{A}_q(\sigma_2) &\le A_q(\sigma_2) + 2|B| + |C| - 1\\ &= A_q(\sigma_2) + |B| + 1 \ , \end{aligned} \tag{8}$$

where the last line follows by applying (3).

Using (1), we get

$$\begin{aligned}\hat{A}_q(\sigma_r) &= A_q(\sigma_1) + \hat{A}_q(x_i^3, \sigma_2)\\ &\le A_q(\sigma_r) - 2\beta + |B| + 3 \ , \text{ using (7) and (8)},\\ &= A_q(\sigma_r) - |B| + 1 \ , \text{ using (4)},\\ &\le A_q(\sigma_r) \ , \text{ by (2)}.\end{aligned}$$

Case 2: $x_i \in R_j(L_j)$ and $x_i \in R_{j+1}(L_{j+1})$, i.e. x_i is requested in the next round.

Recall that $\sigma_r = \langle \sigma_1, x_i^3, \sigma_2 \rangle$. For $L_j = y, x_1, x_2$, $R_j(L_j) = \langle x_2, x_1^3, x_2^3 \rangle$ and $R_{j+1}(L_{j+1}) = \langle y, x_1^3, y^3 \rangle$. For this case, we define A_{q+1} to be $\hat{A}_q$ as defined above.

Define $\sigma_1 = \langle R_1(L_1), \ldots, R_{j-1}(L_{j-1}), x_2 \rangle$,
$\sigma_3 = \langle x_2^3, y, x_1^3 \rangle$,
$\sigma_4 = \langle y^3, R_{j+2}(L_{j+2}), \ldots, R_r(L_r) \rangle$.

Note that $\sigma_2 = \langle \sigma_3, \sigma_4 \rangle = \langle x_2^3, R_{j+1}(L_{j+1}), \ldots, R_r(L_r) \rangle$.

Cost for $\sigma_3 = \langle x_2^3, y, x_1^3 \rangle$. After serving x_1^3, the configuration of the list of $\hat{A}_q$ is x_1, B, C and the list of A_q is B, x_1, C. By the induction hypothesis, A_q will move x_2 to the front on the first request to x_2 in σ_3. This request and the request to y will each cost 1 more to $\hat{A}_q$ than to A_q if they are in B. If they are in C, there is no additional cost to $\hat{A}_q$ as compared to A_q. Finally, on the first request to x_1 in σ_3, x_1 is no further from the front in $\hat{A}_q$ than it is in A_q. Then, by the induction hypothesis, A_q moves x_1 to the front for the remaining requests to x_1 in σ_3 as does $\hat{A}_q$. Therefore,

$$\hat{A}_q(\sigma_3) \le A_q(\sigma_3) + |B| \ . \tag{9}$$

List Configuration after σ_3. By the induction hypothesis, A_q will move x_2 and x_1 to the front of the list immediately after the first access to each one in σ_3. Consider the state of the lists of A_q and $\hat{A}_q$ immediately after serving σ_3, depending on whether or not A_q moves y to the front. If A_q does not move y to the front of the list, the configuration of its list will be x_1, x_2, y, and, by the definition of $\hat{A}_q$, its list configuration will also be x_1, x_2, y. If A_q does move y to the front of the list, the configuration of its list will be x_1, y, x_2, and, by the definition of $\hat{A}_q$, its list configuration will also be x_1, y, x_2.

Cost for $\sigma_4 = \langle y^3, R_{j+2}, \ldots, R_r \rangle$. After serving σ_3, the configurations of the lists of $\hat{A}_q$ and of A_q are the same. Therefore,

$$\hat{A}_q(\sigma_4) = A_q(\sigma_4) \ . \tag{10}$$

Summing (1), (7), (9), and (10), we get that the cost for $\hat{A}_q$ over σ_r is

$$\begin{aligned}\hat{A}_q(\sigma_r) &\le A_q(\sigma_r) - 2\beta + 2 + |B|\\ &= A_q(\sigma_r) - |B| \ , \text{ using (4)},\\ &< A_q(\sigma_r) \ , \text{ by (2)}.\end{aligned}$$

Case 3: $x_i \in R_j(L_j)$ and $x_i \in R_{j+2}(L_{j+2})$, i.e. x_i is requested in the round after next.

Recall that $\sigma_r = \langle \sigma_1, x_i^3, \sigma_2 \rangle$. We define A_{q+1} to be $\hat{A}_q$. For $L_j = y, x_1, x_2$, $R_j(L_j) = \langle x_2, x_1^3, x_2^3 \rangle$, $R_{j+1}(L_{j+1}) = \langle y, x_1^3, y^3 \rangle$, and $R_{j+2}(L_{j+2}) = \langle x_2, x_1^3, x_2^3 \rangle$.
Define $\sigma_1 = \langle R_1(L_1), \ldots, R_{j-1}(L_{j-1}), x_2, x_1^3 \rangle$,
$\sigma_3 = \langle y, x_1^3, y^3 \rangle$, and
$\sigma_4 = \langle R_{j+2}(L_{j+2}), \ldots, R_r(L_r) \rangle$.
Note that $\sigma_2 = \langle \sigma_3, \sigma_4 \rangle$.

Cost for $\sigma_3 = \langle y, x_1^3, y^3 \rangle$. After serving σ_1, x_2^3, the configuration of the list of $\hat{A}_q$ is x_2, B, C and the configuration of the list of A_q is B, x_2, C. By the induction hypothesis, A_q will move y to the front of the list on the second request to y in σ_3 and x_1 to the front of the list on the first request to x_1 in σ_3. This is exactly the same scenario as σ_2 for Case 1. Similarly to (8) for Case 1 we have,

$$\hat{A}_q(\sigma_3) \le A_q(\sigma_3) + |B| + 1 \ . \tag{11}$$

List Configuration after σ_3. Since both y and x_1 are moved to the front of the list in σ_3 by both A_q and $\hat{A}_q$, the configuration of the lists of both A_q and $\hat{A}_q$ is y, x_1, x_2 after σ_3.

Cost for $\sigma_4 = \langle R_{j+2}(L_{j+2}), \ldots, R_r(L_r) \rangle$. After serving σ_3, the configurations of the lists of $\hat{A}_q$ and of A_q are the same. Therefore,

$$\hat{A}_q(\sigma_4) = A_q(\sigma_4) \ . \tag{12}$$

Summing (1), (7), (11), and (12), we get that the cost for $\hat{A}_q$ over σ_r is

$$\begin{aligned} \hat{A}_q(\sigma_r) &\le A_q(\sigma_r) - 2\beta + |B| + 3 \\ &= A_q(\sigma_r) - |B| + 1 \ , \text{ using (4)}, \\ &\le A_q(\sigma_r) \ , \text{ by (2)}. \end{aligned}$$

To conclude, for each of the three cases possible at each inductive step, we have shown that there exists an algorithm with the desired property. Overall, we have shown that $A'(\sigma_r) = A_q(\sigma_r) \le \cdots \le A_0(\sigma_r) = A(\sigma_r)$ which concludes the proof. ◀

For σ_r as defined above, Lemma 11 shows that there exists an OPT_FREE that moves x to the front when x is requested at least three times in a row. Let OPT_FREE* be such an OPT_FREE. It follows that the list configuration of OPT_FREE* after each $R_j(L_j) \in \sigma_r$ is the same as that of PAID. For an initial list configuration of $L = y, x_1, x_2$, Lemma 9 shows that the algorithm MTF is an optimal free move algorithm for $R(L)$. Since the list configuration of MTF after serving $R_1(L_1), \ldots, R_j(L_j)$, $1 \le j \le r$, is the same as OPT_FREE*, combined with the previous fact, this implies that MTF is an optimal free exchange algorithm for σ_r. Hence, MTF serves all $R_{j+1}(L_{j+1})$ at a cost no more than that of OPT_FREE*. This is formally stated in the following lemma.

▶ **Lemma 12.** *For $\sigma_r = \langle R_1(L_1), \ldots, R_r(L_r) \rangle$, MTF$(\sigma_r)$ = OPT_FREE(σ_r), where $L_1 = y, x_1, x_2$ and L_j, $1 < j \le r$, is the list configuration of* PAID *after serving $\langle R_1(L_1), \ldots, R_{j-1}(L_{j-1}) \rangle$.*

Proof. By Lemma 11, there exists an OPT_FREE that will have the same configuration as PAID and MTF immediately before $R_j(L_j)$, $1 \leq j \leq r$. Let OPT_FREE* be such an OPT_FREE. Since the list configuration of MTF and OPT_FREE* match prior to serving every $R_j(L_j)$, Lemma 9 implies that $\text{MTF}(\sigma_r) = \text{OPT\_FREE}(\sigma_r)$. ◀

Using the fact that, for any $r > 0$, MTF is an optimal free exchange algorithm for σ_r, we can, in the following lemma and theorem, give a lower bound on the worst-case ratio between $\text{OPT\_FREE}(\sigma)$ and $\text{OPT}(\sigma)$ by analysing the ratio between $\text{MTF}(\sigma_r)$ and $\text{PAID}(\sigma_r)$ for $\sigma_r = \langle R_1(L_1), \ldots, R_r(L_r)\rangle$, where $L_1 = y, x_1, x_2$ and L_j, $1 < j \leq r$, is the list configuration of PAID after serving $\langle R_1(L_1), \ldots, R_{j-1}(L_{j-1})\rangle$.

▶ **Lemma 13.** *For* $r > 0$, $\frac{\text{OPT\_FREE}(\sigma_r)}{\text{OPT}(\sigma_r)} \geq \frac{\text{OPT\_FREE}(\sigma_r)}{\text{PAID}(\sigma_r)} = \frac{12}{11} > 1.09$.

Proof. Let $\sigma = \langle R_1(L_1), \ldots, R_r(L_r)\rangle$, where $L_1 = y, x_1, x_2$ and L_j, $1 < j \leq r$, is the list configuration of PAID after serving $\langle R_1(L_1), \ldots, R_{j-1}(L_{j-1})\rangle$.

From Lemma 12 and Lemma 9, MTF is an optimal free move algorithm for σ_r with a cost of $12r$ and, from Fact 7, the cost of PAID for σ_r is $11r$. Therefore, $\frac{\text{OPT\_FREE}(\sigma_r)}{\text{OPT}(\sigma_r)} \geq \frac{\text{OPT\_FREE}(\sigma_r)}{\text{PAID}(\sigma_r)} = \frac{12}{11}$. ◀

▶ **Theorem 14.** *Let* ALG_FREE *be a free move algorithm such that* $\text{ALG\_FREE}(\sigma) \leq \alpha \cdot \text{OPT}(\sigma) + \eta$. *For any* η *not dependent on* σ, $\alpha \geq 12/11$.

Proof. Towards a contradiction, assume $\alpha = \frac{12}{11} - \varepsilon$ for some $\varepsilon > 0$. Hence, $\text{ALG\_FREE}(\sigma_r) \leq \left(\frac{12}{11} - \varepsilon\right) \text{OPT}(\sigma_r) + \eta$. Solving for ε and σ_r such that $\text{ALG\_FREE}(\sigma_r) > \eta$,

$$\varepsilon \;\leq\; \frac{12}{11} - \frac{\text{ALG\_FREE}(\sigma_r) - \eta}{\text{OPT}(\sigma_r)} \;\leq\; \frac{12}{11} - \frac{\text{ALG\_FREE}(\sigma_r) - \eta}{\text{PAID}(\sigma_r)} \;\leq\; \frac{\eta}{\text{PAID}(\sigma_r)} \tag{13}$$

by the fact that $\text{PAID}(\sigma_r) \geq \text{OPT}(\sigma_r)$ and Lemma 13. Since η does not depend on r, and $\text{ALG\_FREE}(\sigma_r)$ and $\text{PAID}(\sigma_r)$ both increase with r, we have a contradiction by choosing a sufficiently large r such that $\text{ALG\_FREE}(\sigma_r) > \eta$ and (13) no longer holds. ◀

4 Conclusions

We showed that the difference in the performance between an offline optimal algorithm restricted to free exchanges and an unrestricted offline optimal algorithm is at least a multiplicative factor of 12/11, answering a question that has been open since 1996 [10].

Based on computer simulations, we believe that it should be possible to generalize the construction presented here and, based on this generalization, improve the lower bound to $3 - \sqrt{3}$.

Further, it would be interesting to consider upper bounds, in particular, an (offline) algorithm restricted to free exchanges that improves upon the 1.6 upper bound that follows from the randomized online algorithm COMB [2].

We note that the currently best known online algorithms use only free exchanges (cf. [8]). Our results bring up the possibility that improving the currently best randomized competitive ratio for the list update problem might necessitate introducing paid exchanges into the algorithm. The same might apply also to offline approximation algorithms.

Acknowledgements The authors would like to thank Amos Fiat, Rob van Stee, and Uri Zwick for useful discussions. We also wish to thank the anonymous reviewer who pointed out a minor change allowing an improvement of the lower bound from 13/12 to 12/11.

References

1 Susanne Albers. Improved randomized on-line algorithms for the list update problem. In Kenneth L. Clarkson, editor, *SODA*, pages 412–419. ACM/SIAM, 1995.

2 Susanne Albers, Bernhard von Stengel, and Ralph Werchner. A combined bit and timestamp algorithm for the list update problem. *Inf. Process. Lett.*, 56(3):135–139, 1995.

3 Christoph Ambühl. Offline list update is np-hard. In Mike Paterson, editor, *ESA*, volume 1879 of *Lecture Notes in Computer Science*, pages 42–51. Springer, 2000.

4 Christoph Ambühl. *On the List Update Problem.* PhD thesis, ETH Zürich, 2002.

5 Jon Louis Bentley, Daniel Dominic Sleator, Robert Endre Tarjan, and Victor K. Wei. A locally adaptive data compression scheme. *Commun. ACM*, 29(4):320–330, 1986.

6 Torben Hagerup. Online and offline access to short lists. In Ludek Kucera and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2007, 32nd International Symposium, MFCS 2007, Ceský Krumlov, Czech Republic, August 26-31, 2007, Proceedings*, volume 4708 of *Lecture Notes in Computer Science*, pages 691–702. Springer, 2007.

7 Sandy Irani. Two results on the list update problem. *Inf. Process. Lett.*, 38(6):301–306, 1991.

8 Shahin Kamali and Alejandro López-Ortiz. A survey of algorithms and models for list update. In Andrej Brodnik, Alejandro López-Ortiz, Venkatesh Raman, and Alfredo Viola, editors, *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, volume 8066 of *Lecture Notes in Computer Science*, pages 251–266. Springer, 2013.

9 Nick Reingold and Jeffery Westbrook. Off-line algorithms for the list update problem. Technical Report YALEU/DCS/TR-805, Yale University, 1990. http://cpsc.yale.edu/sites/default/files/files/tr805.pdf.

10 Nick Reingold and Jeffery Westbrook. Off-line algorithms for the list update problem. *Inf. Process. Lett.*, 60(2):75–80, 1996.

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

12 Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.