Paid Exchanges are Worth the Price ☆,☆☆

Alejandro López-Ortiz[a], Marc P. Renault[b,1], Adi Rosén[c]

[a] *University of Waterloo, Canada*
[b] *University of Wisconsin – Madison, USA*
[c] *CNRS and Université Paris Diderot, France*

Abstract

We consider the list update problem as defined in the seminal work on competitive analysis by Sleator and Tarjan [13]. An instance of the problem consists of a sequence of requests to access items in a linked list. After an item is accessed, that item can be moved to any position forward in the list at no cost (a move called free exchange), and, at any time, any two adjacent items can be swapped at a cost of 1 (a move called paid exchange). The cost to access an item is equal to 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 the question of the asymptotic relative power of the two models which has been open since Reingold and Westbrook [11] showed in 1996 that Sleator and Tarjan erred in [13] when they claimed that the two models are equivalent.

Keywords: List update problem, online computation, online algorithms, competitive analysis, lower bounds

1. Introduction

The *list update problem* is one of the basic problems studied by Sleator and Tarjan in their seminal paper that introduced competitive analysis [13]. It 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

☆A preliminary version of this paper appeared in the Proc. of the 32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015); LIPIcs 30, 636–648, 2015.

☆☆Research by the second and third authors was supported in part by ANR project NeTOC.

Email addresses: alopez-o@uwaterloo.ca (Alejandro López-Ortiz), mrenault@cs.wisc.edu (Marc P. Renault), adiro@liafa.univ-paris-diderot.fr (Adi Rosén)

[1] The work was performed while the second author was at Université Paris Diderot, France.

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 only two problems studied in the fundamental work on competitive analysis of Sleator and Tarjan [13] (the other being the paging problem). It is a fundamental problem in the area of algorithms that has been intensely studied, particularly, due to its importance for compression algorithms [5]. For a recent survey on the list update problem, see [9].

The question of the relative power of the two models (free and paid exchanges versus free exchanges only) has been open for some 20 years now, and has a number of important consequences. First, understanding the exact structure of the optimal (offline) schedule often yields insights useful for improving the online algorithms. Moreover, for the particular problem at hand, the question of the existences of efficient optimal offline algorithms (i.e., whether the problem is NP-hard or not) seems to be sensitive to the exact mode (it is known to be NP-hard in one, and not known if so for the other).

In [13], 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, 8], 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 [12]. 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 [12] with a probability of 4/5 and the non-parameterized, deterministic online algorithm TIMESTAMP [1] with a probability of 1/5. The best known randomized online lower bound is 1.50115 [4]. It should be noted that all the best known online algorithms use only free exchanges [9].

The offline problem is known to be NP-hard [3]. It is not known if this holds if only free exchanges are permitted. In [11], 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.[2] Based on the work of [11], 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].

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

1.1. Free vs. Paid Exchanges

In [13], Sleator and Tarjan claim that any schedule that uses paid exchanges and free exchanges can be converted into a schedule 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 $1, 2, 3$ [11]. An optimal algorithm serves this sequence at a cost of 8 by moving item 1 to the back of the list with paid exchanges at a cost of 2, and then serving the sequence at a cost of 6, without further changing the list. From an enumeration of all possible schedules that use only free exchanges, 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 [11]. 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 a multiplicative 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. However, the question of the relative power of the two models remained open, and in particular it was not known if there exists a multiplicative gap between the performance of the optimal offline algorithm with only free exchanges, and that of the optimal offline algorithm with paid and free exchanges (as only an additive gap was shown by Reingold and Westbrook [11]).

1.2. Our Contribution

In this work, we compare the cost of an optimal algorithm that can only perform free exchanges, denoted in the present paper by OPT_FREE, and an optimal algorithm that can use both paid and free exchanges, denoted in the present paper by OPT. We show that there is a lower bound 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 the question that has been open since Reingold and Westbrook [11] gave the counterexample to the claim of Sleator and Tarjan.

We note that all the online algorithms for the list update problem that achieve the best known competitive ratios use only free exchanges [9]. Thus, the multiplicative gap that we show between the performance of OPT_FREE and OPT suggests that, in order to achieve better randomized upper bounds, it may be necessary to consider online algorithms that make use of paid exchanges.

2. Preliminaries

As defined in the introduction, the *list update problem* consists of a linked list of ℓ items and a finite request sequence of items to access. The cost to access an item at position i is i. Immediately after the requested item has been accessed, it can be moved to any position forward in list (i.e. closer to the head) at no cost. This is called a *free exchange*. In addition, at any time, two adjacent items can be swapped at a cost of 1. This is called a *paid exchange*. The goal is to minimize the cost of accesses and paid exchanges by dynamically rearranging the list 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 a given algorithm ALG and a request sequence σ, we denote the cost to ALG to serve σ by $\text{ALG}(\sigma)$.

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 [10, 11], 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. [10][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 [10, 11], 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 positions immediately after x, such that the relative order of the items in the subset is maintained.

Theorem 3. [11][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 to the same item, 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 were to move x to β immediately after the first request to x, it would not increase its cost. This holds for both offline and online algorithms, but online algorithms generally are not able to 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 exchanges, 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 exchange optimal offline algorithm as compared to the unrestricted optimal offline algorithm. That is, we are comparing the power of paid exchanges and free exchanges together 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 (see Section 3.2 for the formal definition of PAID), increases with r, and such that

$$\frac{\text{OPT\_FREE}(\sigma_r)}{\text{OPT}(\sigma_r)} \geq \frac{\text{OPT\_FREE}(\sigma_r)}{\text{PAID}(\sigma_r)} \geq \frac{12}{11} \ .$$

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. For a given initial list configuration L, we define the request sequence $R(L)$ and a certain (not necessarily optimal) deterministic offline algorithm PAID that uses paid exchanges. By relabeling 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 relabeling of the list 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.

3.1. 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 sequences $R(L)$ that are defined using a relabeling of the list of PAID after each $R(L)$.

First, we prove two claims related to the short request sequence $R(L)$. Namely, we show that PAID serves $R(L)$, starting with list configuration L, at a cost of 11, and we show that any OPT_FREE that serves $R(L)$, starting with list configuration L, has a cost of 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.

Next, we concatenate these short request sequences to create a long request sequence. It is important to note that a multiplicative gap does not automatically 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 a 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 repetitions of $R(L)$). To overcome this difficulty, we prove that, for the long sequences, σ_r, any OPT_FREE must reach the same configuration as PAID does at the end of each $R(L)$. From this, we can conclude that for σ_r the cost of PAID is $11r$ and the cost of any OPT_FREE is at least $12r$.

We note that the $\langle 3, 2, 2, 3\rangle$ sequence of Reingold and Westbrook [11] (denoted henceforth σ_{RW}) does not lend itself as a building block for such concatenations and proof. Indeed, while the cost of OPT_FREE on σ_{RW} is 9, an optimal cost of 8 (with paid exchanges) can only be achieved if the final configuration is $2, 3, 1$. Trying to repeat the same line of proof by concatenating σ_{RW} sequences, with relabeling based on the initial configuration $2, 3, 1$, results (for two sequences σ_{RW}) with the sequence $\langle 3, 2, 2, 3, 1, 3, 3, 1\rangle$. This sequence can however can be served, with free exchanges only, at the cost of 16.[3] Furthermore, computer calculations using dynamic programming show that longer concatenations of σ_{RW} with relabeling result in sequences with only a difference of 1 between the optimal cost and the optimal cost with only free exchanges.

3.2. Offline Paid Exchange Algorithm

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

Let PAID be an unrestricted offline algorithm for $R(L)$ defined as follows. Before the first request of $R(L)$, using two paid exchanges, y and z are moved to the front of the list. This moves x to the tail of the list. Then, immediately before the any ν^3, $\nu \in \{y, z\}$, PAID moves x to the front, using paid exchanges.

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

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

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

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

[3] Using MTF on the first sequence has a cost of 9 and reaches a list configuration of $3, 2, 1$. On the first request in the second repetition, one moves 1 ahead of 2 for a list configuration of $3, 1, 2$ and serves the remaining requests from this configuration. The cost for the second repetition is 7 for a total of 16.

3.3. Arbitrarily Long Request Sequences

For an initial list configuration of $L = x, y, z$, from Fact 6, the configuration of the list of PAID after serving $R(L)$ is z, y, x. Therefore, after serving $R(L)$, with a relabeling 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 $L' = z, y, x$ (as is the list configuration PAID after serving $R(L)$ for $L = x, y, z$), then $R(L') = \langle x, y^3, x^3 \rangle$.

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.

3.4. 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 $L = x, y, z$, after serving $R(L)$, the list configuration of* MTF *is z, y, x.*

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

Lemma 9. *For an initial list configuration $L = x, y, z$,* 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 that access. They are x, y, z; x, z, y; and z, x, y (the last configuration corresponds to that of MTF). By Theorem 4, applied to the suffix of $R(L)$ after serving the first request of $R(L)$ (i.e., the subsequence $\langle y^3, z^3 \rangle$) every OPT_FREE moves y and z to the front of the list on the first request to each item. This means that OPT_FREE is fixed for that suffix. Table 1 summarizes the costs of the 3 possible ways in which OPT_FREE can serve $R(L)$, as a function of the list configuration after the first request. The actions of MTF on $R(L)$ correspond to the z, x, y 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.

3.5. 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, that we call a *treble* request, to x in $R_r(L_r)$ of σ_r, i.e. in the last round of σ_r. Observe that this lemma applies only to the last round of σ_r.

Table 1: For an initial list configuration of $L = x, y, z$, this table summarizes the potential optimal free exchange algorithms for $R(L) = \langle z, y^3, z^3 \rangle$. 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		x, y, z	x, z, y	z, x, y
1	z	3	3	3
2	y	2	3	3
3	y^2	2	2	2
5	z	3	3	2
6	z^2	2	2	2
Total:		12	13	12

Lemma 10. *For $\sigma_r = \langle R_1(L_1), \ldots, R_r(L_r) \rangle$, every* OPT_FREE *moves any item ν to the front of the list immediately after the first access of a treble request to ν in $R_r(L_r)$, where $L_1 = x, y, z$ 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 $L_r = x, y, z$ and let A be an arbitrary OPT_FREE algorithm. By way of contradiction, assume that A does not move some $\nu \in \{y, z\} \in R_r(L_r)$ to the front immediately after the access to ν of $\nu^3 \in R_r(L_r)$.

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

Since x 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, given the list configuration of both A and $\hat{A}$ after serving σ', $\hat{A}$ is OPT_FREE over the remainder of the sequence and A is strictly worse than OPT_FREE over the remainder of the sequence. Hence, $\hat{A}(\sigma'') = \text{OPT\_FREE}(\sigma'') < A(\sigma'')$. 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. □

3.6. The Rest of σ_r

In the next lemma, we prove that the property shown in Lemma 10 for the last round of σ_r holds 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 treble request. This is the main technical result, and the proof of the lemma is by reverse induction over the treble requests, where, for each step, we show that a free-move algorithm that moves the treble-requested item to the front has a cost that is no more than the cost of a free-move algorithm that does not move the item to the front.

Seemingly, the next lemma is a strengthening of Theorem 4; this is however not the case. The next lemma states that, for specific types of request sequences, there exists an OPT_FREE that moves the requested item to the front on the first request of any treble request. It is interesting to note that, perhaps counterintuitively, it is sometimes to the disadvantage of an algorithm that can only use free exchanges to move to the front at the start of a treble request. For example, it can be verified by enumerating all the free-move schedules 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, whereas, if ALG_FREE moves item 5 to the front immediately after the first request to that item, 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 ν to the front of the list immediately after the first access of a treble request to ν, where $L_1 = x, y, z$ 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 ν to the front of the list immediately after the first access of a treble request to ν, and $A'(\sigma_r) \leq A(\sigma_r)$. 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 $\nu^3 \in R_j(L_j) \in \sigma_r$, assuming that $L_j = x, y, z$. That is, we consider the rounds from $R_r(L_r)$ to $R_1(L_1)$ and the consecutive treble requests in each round from the last treble request to the first. Assume that $L_j = x, y, z$, then, for each ν^3, if A_q does not move ν to the front immediately after the first request, we define A_{q+1}, based on A_q, such that the desired property holds for ν^3 and all subsequent consecutive treble request, and we show that the cost of A_{q+1} does not increase as compared to the cost of A_q.

In the proof, we use the following notations. Let ν^3 be a treble request in $R_j(L_j)$, assuming $L_j = x, y, z$, for which A_q does not have the desired property. We will denote all the requests in σ_r before ν^3 by σ_1. The requests of σ_r after ν^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 a number of subsequences $\langle \sigma_3, \ldots, \sigma_p\rangle$ such that $\sigma_r = \langle \sigma_1, \nu^3, \sigma_3, \ldots, \sigma_p\rangle$. 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 $\text{ALG}(\sigma_r) = \text{ALG}(\sigma_1) + \text{ALG}(\nu^3) + \text{ALG}(\sigma_3) + \cdots + \text{ALG}(\sigma_p)$. Further note that by Theorem 5, we can assume without loss of generality that A_q does not move ν further ahead in the list on the second or third requests of ν^3.

Definition of $\hat{A}_q$. We first define an algorithm $\hat{A}_q$ that we use extensively in the proof. For $\sigma = \langle \sigma_1, \nu^3, \sigma_2 \rangle$ 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 ν from position $\alpha > 1$ to the front of the list immediately after the first request of ν. Immediately after serving ν, the configuration of the list of A_q is some B, ν, C and the configuration of the list of $\hat{A}_q$ is ν, B, C, where B is the set of items ahead of ν in the configuration of A_q at this time and C is the set of items behind ν in the configuration of A_q. As long as the list configurations of A_q and $\hat{A}_q$ differ, for each $v \in \sigma_2$, if A_q moves v to the front, $\hat{A}_q$ moves v to front. Otherwise, $\hat{A}_q$ does not move v forward at all. Once the list configurations of A_q and $\hat{A}_q$ match, $\hat{A}_q$ serves the remaining requests exactly as A_q. Note that it is possible that the list configuration of $\hat{A}_q$ will 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.

$$\hat{A}_q(\sigma_1) = A_q(\sigma_1) \ , \tag{1}$$

$$1 \leq |B| \leq 2 \ , \tag{2}$$

$$|C| = 2 - |B| \ , \tag{3}$$

$$\beta = |B| + 1 \ , \tag{4}$$

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

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

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

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

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 $R_j(L_j)$: x, y, z and z, y, x. Therefore, y is requested in every $R_j(L_j)$, and z and x are requested in alternating $R_j(L_j)$'s.

For $\nu \in \{y, z\} \in R_j(L_j)$, which is the last point in σ_r for which A_q does not move ν to the front immediately after the first request of a treble request, we can distinguish between three cases: (1) ν is never requested again in σ_r; (2) ν is requested again in $R_{j+1}(L_{j+1})$, i.e. in the next round; (3) ν 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: $\nu \in R_j(L_j)$ is never requested again in σ_r.

Recall that $\sigma_r = \langle \sigma_1, \nu^3, \sigma_2 \rangle$. 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 $L_{r-1} = x, y, z$ and, hence, $R_{r-1}(L_{r-1}) = \langle z, y^3, z^3 \rangle$. For $L_{r-1} = x, y, z$, $R_r(L_r) = \langle x, y^3, x^3 \rangle$ and z is not requested in $R_r(L_r)$.

Denote $\sigma_1 = \langle R_1(L_1), \ldots, R_{r-2}(L_{r-2}), z, y^3 \rangle$,
$\sigma_2 = \langle x, y^3, x^3 \rangle$.

Cost for $\sigma_2 = \langle x, y^3, x^3 \rangle$. By the induction hypothesis we know that A_q moves y to the front of the list on the first request to y in σ_2, and moves x to the front of the list on the second request to x in σ_2. It follows that the configurations of the lists of $\hat{A}_q$ and A_q match before the third request to x is processed.

If x is in B, then the total cost to access x 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 x before A_q must move x to the front, according to the induction hypothesis and, by the definition of B, if x is in B, then $\hat{A}_q$ has y in front of x, whereas A_q does not.

If x is in C, the total cost to access x 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 x in σ_2, A_q were to move x between y and z in its list. Then, on the second access, x 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 y to the front on the first request to y in σ_2. Therefore, if y is in B, the first access costs 1 more to $\hat{A}_q$ as compared to A_q and, if y is in C, the cost for the first access is the same for both $\hat{A}_q$ and A_q.

Note that, for every combination of x and y in B or C, the additional cost to $\hat{A}_q$ as compared to A_q for σ_2 can be bounded by $2|B| + |C| - 1$. This gives that for σ_2,

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

where the last equality follows by applying (3).

Using (1), we get

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

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

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

Define $\sigma_1 = \langle R_1(L_1), \ldots, R_{j-1}(L_{j-1}), z\rangle$,
$\sigma_3 = \langle z^3, x, y^3\rangle$,
$\sigma_4 = \langle x^3, R_{j+2}(L_{j+2}), \ldots, R_r(L_r)\rangle$.
Note that $\sigma_2 = \langle \sigma_3, \sigma_4\rangle = \langle z^3, R_{j+1}(L_{j+1}), \ldots, R_r(L_r)\rangle$.

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

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

List Configuration after σ_3. By the induction hypothesis, A_q moves z and y 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 x to the front. If A_q does not move x to the front of the list, the configuration of its list will be y, z, x, and, by the definition of $\hat{A}_q$, the configuration of the list of $\hat{A}_q$ will also be y, z, x. If A_q does move x to the front of the list, the configuration of its list will be y, x, z, and, by the definition of $\hat{A}_q$, the list configuration of $\hat{A}_q$ will also be y, x, z.

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

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

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

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

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

We define A_{q+1} to be $\hat{A}_q$. For $L_j = x, y, z$, we have $R_j(L_j) = \langle z, y^3, z^3\rangle$, $R_{j+1}(L_{j+1}) = \langle x, y^3, x^3\rangle$, and $R_{j+2}(L_{j+2}) = \langle z, y^3, z^3\rangle$. Recall that $\sigma_r = \langle \sigma_1, z^3, \sigma_2\rangle$.
Define $\sigma_1 = \langle R_1(L_1), \ldots, R_{j-1}(L_{j-1}), z, y^3\rangle$,
$\sigma_3 = \langle x, y^3, x^3\rangle$, 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 $\sigma_3 = \langle x, y^3, x^3 \rangle$. After serving $\langle \sigma_1, z^3 \rangle$, the configuration of the list of $\hat{A}_q$ is z, B, C and the configuration of the list of A_q is B, z, C. By the induction hypothesis, A_q moves x to the front of the list on the second request to x in σ_3 and moves y to the front of the list on the first request to y in σ_3. This is exactly the same scenario as σ_2 for Case 1. Similarly to (8) for Case 1 we have,

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

List Configuration after σ_3. Since both x and y 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 $\hat{A}_q$ after serving σ_3 is x, y, z.

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, according to the definition of $\hat{A}_q$,

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

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

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

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 $A'(\sigma_r) = A_q(\sigma_r) \leq \cdots \leq A_0(\sigma_r) = A(\sigma_r)$ which concludes the proof. □

For σ_r as defined above, Lemma 11 shows that there exists an OPT_FREE that moves ν to the front when ν 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 $L = x, y, z$, Lemma 9 shows that the algorithm MTF is an optimal free move algorithm for $R(L)$. Since the list configuration of MTF after serving $R_1(L_1), \ldots, R_j(L_j)$, $1 \leq j \leq r$, is the same as that of 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* $L_1 = x, y, z$ *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. By Lemma 11, there exists an OPT_FREE that has 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 MTF(σ_r) = OPT_FREE(σ_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 $L_1 = x, y, z$ 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 $L_1 = x, y, z$ 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 an algorithm that only makes free exchanges and that* $\text{ALG\_FREE}(\sigma) \leq \alpha \cdot \text{OPT}(\sigma) + \eta$ *for any finite request sequence σ. For any η not dependent on σ, $\alpha \geq 12/11$.*

Proof. Given any $\eta \geq 0$ not dependent on σ, assume towards a contradiction that $\alpha = \frac{12}{11} - \varepsilon$ for some $\varepsilon > 0$. Hence, in particular for the sequences σ_r defined in our proof, we have $\text{ALG\_FREE}(\sigma_r) \leq \left(\frac{12}{11} - \varepsilon\right) \text{OPT}(\sigma_r) + \eta$. Because $\text{OPT}(\sigma_r)$ is increasing with r, we can pick a large enough r^* such that $\text{OPT}(\sigma_{r^*}) > \eta/\varepsilon$. We have

$$\frac{12}{11} \cdot \text{OPT}(\sigma_{r^*}) \leq \text{OPT\_FREE}(\sigma_{r^*}) \leq \text{ALG\_FREE}(\sigma_{r^*}) \leq (\frac{12}{11} - \varepsilon) \cdot \text{OPT}(\sigma_{r^*}) + \eta \ ,$$

where the first inequality follows from Lemma 13, and the last inequality follows by the assumption above. It follows that $\text{OPT}(\sigma_{r^*}) \leq \eta/\varepsilon$, a contradiction to the choice of r^*. □

4. Conclusions

We have shown that the gap between the performance of the offline optimal algorithm restricted to free exchanges and that of the unrestricted offline optimal algorithm is at least a multiplicative factor of 12/11.

We note that all the currently known online algorithms for the list update problem with best known competitive ratios use only free exchanges (cf. [9]). 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.

Further to the present work, it would be interesting to also consider upper bounds on the gap between the performance of an optimal (offline) algorithm restricted to free moves, and an unrestricted optimal algorithm, that is, to determine the exact worst case ratio between the performances of the two optimal algorithms. One important step towards this goal would be to find an (offline) algorithm restricted to free exchanges that improves upon the 1.6 upper bound that follows from the randomized online algorithm COMB [2].

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 STACS reviewer who pointed out a minor change allowing an improvement of the lower bound from 13/12 to 12/11.

[1] Albers, S., 1995. Improved randomized on-line algorithms for the list update problem. In: Clarkson, K. L. (Ed.), SODA. ACM/SIAM, pp. 412–419.

[2] Albers, S., von Stengel, B., Werchner, R., 1995. A combined bit and timestamp algorithm for the list update problem. Inf. Process. Lett. 56 (3), 135–139.

[3] Ambühl, C., 2000. Offline list update is np-hard. In: Paterson, M. (Ed.), ESA. Vol. 1879 of Lecture Notes in Computer Science. Springer, pp. 42–51.

[4] Ambühl, C., 2002. On the list update problem. Ph.D. thesis, ETH Zürich.

[5] Bentley, J. L., Sleator, D. D., Tarjan, R. E., Wei, V. K., 1986. A locally adaptive data compression scheme. Commun. ACM 29 (4), 320–330.

[6] Hagerup, T., 2007. Online and offline access to short lists. In: Kucera, L., Kucera, A. (Eds.), Mathematical Foundations of Computer Science 2007, 32nd International Symposium, MFCS 2007, Ceský Krumlov, Czech Republic, August 26-31, 2007, Proceedings. Vol. 4708 of Lecture Notes in Computer Science. Springer, pp. 691–702.
URL http://dx.doi.org/10.1007/978-3-540-74456-6_61

[7] Irani, S., 1991. Two results on the list update problem. Inf. Process. Lett. 38 (6), 301–306.

[8] Irani, S., 1996. Corrected version of the split algorithm for the list update problem. Tech. Rep. 96-53, ICS Department, University of California, Irvine.

[9] Kamali, S., López-Ortiz, A., 2013. A survey of algorithms and models for list update. In: Brodnik, A., López-Ortiz, A., Raman, V., Viola, A. (Eds.), Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday. Vol. 8066 of Lecture Notes in Computer Science. Springer, pp. 251–266.
URL http://dx.doi.org/10.1007/978-3-642-40273-9_17

[10] Reingold, N., Westbrook, J., 1990. Off-line algorithms for the list update problem. Tech. Rep. YALEU/DCS/TR-805, Yale University, http://cpsc.yale.edu/sites/default/files/files/tr805.pdf.

[11] Reingold, N., Westbrook, J., 1996. Off-line algorithms for the list update problem. Inf. Process. Lett. 60 (2), 75–80.

[12] Reingold, N., Westbrook, J., Sleator, D. D., 1994. Randomized competitive algorithms for the list update problem. Algorithmica 11 (1), 15–32.

[13] Sleator, D. D., Tarjan, R. E., 1985. Amortized efficiency of list update and paging rules. Commun. ACM 28 (2), 202–208.